



**Universiteit  
Leiden**  
The Netherlands

# Opleiding Informatica

Autonomous navigation of the spherical robot BB-8

Renzo Scholman

Supervisors:

Dr. Erwin M. Bakker & Dr Michael S.K. Lew

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)

[www.liacs.leidenuniv.nl](http://www.liacs.leidenuniv.nl)

20/08/2018

## Abstract

This paper focuses on the parts of the design of a navigation architecture for a life-size spherically shaped robot with an internal drive platform called BB-8, along with the hardware and software implementation that is needed to control this robot. It will, eventually, be used as a tour guide during open days in the building of the The Hague University of Applied Science (THUAS) in Zoetermeer. The subject that will be researched is: what navigation architecture, consisting of several sensors, will result in the best autonomous navigation in a controlled environment for a spherical robot. Background information about the robot, localization, mapping, simultaneous localization and mapping, and path planning is given, with the focus on the robot and localization and the rest for future work. All the processing, logic and controlling the movement of the robot is done with the aid of a framework named the Robot Operating System (ROS). This framework enables its users to quickly create multi-threaded programs which are able to communicate with each other via a high speed TCP/IP network. It is essential to have a good understanding of this framework, ROS, to correctly understand the implementation of the described navigation architecture. Various levels in the proposed navigation architecture have been defined of which a few have been implemented, focusing on internal sensors, with the rest being proposed for future work. Per extra sensor an increase in autonomous capabilities is expected. The implemented levels consist of the odometry from the wheel base as well as an inertial measurement unit (IMU) to compensate the rotations of the robot due to its spherical shape as well as errors in the movement. Each of the levels of the navigation architecture has been compared to one another with the results from an experimentation in which the autonomous navigation was tested by means of dead reckoning. Concluded was that the use of three omni wheels on an internal drive mechanism was found to be usable together with a sphere consisting of two halves. Furthermore, the robot was found to have much better autonomous navigation capabilities after the inclusion of the orientation sensor, the IMU, as was expected. This is due to the instable nature of its spherical shape resulting in the shell turning easily whilst driving.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	The robot . . . . .	2
1.2	Research goal . . . . .	2
1.3	Organization of this paper . . . . .	3
<b>2</b>	<b>Related Work</b>	<b>5</b>
2.1	Tour-guide robots . . . . .	5
2.2	Spherical robots . . . . .	6
<b>3</b>	<b>Background</b>	<b>8</b>
3.1	Driving and shell . . . . .	8
3.2	Head and head mount . . . . .	10
<b>4</b>	<b>Localization</b>	<b>12</b>
4.1	Markov Localization . . . . .	13
4.2	Particle filter . . . . .	16
4.3	Kalman filter . . . . .	19
4.4	Extended Kalman filter . . . . .	21
4.5	Comparison and recommendation . . . . .	21
<b>5</b>	<b>Mapping</b>	<b>23</b>
5.1	Indoor structured environments . . . . .	24
5.1.1	Occupancy grids . . . . .	24
5.1.2	Line maps . . . . .	26
5.1.3	Topological maps . . . . .	27
5.1.4	Landmark-based maps . . . . .	28
5.2	Natural environments . . . . .	28
5.3	Recommendation . . . . .	29
<b>6</b>	<b>Simultaneous Localization and Mapping</b>	<b>31</b>
6.1	Taxonomy of SLAM . . . . .	31
6.2	Loop Closures . . . . .	32
<b>7</b>	<b>Motion Planning</b>	<b>34</b>
7.1	Combinatorial planning . . . . .	36
7.1.1	Visibility graphs . . . . .	37
7.1.2	Exact cell decomposition . . . . .	38
7.1.3	Approximate cell decomposition . . . . .	39
7.2	Sample based planning . . . . .	40
7.2.1	Probabilistic Road Maps . . . . .	40

7.2.2	Rapidly Exploring Random Trees . . . . .	41
7.3	Recommendation . . . . .	42
<b>8</b>	<b>Robot Operating System</b>	<b>43</b>
8.1	Overview . . . . .	43
8.2	Communication . . . . .	44
<b>9</b>	<b>Navigation Architecture</b>	<b>46</b>
9.1	Wheel encoders . . . . .	47
9.1.1	Four omni-wheel setup . . . . .	48
9.1.2	Alternative three omni-wheel setup . . . . .	50
9.1.3	Calibration . . . . .	52
9.2	Inertial measurement unit . . . . .	54
9.2.1	Madgwick . . . . .	55
9.2.2	Sensor fusion . . . . .	55
9.2.3	Calibration . . . . .	56
<b>10</b>	<b>Evaluation of dead reckoning capabilities</b>	<b>59</b>
10.1	Experimental test setups . . . . .	59
10.2	Evaluation of experimental test setups . . . . .	60
10.3	Results . . . . .	61
10.3.1	Straight lines . . . . .	61
10.3.2	Turning . . . . .	66
10.3.3	General remarks and four wheeled setup . . . . .	70
<b>11</b>	<b>Conclusions</b>	<b>71</b>
<b>12</b>	<b>Future navigation architecture</b>	<b>72</b>
12.1	Bluetooth positioning . . . . .	72
12.1.1	Retreiving RSSI . . . . .	73
12.1.2	Determine position . . . . .	73
12.2	Radar collision detection . . . . .	75
12.3	RGB-D Camera . . . . .	76
12.4	Future evaluation . . . . .	78
<b>13</b>	<b>Discussion</b>	<b>79</b>
<b>A</b>	<b>BB-8 technical details</b>	<b>80</b>
A.1	Shell . . . . .	80
A.2	Motors and servo's . . . . .	81
A.3	Hardware and electrical components . . . . .	82
<b>B</b>	<b>Implementation into ROS</b>	<b>85</b>
B.1	Packages . . . . .	85
B.2	ROS computation graphs . . . . .	87
	<b>Bibliography</b>	<b>88</b>

# List of Tables

2.1	Tour guide robot overview per location, updated version of [1]	6
4.1	KF vs EKF Differences	21
4.2	Comparison of localization techniques	22
9.1	Calibration parameters	54
9.2	Performance of different grades of inertial sensors. [2]	55
10.1	Evaluation parameters	60
10.2	Results straight line 1m odometry only	63
10.3	Results straight line 1m IMU + odometry	63
10.4	Results straight line 2m odometry only	64
10.5	Results straight line 2m IMU + odometry	64
10.6	Results straight line 3m odometry only	65
10.7	Results straight line 3m IMU + odometry	65
10.8	Results two straight lines 1m with 90 degree turn in between odometry only	67
10.9	Results two straight lines 1m with 90 degree turn in between IMU + odometry	67
10.10	Results two straight lines 2m with 90 degree turn in between odometry only	68
10.11	Results two straight lines 2m with 90 degree turn in between IMU + odometry	68
10.12	Results two straight lines 3m with 90 degree turn in between odometry only	69
10.13	Results two straight lines 3m with 90 degree turn in between IMU + odometry	69
B.1	List of packages used in combination with ROS	85

# List of Algorithms

1	Markov Localization pseudo code	14
2	Particle Filter Localization pseudo code	17
3	Kalman filter pseudo code	20
4	Extended Kalman filter pseudo code	21
5	posterior occupancy probability of cell $m_i$ , given observation $z_t$ in state $x_t$ [3]	25

6	Visibility graph path planning pseudo code . . . . .	37
7	Exact cell decomposition path planning pseudo code . . . . .	39
8	Approximate cell decomposition path planning pseudo code . . . . .	40
9	Probabilistic Road Map path planning pseudo code . . . . .	41
10	Rapidly Exploring Random Trees path planning pseudo code . . . . .	42
11	Odometry algorithm pseudo code . . . . .	48
12	Four wheel odometry . . . . .	49
13	Three wheel odometry . . . . .	52
14	Odometry calibration [4] . . . . .	53
15	Correction formula to calibrate and align IMU [5] . . . . .	57
16	RSSI Smoothing [6] . . . . .	73
17	Position smoothing [7] . . . . .	75

## List of Figures

1.1	Inner mechanism of BB-8 . . . . .	2
2.1	Characteristics of the developed spherical robots. D, G, and A indicate direct-driving, gravity, and angular momentum methods, respectively. The mark v indicates the robot is capable of performing the specific motion, and the mark v* indicates the motion is achievable in most situations (i.e., with singularity). [8] . . . . .	7
3.1	BB-8 drive mechanism . . . . .	8
3.2	BB-8 frame . . . . .	9
3.3	BB-8 head and head mount overview . . . . .	10
4.1	Markov localization example [9]. . . . .	15
4.2	Particle localization example [9]. . . . .	18
4.3	Illustration of Kalman filters. [9] . . . . .	19
5.1	Grid map visualization 5.1b of room 5.1a. [10] . . . . .	25
5.2	Line map visualization 5.2b of laser scan data 5.2a. [11] . . . . .	26
5.3	Example of a generalized Voronoi graph. [3] . . . . .	27
5.4	Visualization of a landmark-based map. [3] . . . . .	28
6.1	Loop closure example. [12] . . . . .	33
7.1	Visualization of obtaining configuration space from work space [13] . . . . .	35
7.2	Configuration space definitions [14] . . . . .	36
7.3	Visibility graph algorithm visualization [14] . . . . .	38

7.4	Exact cell decomposition algorithm visualization [13] . . . . .	38
7.5	Approximate cell decomposition algorithm visualization [13] . . . . .	39
7.6	Probabilistic Road Map algorithm visualization [15] . . . . .	41
7.7	Rapidly Exploring Random Trees algorithm visualization [15] . . . . .	42
8.1	ROS communication example [16] . . . . .	45
9.1	Driving mechanism of BB-8 . . . . .	47
9.2	Driving mechanism of BB-8 with three omni-wheels . . . . .	50
9.3	Graphic diagram for three wheeled kinematic model, adjusted from [17] for the current setup . . . . .	51
10.1	Test setup examples. . . . .	60
10.2	Visualization of Tables 10.2 and 10.3 for first test setup at length 1m . . . . .	63
10.3	Visualization of Tables 10.4 and 10.5 for first test setup at length 2m . . . . .	64
10.4	Visualization of Tables 10.6 and 10.7 for first test setup at length 3m . . . . .	65
10.5	Visualization of Tables 10.8 and 10.9 for second test setup at two times length 1m with 90 degree turn in between . . . . .	67
10.6	Visualization of Tables 10.10 and 10.11 for second test setup at two times length 2m with 90 degree turn in between . . . . .	68
10.7	Visualization of Tables 10.12 and 10.13 for second test setup at two times length 3m with 90 degree turn in between . . . . .	69
12.1	Modular Multi Layer Perceptron [18]. Left is the selector which selects which of the trained networks is used. Right are the networks with the input layers and the Acces Points (AP 1-3), the hidden layer and the Location ID which comes out of the output layer . . . . .	74
12.2	Radar based SLAM implementation [19] . . . . .	76
12.3	Occupancy grid visualization of obtained point cloud for lab [20] . . . . .	77
12.4	RGB-D base SLAM implementation [21] . . . . .	77
12.5	Future test setup examples. . . . .	78
A.1	BB-8 shell mock-up sketch . . . . .	81
A.2	Jetson TX1 Connections [22] . . . . .	83
A.3	BB-8 electrical diagram . . . . .	84
B.1	ROS computation graph for the first navigation architecture level . . . . .	87
B.2	ROS computation graph for the first navigation architecture level . . . . .	87

# Chapter 1

## Introduction

Everyone encounters them in any way, form or shape these days: robots. Scientists have been trying to make more and more advanced robots for decades now, but truly autonomous robots do not yet exist. The ultimate goal for the robot researched in this thesis is to be autonomous and able to give guided tours through an office building. Having a robot move through an office building is a big challenge with all of the moving objects and stationary objects which can be moved by people. This problem has been studied extensively by many, one of which is The Office Marathon [23] where they were able to let a robot drive a complete marathon in a real office building. Robots giving guided tours is also a subject that has been extensively studied, usually their goal was to give guided tours through a museum. One of the first robots that has been created whilst studying the field of autonomous robots is Rhino [24], which was created in 1997 to give guided tours through the "Deutsches Museum Bonn". Rhino was quickly succeeded a year later by Minerva [25]. Minerva was made by mostly the same creators of Rhino but this time it was set up to give guided tours in the Smithsonian National Museum of American History and was improved on many fronts. After these two groundbreakers, many others have studied the subject of guided tours by robots. Most recently the studies into tour-guide robots have become more in-depth, even using frameworks like the Robot Operating System (ROS) in the case of [26]. They have also been studied whilst using less powerful hardware like the Raspberry Pi 2 in the case of [27]. More recently with new, more advanced, algorithms and new hardware capable of parallel processing at a lower energy consumption a start to the solution of problems faced in dynamic environments has been found by [28]. They processed the data of an RGB-D camera to model the very dynamic, real life, world as a static environment by identifying and removing dynamic objects. Even neural networks have been applied to the problem of robotic navigation by [29] where researchers from DeepMind designed a neural network able to localize the robot and find a path to its destination.



## 1.1 The robot

The robot on which this paper focuses is a spherical robot resembling BB-8, a robot known from the movie Star Wars, The Force Awakens. Its total size is about 65cm in diameter for the body and 35 cm in diameter for the head. The movement comes from an internal drive mechanism which moves the outer shell. The concept of moving the robot from the inside was first introduced in a robot with two points anchored to the outer shell and one active wheel on the inside drive unit [30]. The head will not be used in conjunction with the spherical body during the evaluation. Adding the head during driving is to be part of future work. After more future work to expand the autonomous navigation capabilities, the robot should be able to give guided tours in a building. The building in question is a building from The Hague University of Applied Science (THUAS): The Dutch Innovation Factory (DIF) in Zoetermeer, The Netherlands. In the DIF potentially new students of THUAS should be guided around to give them a look and feel of what studying at THUAS would be like.

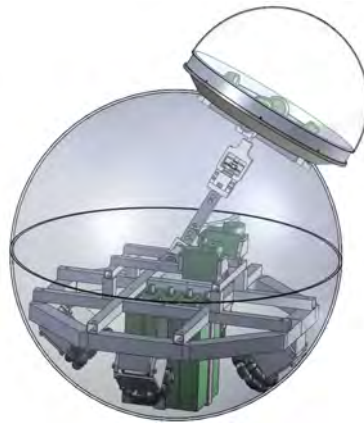


Figure 1.1: Inner mechanism of BB-8

## 1.2 Research goal

As a step towards the final goal of giving guided tours, the research goal will be focused on making a navigation architecture for an autonomous spherical robot. This proposed navigation architecture should be able to use dead reckoning to return the robot to its starting position. For any robot to be able to move autonomously it will need a way to sense the environment. To achieve this, the navigation architecture will be given to provide the robot with the necessary sensors to enable it to drive through a building without needing any adjustment or input from humans. This will be built up in various levels of the to be implemented navigation architecture, focusing on internal sensors only each level. These levels will be built up by first using only the wheel encoders followed by introducing an inertial measurement unit (IMU) to achieve the current research goal of enabling dead reckoning. All of the software needed will be written for the Robot Operating System (ROS) framework, which has a lot of already available drivers and controllers for various sensors; algorithms for path finding and much more. The navigation architecture will be tested and evaluated

by using dead reckoning to start from a home position, let a human guide the robot to a random position and then to let the robot find its home position again. This will then be evaluated by checking its speed, precision and level of difficulty (e.g. by going in multiple directions before letting it return).

### **1.3 Organization of this paper**

The related work will be described first in Chapter 2. In this chapter a lot of other tour guide robots will be described followed by spherical robots. Furthermore information about many possible drive mechanisms of spherical robots is given.

In Chapter 3 background information is given about the robot BB-8 itself. Details will be given about the sphere and the drive mechanism used within the sphere. Lastly information about the robots head is given, although the head is not used in the experimental setup, it is a part of the robot as a whole.

The first of the explained techniques used is localization, given in Chapter 4. In this chapter information is given about what problems are faced during the localization of a robot. Four different localization techniques are discussed, showing how they work and what their strengths and weaknesses are. Afterwards a comparison between the methods is given and one is advised for usage in the navigation architecture.

Chapter 5 about mapping methods, Chapter 6 about Simultaneous localization and mapping (SLAM) and Chapter 7 about motion planning give background information for the future work. For the navigation architecture as implemented and evaluated all of these chapters are not needed. An overview of the mapping problem is given first followed by multiple mapping methods, being divided into two- and three-dimensional methods. SLAM poses a solution for the problem of the dependence of the localization and mapping methods given in the previous chapters on one another to work correctly. A taxonomy of the SLAM problem is given followed by information about loop closures, which enables the detection of previously visited locations. For the motion planning chapter, first a general idea is given in order to define the problems faced. Two types of methods are then discussed, each with their own algorithms, which are capable of solving this problem. At the end of both the mapping and motion planning chapters a recommendation as to made for which method is to be used with the future work.

The framework named Robot Operating System (ROS), explained in Chapter 8, is only relevant for the reader when details of the implementation are to be read in the appendix. Firstly a quick overview of the framework is given detailing its key characteristics after which information about the communication between various subprograms of the robot is given.

Chapter 9 is about the various levels in the navigation architecture itself, which is the most important topic towards which all of the background information is pointed. It starts off with defining the various levels of the navigation architecture including the proposed levels for future work. The various levels are then explained on a per section basis. Firstly the wheel encoders which give information about the odometry is given, for both the four wheeled drive mechanism as well as the three wheeled drive mechanism. Lastly the second level

of the navigation architecture as implemented is given. This second level introduces an inertial measurement unit in order to add orientation information to the robots movements.

The evaluation and results are given in Chapter 10. Here the two implemented levels of the navigation architecture are tested by first introducing the reader to the two used experimental test setups. This is followed by the parameters with which each of the levels is evaluated. Next the results follow, being split up in a subsection per test setup. Lastly in the results some general remarks are given.

The conclusions from the experiments are given in Chapter 11. This chapter sheds more light on the comparison between the three wheeled setup and the four wheeled setup for the internal drive platform. Lastly the conclusions over the two tested levels of the navigation architecture are given, in order to show the found improvements to the reader.

Chapter 12 will go into detail about the levels of the navigation architecture as proposed for future work. Some work has already gone into the levels of the proposed future work, which is shown for each level. The levels are elaborated on in order on a per level basis as with the chapter about the already implemented and tested navigation architecture.

Lastly the discussion is given in Chapter 13. This chapter elaborates on the improvable parts of this paper, showing the shortcomings and possible improvements.

The appendices follow lastly, starting with technical information about the robot itself in Appendix A, followed by Appendix B, where information about the implementation of the navigation architecture levels into ROS is given. Firstly details and the creation of the shell are given followed by the used motors and servo's that make up the drive platform and head mount. Last in Appendix A the used hardware and electrical components are explained. In Appendix B the used and built packages are given first in a quick overview followed by a more elaborate description. Lastly the graphs showing the actual combination of packages used in each of the implemented levels of the navigation architecture is given.

# Chapter 2

## Related Work

In the case of BB-8 two areas of related work can be identified. In the first case this robot is to be a tour-guide robot, a much studied field. Secondly it is also a spherical robot, which can be built and operated in various different ways. Both of these subjects will, in the given order, be elaborated on in this chapter in their respective subsection.

### 2.1 Tour-guide robots

Over the years many tour-guide robots have been created and studied, beginning back in 1993 with very primitive tours by Polly [31]. To give a good overview of the tour guide robots that have been created over the years, check Table 2.1 where an overview is given of the most well-known tour-guide robots that exist and their location. This table is an extension of the work done by [1]. It has been altered to make it more up to date.

Most of the robots in the table have been described by [1]. A noteworthy, more recent, robot is Aggie [43]. This robot is a continued development of the robot by [1], which they gave more capabilities such as more languages that it can understand and speak. Another noteworthy of these is Kbot-1 [42], it was created back in 2004 by Neobotix but broke down two years later. Since then it stood in a garage until 2014 when it was taken over by members of the University of Basque Country (UPV/EHU) and the RSAIT group to be fixed and is one of the first to have the Robot Operating System (ROS) integrated. Besides these robots that actually perform real world tours a few more interesting studies have been done. Other tour-guide robots to have ROS integrated are [26], [27] and [58]. The study performed by [26] enabled a robot to learn a new tour whilst following a human tour guide. And more recently work by [27] enabled tour-guide robots powered by less powerful hardware like the Raspberry Pi 2. For the proposed navigation architecture [58] is one of the more important papers since they try to localize the robot with the use of various sensors. These sensors are then combined via a fusion algorithm to get a better understanding of the robots exact location.

Location type	Name	Location Name
Museum	Rhino [24]	Deutsches Museum Bonn
	Minerva [25]	Smithsonians National Museum of American History
	Sage [32]	Carnegie Museum of Natural History
	Chips [33]	Carnegie Museum of Natural History
	Care-O-Bot [34]	Museum für Kommunikation
	HERMES [35]	Heinz Nixdorf Museums Forum
	Jinny [36]	National Science Museum of Korea
	Robovie [37]	Osaka Science Museum
	Enon [38]	Kyotaro Nishimura Museum
	Urbano [39]	Principe Felipe Museum
	Indigo [40]	Cultural institute
	Cicerobot [41]	Archaeological Museum of Agrigento
	Kbot-1 (2004) [42]	Eureka Museum of Science
	Aggie [43]	Art Gallery of Western Australia
Exhibition hall	Robox [44]	Swiss National Exposition Expo.02
	Toyota [45]	Toyota Kaikan Exhibition Hall
	Lumen [46]	Electrical Engineering Days 2015
University	Polly [31]	MIT AI Lab 1993
	Virgil [47]	Rice University
	Bryn Mawr [48]	Bryn Mawr College
	NTU-I [49]	National Taiwan University
	Konrad, Suse [50]	Konrad Zuse building
	Nao [1]	Industrial Informatics and Robotics Laboratory
Various	GRACE [51]	AAAI Robot Challenge
	BIRON [52]	Home-tour guide
	Robotinho [53]	The 12 Cellists of the Berlin Philharmonic
	TOOMAS [54]	Germany stores
	SANCHO [55]	Live TV shows
	CATE [56]	Engineering and Technology building
	FROG [57]	Royal Alczar

Table 2.1: Tour guide robot overview per location, updated version of [1]

## 2.2 Spherical robots

Spherical robots are quite unstable due to their single point of contact with the ground. There are also various ways in which they can be driven, all differing hugely between each other. An overview of all of the drive mechanisms can be seen in Figure 2.1. This figure has been supplemented in the column for the hamster car with a  $v^*$  in the omnidirectional locomotion row, which was empty at first. This can be explained by looking at the SPHERICLE robot by [59] and the proposed BB-8 robot which both have omnidirectional capabilities despite having a hamster car mechanism. [8] created a spherical robot which was remotely controlled by a human, in their experiments they moved the robot about 1m forward measuring its lateral displacement and angular deviation. They achieved a displacement of 4.1cm and a deviation of 13 degrees, reaching a maximum speed of 235mm/s.










Robot Configuration										
		Single wheel	Hamster car	Pendulum on rotating axis	Gimbal mechanism	Single ball	Mass movement	Orthogonally mounted flywheels	Flywheel on pendulum	Parallely mounted flywheels
Driving mechanism	Driving	D	D	D		D	G	A	D/A	A
	Steering	D	D	D		D	G	A	A	G
Input needed		2	$\geq 2$	2		2	4	2	3	2
Sharp turn		V	V	V*		V	V	V*	V	
Omnidirectional locomotion			V*	V*		V	V	V*		

Figure 2.1: Characteristics of the developed spherical robots. D, G, and A indicate direct-driving, gravity, and angular momentum methods, respectively. The mark v indicates the robot is capable of performing the specific motion, and the mark v\* indicates the motion is achievable in most situations (i.e., with singularity). [8]

While spherical robots date back all the way to 1996 with the first by [60] there are of course more recent papers researching the subject of spherical robots. One example of these is [61] which uses four separate pendulums that rotate around on a separate axis, thereby moving its center of mass in a way that will make the ball roll. One of the advantages of using a spherical shape is that it can be used either on land or on water with little drag. This fact is shown by [62] which made a amphibious spherical robot. This robot has a conduit built in to the sphere making it not perfectly round, but providing a housing for the propeller that is used under water. Its steering under water is done by using a flywheel with substantial mass. On land or on the bottom underwater it uses two pendulums however, having such a conduit means this robot is not capable of driving omnidirectional. In terms of speed it was capable of reaching a maximum of around 0.5m/s. [63] created a more comparable robot, albeit smaller, which also uses an internal omniwheel platform. In this paper they performed experiments by letting their robot move in a straight line and with a small turn. Whilst moving forward 37 cm on the y axis, just over two times the diameter of the robot, they achieved the following:  $x_e -1.392 \pm 28.9mm$  and  $y_e 369.2 \pm 37.7mm$ . Combined with a turn the accuracy went up from around 10% deviation to over 20% when moving 15cm in the x direction and 50cm in the y direction.

## Chapter 3

# Background

To give a more complete picture of how the robot works, this chapter will provide more in depth information. First of all the actual driving mechanism, which is vital to the defined navigation architecture, is explained in both its possible configurations. Then the head and head mount are explained to give an understanding of how the extra levels of the navigation architecture that have been defined for future work can be incorporated. All of this information is aimed to lead to a better understanding of the used and created software and all of the definitions. To see all of the details of the hardware and the specific parts that are used, see Appendix A.

### 3.1 Driving and shell

The body of BB-8 is a large hollow sphere. Within this sphere there is an internal drive mechanism which moves the outer shell around it. The outer shell itself is comprised of two hemispheres with a total inner diameter of 600mm. Both halves are joined together with the aid of hook-and-loop fasteners to allow quick and easy opening and closing of the sphere. To prevent further wheel slippage the inside was coated with a rubber spray that increases the friction on the wheel, thereby reducing the slippage as much as possible. A complete breakdown of all the components and how the shell is built up can be read in Appendix A.1.

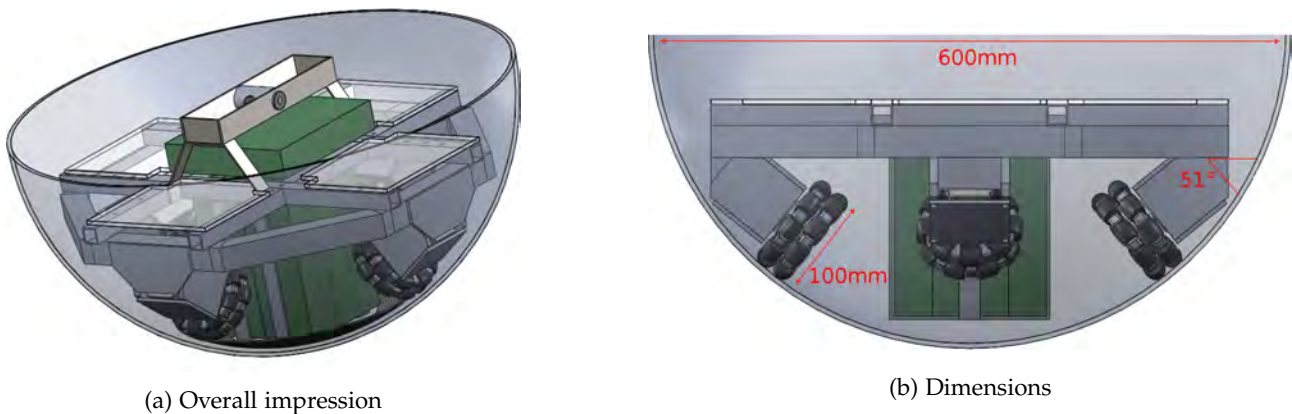
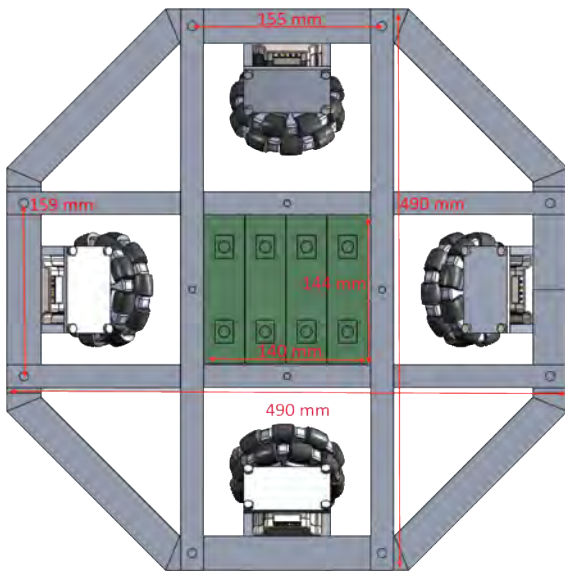


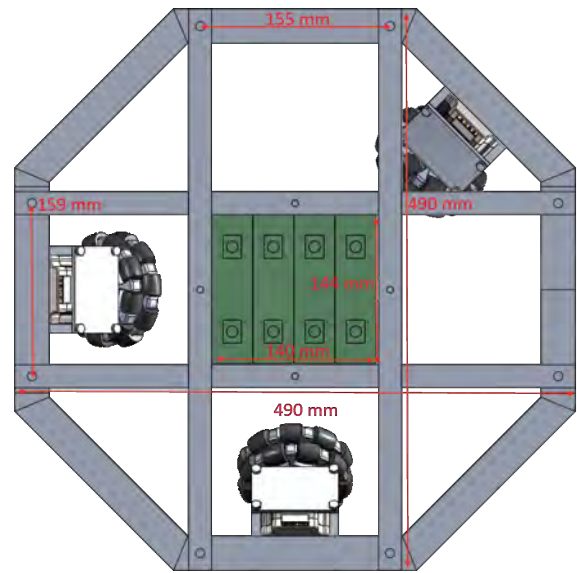
Figure 3.1: BB-8 drive mechanism

The mechanism that moves the outer shell is a platform on which three or four omni wheels are mounted. This type of mechanism is called a hamster ball mechanism [8]. Whilst driving around, the mechanism moves the robot's centre of mass and thereby making the robot itself move, which is also called the barycentre offset principle [64]. Each omni wheel is 100mm in diameter and consists of two parts with the smaller wheels on the outside, each positioned in such a way that there is always an outside wheel on the ground. The omni wheels are fitted straight onto the axis of each of the motors. The entire unit is then fitted into the frame under a 51 degree angle, see Figure 3.1. This is done to ensure that the wheels are always perpendicular to the sphere's inner surface. More in depth detail of the motors and their characteristics is given in Appendix A.2.

In Figure 3.1b we see the dimensions of the robot, which will be useful later on when the wheel odometry is calculated. In Figure 3.1a an overall impression is given of how it would look inside of the ball. The green part in the middle of the overall impression are the batteries with on top of them some more hardware, all together modeled as a single green box. Below in Figure 3.2 more dimensions for the frame are given and the setup with three or four omni wheels is shown. The three wheeled setup as seen in Figure 3.2b is not optimal, i.e. the wheels are not separated according to the perfect distribution of  $120^\circ$  in between. The advantage of this setup however is that it can be quickly converted to a four wheeled setup as seen in Figure 3.2a, since it was not feasible to build a completely new frame. The frame is the basis on which everything rests, holding the motors in their respective motor mounts and the batteries in the centre. Everything else is fitted to the plateau which lies on top of what is visible here, thus on top of the frame. An electrical diagram is given later in Appendix A.3 to show how everything is connected and powered.



(a) 4 wheel setup



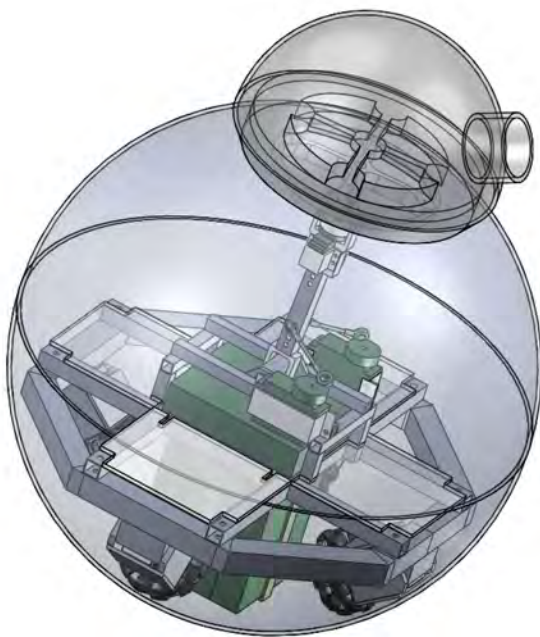
(b) 3 wheel setup

Figure 3.2: BB-8 frame

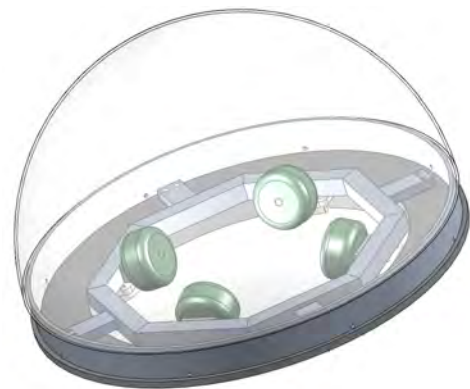


## 3.2 Head and head mount

To have a head mounted on top of the body a head mount was developed which would reside inside of the body and then reach up to the upside of the shell where an array of magnets was mounted. This solution was needed to allow the outer shell to move freely whilst the head would stay attached. The magnets used in the head mount would then draw on another array of magnets inside the head to make sure they would stay at the same position. Since this would require quite some force very powerful neodymium magnets were used with a pulling force of 10kg at a size of just slightly over 2.5cm. This arm that would rise up would have a joint right in the center of the sphere to make sure the array of magnets would, no matter in what position it was, be just under the shell but not touching it. The movement and rotation of this arm come from within the sphere driven by servo motors which are controlled by an arduino. Two servos move the arm, with those two being more powerful to make sure gravity would not pull it down when the head is positioned at an angle. A less powerful servo is then used to control the head's rotation. In the future work when RGB-D camera's are used they are to be fitted in the head, since it is the only place sensors can be mounted that need a straight line of sight.



(a) Headmount overview



(b) Head overview

Figure 3.3: BB-8 head and head mount overview

In Figure 3.3 the entire assembly can be seen. In this figure the head mount is in the center with the two powerful servo's next to it in green. The smaller servo is located inside of the arm near the head also in green, this was the one that only controls the rotation, giving the head about  $180^\circ$  of range. At the end of this arm the array of magnets is mounted which is then also located in the head just below the cross that is visible in Figure 3.3a and below the grey ring surrounding the green wheels in Figure 3.3b, both arrays attracting each other. The head thus rests on the four wheels as seen in Figure 3.3b which are smaller omniwheels, 48mm

in diameter, again tilted slightly to be perpendicular to the outside of the shell. These wheels are held in place by a smaller octagonal frame on which a platform is mounted just as with the frame in the body. This head mount has been developed to enable the usage of other sensors in future work, but is not used in the evaluation of the navigation architecture defined and implemented so far.

## Chapter 4

# Localization

In order for a robots to eventually be autonomous they need to localize themselves in a map by using the available sensor measurements. This of course is at the basis of the dead reckoning capabilities researched here with the proposed navigation architecture, since the robot has to be able to know the amount of distance that is covered. For this reason this is one of the most important chapters to understand correctly. The proposed navigation architecture and its dead reckoning capabilities only requires these techniques to be used for local localization, i.e. only looking at the robot itself. In future work all of these techniques can also be used for global localization, i.e. they look at a known map by comparing the sensor measurements to this map.

With every movement that is made, the estimation of the robots position changes and eventually drifts over time. The drift is caused by the fact that during movement the stochastic models used, lose data by the uncertainty in this movement. This can be compensated by actively computing the robots location with the use of sensor data, thereby adding data to the stochastic model and thus getting a higher probability of the robots exact location [65]. Even though a higher probability of the location can be obtained in this way, some uncertainty always remains. This is caused by a combination of the following four aspects:

- Sensor measurements are noisy
- Problematic sensor measurements might arise due to abnormalities in the real world
- Estimation is always done indirectly
- Sensor measurements might not be available all of the time.

To make the localization of the robot easier, the precision of sensor readings should be increased. Firstly the precision of the movements should be increased in order to know as precise as possible how much distance has been covered. This is done by first calibrating and then calculating the robots odometry, meaning to use data from various motion sensors to estimate the absolute change in position A to position B in a given time frame. By combining more sensors the expectation would be that this would lead to a better understanding of the exact odometry. This means that once a more precise odometry can be calculated, the probability that the absolute movement in the real world is the same as the calculated movement from the odometry will be

higher, thus leading to a lower information loss.

The most used algorithms that tackle the localization problem are implementations of Bayesian or Gaussian models [66]. Based on these uncertainty characteristics of this problem, various probabilistic methods have been designed, the most commonly used being [9]:

- Markov Localization (Bayesian). Uses probability distribution over the state space of all possible hypotheses to localize robot.
- Particle Filters (Bayesian). Uses a distribution of samples over the state space of all possible hypotheses to localize robot.
- Kalman Filter (Gaussian). A recursive filter that estimates the state of a *linear* robot from a series of noisy measurements.
- Extended Kalman Filter (Gaussian). A recursive filter that estimates the state of a *nonlinear* robot from a series of noisy measurements

At the end of this chapter these various localization techniques will be compared to one another. After that a recommendation for the specific technique useful for the proposed navigation architecture is given.

## 4.1 Markov Localization

Markov localization is a straightforward application of Bayes rule to solve the localization problem. It estimates the robots location based on the sensor data by comparing these to the known map. It does this by calculating the probability that the world observed from a position  $x$  at time  $t$  corresponds to the obtained sensor data. By doing so, this method is able to give an accurate estimation of the robots position in dynamic environments. The main idea here is to make sure the robot can attain a high probability density in one position compared to all of the possibly positions of the robot. Thus Markov localization maintains a probabilistic distribution over the entire map for the robots current position.

Before getting to an example, a more formal mathematical definition of Markov localization is given. For simplicity an one dimensional world is used in the equations and the example, thereby only having to apply one axis in which the robots state ( $x_t$ ) can reside. A robot can never know its exact position due to noisy sensor measurements. It does, however, have a belief of where it currently is, which will be graphically depicted in the example by the belief distribution ( $Bel(x_t)$ ).

The following two equations are the equations for belief prediction and distribution [9].

$$\overline{Bel}(x_t) = \int p(x_t|u_t, x_{t-1}, m) Bel(x_{t-1}) dx \quad (4.1)$$

$$Bel(x_t) = \eta p(z_t|x_t, m) \overline{Bel}(x_t) \quad (4.2)$$

In these equations  $x_t$  is the robots state,  $z_t$  the sensor measurements to compare to the map  $m$ .  $u_t$  is the believed odometry. The first of these equations represents the prediction and the second represents the correction step. Both of these steps can be found in the pseudocode representation of the Markov localization

Algorithm 1.

---

**Algorithm 1** Markov Localization pseudo code

---

```
1: for  $i = 0$  to  $n$  do
2:    $P(x_i) = 1/n$ 
3: end for
4: while true do
5:    $u_t = \text{getOdometryMeasurements}$ 
6:    $z_t = \text{getSensorMeasurements}$ 
7:   for  $i = 0$  to  $n$  do
8:      $P(x'_i) = \text{motion\_update}(P(x_i), u_t)$ 
9:   end for
10:  for  $i = 0$  to  $n$  do
11:     $P(x_i) = \text{sensor\_update}(P(x'_i), z_t)$ 
12:  end for
13: end while
```

---

In Figure 4.1 an example is given of how this localization technique would work. In this example the belief is modeled as a continued Bayesian model. In a real world application this would be portrayed with the aid of a grid consisting of several cells which have a value equal the average of the Bayesian model for that part of the world. In the example shown here this would simply be a one dimensional grid, which could be extended to two dimensions for a robot driving around on a single floor or three dimensions when applied for a drone for example.

In the first Figure 4.1a the robot is in its initial position. This is a clear example of the kidnapped robot problem. The robot knows the map (as seen behind it in the form of a wall with three doors), but it does not know its current state. This is shown as the belief distribution being uniformly distributed across all possible states (black bar at on the x axis). Thus the robot assumes it can be at any of the possible states by giving them all the same probability.

In Figure 4.1b the robot is in front of a door. This should, when sensed by the robot, give some more information about where the robots position might be, due to the fact that there are only three doors in the entire map. With the use of Markov localization, the robot is now able to update its belief distribution by increasing the probability of the states that are in close proximity to a door. This can be seen as the  $p(z|x)$  vs  $x$  graph in red in Figure 4.1b. After updating the belief distribution at the bottom of Figure 4.1b in the black graph there are now three assumptions of the position. One thing to notice however is that although a sensor reading of a door might make one assume that all other possible states, not in proximity to a door, should have a probability of zero, this is not the case. Since all sensor measurements will be noisy to a certain degree the change that it falsely reads a door is still present. To compensate for this the probabilities of the other positions remain small, but are not equal to zero.

In the next case the robot moves forward a few meters, which is provided as input to the localization algorithm. In the case of Markov localization the robot then incorporates this information into the belief distribution by moving it forward accordingly as seen in Figure 4.1c. Since even the odometry information can be noisy, and a robot always over- or undershoots it desired location, the belief distribution will be more evenly

distributed by smoothing out the most certain points.

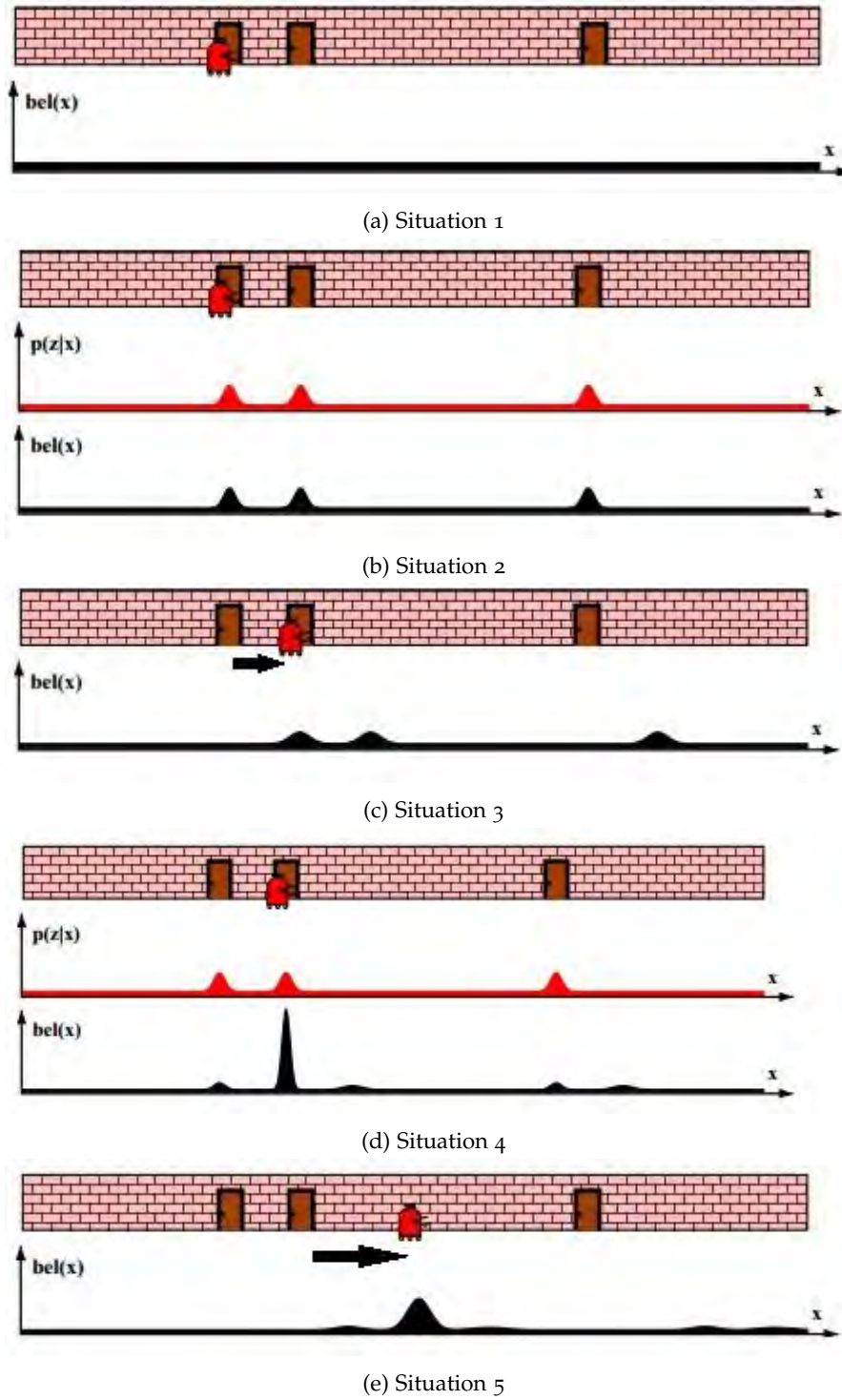


Figure 4.1: Markov localization example [9].

After having moved again the next state is next to a door as seen in Figure 4.1d. If the environment is sensed and close proximity to a door is found which leads to the same graph for probability  $p(z|x)$  vs  $x$  in red in Figure 4.1d as seen in Figure 4.1b. By updating the belief distribution with the new information a single point arises which has the highest probability, with the rest only having small peaks. This is for humans easy to

see since sensing a door, moving a few meters to the right and sensing a door once again can only be if two doors are in close proximity to each other and the robot moved between them. By using Markov localization the robot is able to come to the same conclusion as seen in the belief distribution in the black graph at the bottom of Figure 4.1d.

Now a high confidence over the robots position is found in Figure 4.1d, it moves further than the first time, which leads to some information loss. Because of that, the probability distribution as seen in Figure 4.1e starts to smooth the peaks. If moved indefinitely without sensing new information this distribution would return to a uniform distribution.

## 4.2 Particle filter

The particle filter (PF) [67], also known as Monte Carlo localization, works in an entirely different way then the Markov localization, whilst still solving the same problem. The basic idea here is that it find the most likely location based on a set of sample states, also known as particles. More particles closely together in a single location means that the probability of the robot being in that location is higher. Furthermore each particle can have a weight added to it to distinguish more likely locations. This weight is simply the chance that a measurement  $z_t$  at a location  $x_t$  is correct:  $p(z_t|x_t)$ . This leads to the notation of a particle; it has a location  $x_t$  and a weight  $w_t$  at time  $t$ :  $\langle x_t, w_t \rangle$

There are a few formulas and assumptions on the basis of which this localization method works.

$$\sum_{i=1}^n w_t^{(i)} = 1$$

$$Bel(x) = \left\{ \left\langle x_t^{(i)}, w_t^{(i)} \right\rangle \mid i \in [1...n] \right\} \quad (4.3)$$

Normally a large amount of particles is used to get a better answer. The fact that the amount of particles can differ also helps the PF to be able to better adapt to different situations compared to Markov Localization. In a robot which has less computing power less particles could be used to be able to more quickly, although less precisely, calculate the position.

In order to get a grasp of the theory behind this filter an example is given in which each situation shown corresponds to a situation in the example for Markov Localization. The difference with Markov localization is clearly shown as numerous tiny lines, representing the particles, are visible in the graph in Figure 4.2a compared to a probabilistic distribution. As is the case with Markov localization where the belief distribution was uniform, so do the particles now have a uniform weight (which is denoted by the height of the lines).

Once the robot senses its environment in Figure 4.2b, it calculates the chances of a measurement  $z_t$  at location  $x_t$ :  $p(z_t|x_t)$  (shown in the top graph in red), this step is also called the re-sampling step. Each particle is assigned a new weight corresponding to the probabilities as seen in the bottom graph in Figure 4.2b. However in this state the robot still has no extra clue over where its position is. To recall, the robot estimates its position based on the amount of particles that are in a certain location, not on their weight.

In Figure 4.2c the particles have been redrawn at new locations, which is called the prediction step. In this step new particles are again distributed over the entire graph, however the particles have a higher chance to get in a location where previously a particle with a high weight existed.

If the robot moves as seen in Figure 4.2d, the localization algorithm loses some information due to the uncertain movement. Just as seen in Figure 4.1c where the belief distributions smoothed out a little the particle filter shows the same behavior by spreading out the particles a bit more.

In Figure 4.2e the particles are once again updated with new weight according to the sensed environment. In order to arrive at the same conclusion as the Markov Localization algorithm as seen in Figure 4.1e the PF however needs another prediction step, which is however omitted here since this has already been explained. The pseudo code of the PF can be found in Algorithm 2.

---

**Algorithm 2** Particle Filter Localization pseudo code

---

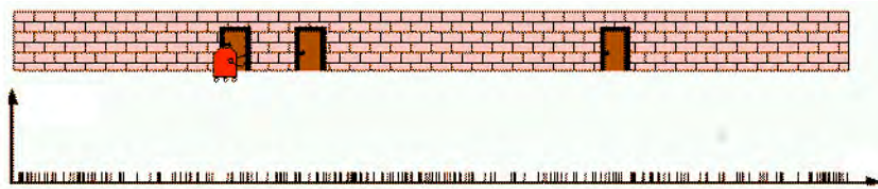
```

1:  $X_{t-1} = \emptyset$ 
2: for  $i = 0$  to  $n$  do draw  $x_{t-1}^{(i)}$  with probability  $\propto \frac{1}{n}$  // Initialize 'previous' state add  $x_{t-1}^{(i)}$  to  $X_{t-1}$ 
3: end for
4: while true do
5:    $X_t = \bar{X}_t = \emptyset$ 
6:    $u_t = \text{getOdometryMeasurements}$ 
7:    $z_t = \text{getSensorMeasurements}$ 
8:   for  $i = 0$  to  $n$  do
9:      $x_t^{(i)} = \text{motion\_update}(u_t, x_{t-1}^{(i)})$ 
10:     $w_t^{(i)} = \text{sensor\_update}(z_t, x_t^{(i)})$ 
11:    add  $\langle x_t^{(i)}, w_t^{(i)} \rangle$  to  $\bar{X}_t$ 
12:   end for
13:    $\text{normalize}(w_t)$ 
14:   for  $i = 0$  to  $n$  do
15:     draw  $x_t^{(i)}$  from  $\bar{X}_t$  with probability  $\propto w_t^{(i)}$  // Resample particles
16:     add  $x_t^{(i)}$  to  $X_t$ 
17:   end for
18: end while

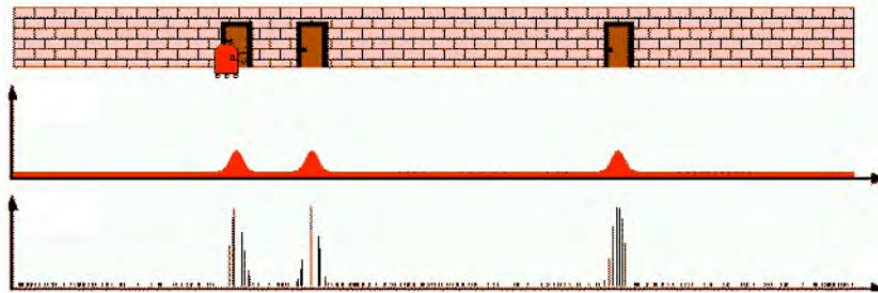
```

---

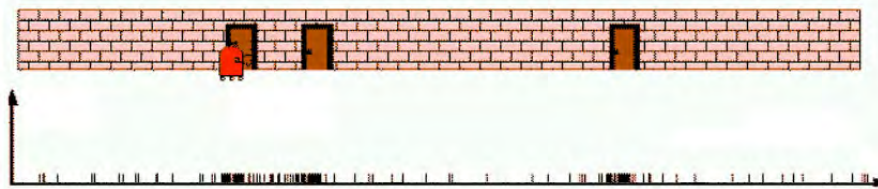




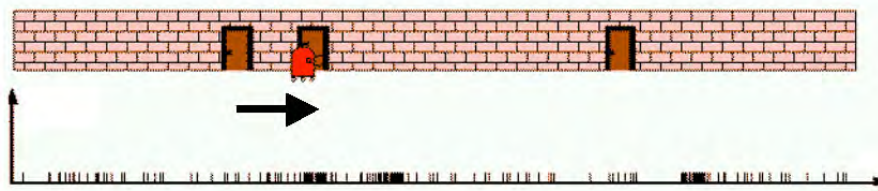
(a) Situation 1



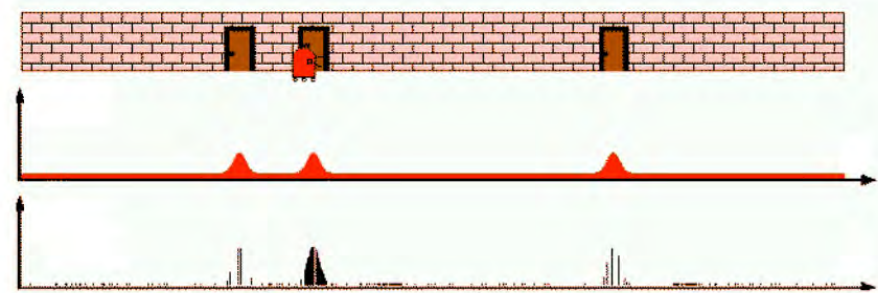
(b) Situation 2



(c) Situation 3



(d) Situation 4



(e) Situation 5

Figure 4.2: Particle localization example [9].

## 4.3 Kalman filter

The earliest version of this filter is simply called the Kalman filter [68] and in more elaborate and newer forms one of the most used in robotics nowadays. The algorithm itself is reviewed mathematically in a more elaborate way after a short visual representation of the workings of this filter.

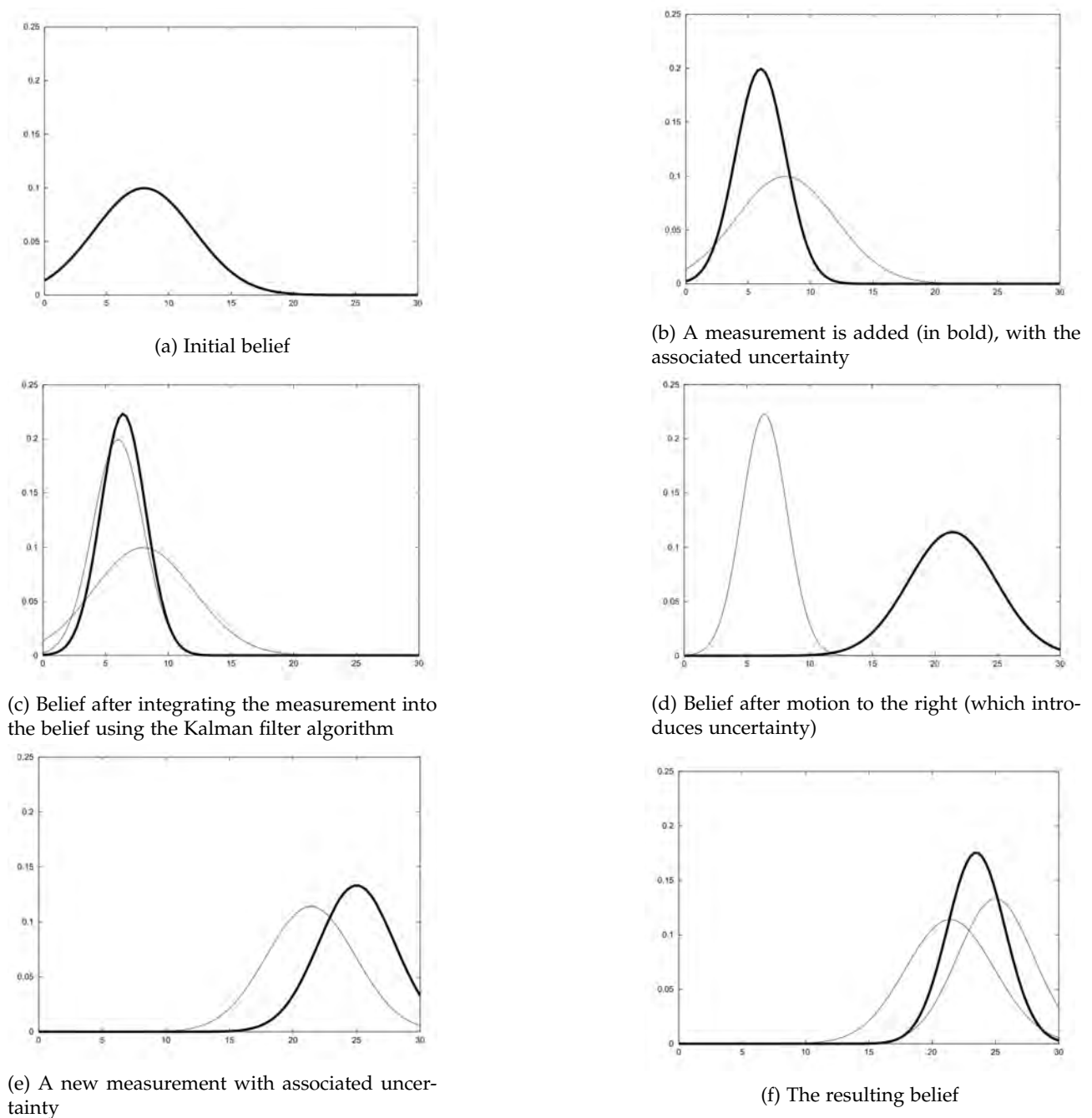


Figure 4.3: Illustration of Kalman filters. [9]

The representation is given in Figure 4.3, a more complete break down of each step is not given since this is in essence the same as with the Markov and particle localization. In the figure its shown clearly that this filter is a Gaussian distribution instead of the Bayesian models which are used in the earlier localization techniques.

This is shown by having only one peak where the robot believes its current location is instead of the Bayesian models which can have multiple peaks.

---

**Algorithm 3** Kalman filter pseudo code

---

```

1: while true do
2:    $u_t = \text{getOdometryMeasurements}$ 
3:    $z_t = \text{getSensorMeasurements}$ 
4:    $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$ 
5:    $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$ 
6:    $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$ 
7:    $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$ 
8:    $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$ 
9: end while

```

---

The filter's algorithm is shown in table 3. It is a different representation of the belief distribution  $Bel(x_t)$  and predicted belief distribution  $\overline{Bel}(x_t)$ . The  $Bel(x_t)$  at time  $t$  is represented by the mean  $\mu_t$  and covariance  $\Sigma_t$ , both of which are vectors with length corresponding to the number of dimensions. Likewise, as with  $Bel(x_t)$ ,  $\overline{Bel}(x_t)$  is represented by  $\bar{\mu}_{t-1}$  and  $\bar{\Sigma}_{t-1}$ . This algorithm's input is  $Bel(x_{t-1})$ ,  $\mu_{t-1}$  and  $\Sigma_{t-1}$ , alongside with control  $u_t$  and measurement  $z_t$ .

In lines 4 and 5 of the algorithm the  $\overline{Bel}(x_t)$  is calculated in the prediction steps. In these two lines the variable  $A_t$  represents the state transition model, e.g. physical laws of motion which predict its movement, and  $B_t$  represents the control input model. Since these two variables are considered to be linear in the case of the Kalman filter, this entire algorithm can only be applied to linear models.  $R_t$  represents the covariance of the Gaussian random vector  $\epsilon_t$ . This  $\epsilon_t$  is a random variable that models the randomness in the state transition that is represented by  $p(x_t|u_t, x_{t-1})$ . This state transition is also used in the Markov localization example, however then it also uses the given variable  $m$  denoting the current map.

In line 6 the so-called Kalman gain,  $K_t$ , is computed. This specifies to which degree the measurement is incorporated into the new estimate, thereby reducing the noise of measurements. The Kalman gain is the really important part of this filter. To understand why it is so useful, consider the following two points:

- Once the measurement noise is large, the error between the expected value and measurements might just be due to the noise. Meaning it is probably not an actual new step and should thus not be taken too seriously.
- If the process noise is large, i.e. the states of the robot are expected to change suddenly, the new value should be taken more seriously. Because due to the fact that this states change quickly the large error in expected and actual measurements is not that weird.

In the same line 6 the variable  $C_t$  and its transpose  $C_t^T$  are found. These represent the observation matrix, which can be used for example to convert polar coordinates to Cartesian coordinates to continue with if the rest is in Cartesian. Lines 7 and 8 then calculate the new  $Bel(x_t)$ . The complete mathematical derivation for this algorithm can be found in chapter 3, subsection 3.2.4, of [9].

## 4.4 Extended Kalman filter

One downside of the 'normal' Kalman filter is that it can only be used for linear system because it uses the assumption of linear state transitions and linear measurements with added Gaussian noise. However nearly every robot that is made, at least in the current day and age, cannot be described in a linear fashion. This is simply due to fact that once it is capable of making turns it already becomes nonlinear. The visual representation is the same as with the Kalman filter and is thus omitted.

For this problem various extensions to the Kalman filter have been defined, one of which is the extended Kalman filter (EKF). This filter handles the nonlinearity by assuming that the next state probability,  $x_t$ , and measurement probabilities,  $z_t$ , are dictated by two nonlinear functions:  $g$  for  $x_t$  and  $h$  for  $z_t$ . The entire EKF pseudocode can be found in Algorithm 4 with the key differences being show in Table 4.1. One other obvious difference is the use of the Jacobians  $G_t$  and  $H_t$  in the case of EKF. The Jacobian  $G_t$  corresponds to the matrices  $A_t$  and  $B_t$  from the Kalman filters. Lastly, the Jacobian  $H_t$  corresponds to the matrix  $C_t$ , both representing the observation matrix.

	Kalman Filter	EKF
State prediction (line 4)	$A_t\mu_{t-1} + B_tu_t$	$g(u_t, \mu_{t-1})$
Measurement prediction (line 7)	$C_t\bar{\mu}_t$	$h(\bar{\mu}_t)$

Table 4.1: KF vs EKF Differences

---

### Algorithm 4 Extended Kalman filter pseudo code

---

```

1: while true do
2:    $u_t = \text{getOdometryMeasurements}$ 
3:    $z_t = \text{getSensorMeasurements}$ 
4:    $\bar{\mu}_t = g(u_t, \mu_{t-1})$ 
5:    $\bar{\Sigma}_t = G_t\Sigma_{t-1}G_t^T + R_t$ 
6:    $K_t = \bar{\Sigma}_tH_t^T(H_t\bar{\Sigma}_tH_t^T + Q_t)^{-1}$ 
7:    $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$ 
8:    $\Sigma_t = (I - K_tH_t)\bar{\Sigma}_t$ 
9: end while

```

---

## 4.5 Comparison and recommendation

Of all of these specific methods one has to be implemented for the robot. Each of these methods has it advantages and disadvantages, both in memory usage and needed computational power. In Table 4.2 a comparison is given where a few of the advantaged and disadvantages are shown for each method. The KF and EKF have been combined into one column since they are nearly the same. The only difference between the two is shown as the last point in the disadvantages cell of the table.

One of the biggest downsides to Markov localization is its memory usage. This comes into play when the world is to be modeled very precisely due to the very large amount of cells in the grid that are then used. This would in turn also lead to a higher computational load for the robot. The PF can overcome this problem

Technique	Markov	Particle Filter	(Extended) Kalman Filter
Advantages	<ul style="list-style-type: none"> <li>• Good in global localization starting from scratch</li> <li>• Recovers well from ambiguous situations</li> <li>• Tunable precision with changing grid resolution</li> </ul>	<ul style="list-style-type: none"> <li>• Memory and computational usage can be adjusted for application</li> <li>• High parellization possible</li> </ul>	<ul style="list-style-type: none"> <li>• Tracks the robot with high precision</li> <li>• Very efficient due to its single representation</li> <li>• Can handle noise quite well</li> </ul>
Disadvantages	<ul style="list-style-type: none"> <li>• High memory and computational usage in environments where high detail is needed</li> </ul>	<ul style="list-style-type: none"> <li>• With uninformative sensor readings samples can quickly congregate</li> <li>• Optimization is hard, no real 'benchmark' possible</li> <li>• Same inputs can lead to different outputs, thus hard to predict and debug</li> </ul>	<ul style="list-style-type: none"> <li>• Is bad at global localization from scratch</li> <li>• Cannot cope well with high uncertainty</li> <li>• (normal only) Cannot handle non linear systems</li> </ul>

Table 4.2: Comparison of localization techniques

by simply reducing the amount of particles used for localization. Where this would lead to a lower certainty of the robots position this means it can also be applied to any large map by optimizing the amount of particles. A big downside to the particle filters however is that with uninformative inputs the particles can conjugate. This happens mostly in ambiguous situations where multiple rooms exists that are very alike and sensor readings are valid for the same position in those rooms. One advantage of both the PF and Markov localization is however that they can in fact localize the robot starting from scratch. Global localization from scratch is however something the KF and EKF are not so good at due to how they work. That is because these filters track the robots pose itself in stead of where the robot might be. The advantage of KF and EKF however is that they are superior in tracking the robots pose over the other localization techniques, since they are far better at handling noise.

With the research goal and proposed navigation architecture in mind the most obvious choice for now would be to implement an Extended Kalman Filter to track the robots position. This is only useful for now since the proposed levels of navigation architecture that are implemented can only track the robot's own movements. When higher levels of the navigation architecture would be added a more advanced form of a particle filter as proposed by [69] would be better in combining actual sensor readings to get the robots pose in reference to the world. Even though a particle filter is used the EKF could still be used to track the robot's own pose which can then be combined into the PF by using the EKF's outcome as input for the motion update in the PF. The reason a PF is chosen over Markov localization is that the PF is, especially in large environments, better suited to be used for localization knowing the memory required for Markov localization. Once the robot is finished it would move through a large building where it needs to have a detailed map to be able to navigate through doors. With Markov Localization this level of detail required would probably result in a way to large map for the robot to efficiently use.

## Chapter 5

# Mapping

This chapter will be elaborating on various 2D and 3D mapping techniques. The mapping techniques discussed here are not yet necessary for the proposed navigation architecture. This can be derived from the fact that for its dead reckoning capabilities the robot only looks at itself and not the environment. However it is useful to understand how the localization techniques can portray the robot in a map and what their (dis-)advantages are. In order to correctly understand the proposed future work it is essential to understand the mapping techniques.

In an office environment furniture tends to get moved around and people walk all around the robot. That is why programming a map into its memory will not work, the solution would be to keep track of the previously build maps and identify its current location based on the landmarks in these previous maps.

The problem of mapping has many different examples, which are not all equally hard. This is a result of a number of different factors, of which the most important are, as described in [9]:

- **Size.** When a map is bigger then what a robot can perceive it becomes more difficult to successfully map the entire environment.
- **Noise in perception and actuation.** Since all sensors and actuators have noisy measurements mapping becomes a really difficult problem due to having to compensate for all the noise. Clearly more noise results in a more difficult problem.
- **Perceptual ambiguity.** When certain places are very much alike it becomes really hard to distinguish them from one another. This problem of course becomes even more difficult the more places have the described close resemblance.
- **Cycles.** Mapping various cycles in an environment, like in the most simple case of moving around a table, is very hard. When a robot moves through a hallway back and forth it can compensate the odometry errors when coming back since walls can not move and thus provide an absolute reference. However when making cycles via different paths the odometric error can become very large due to the fact that it cannot compensate it whilst navigating.

Since there are many different environments in which a robot can find itself an universal world representation

does not exist [3]. The map must be adapted to the task that the robot is to perform and the environment it is performed in. Thus in every occasion a certain map type is to be chosen that is the most efficient and compact for that particular type of environment. In that way if every bit of essential information is stored it can be used quickly and efficiently by other parts of the robots system such as the path planners. The most important part however is that whatever type of map is chosen they must all be able to represent the uncertainty that is inherently in all sensors. In this way the maps can be adjusted later on as will be described in a latter Section 6.2 about loop closures.

A clear distinction can be made between two classes of maps; maps for indoor and structured environments (2D representations) and maps for natural environments (3D representations). In the first case the maps are able to take advantage of the fact that most buildings are highly structured since they are compromised of mostly linear structures with straight corners. In this type of environment robots would normally see very little movement of the objects except for naturally dynamic objects like other agents, humans or vehicles. The maps for natural environments however not only need to be able to hold data for an extra dimension, height, but also the dynamic nature of the objects that are perceived. Where in the indoor environments a 2-D map would suffice due to the fact that nearly all building's floor do not differ in height this would not be the case in natural environments. Besides the needed height information the maps for natural environments need to be able to store more data that is relevant to the robot. For example, sensors could pick up a lake as a flat surface which in the first place looks good to drive on, but that would of course not be possible.

## **5.1 Indoor structured environments**

In the case of a robot that is to perform its tasks indoor a few common map types exist. These shall be explained in turn and their respective benefits will be given. In contrast to the maps about natural environments the maps for this type will be given more detail about since the robot shall perform its tasks indoors. The most common maps for this type of environment are implementations of the following types [3]:

- Occupancy grids
- Line maps
- Topological maps
- Landmark-based maps

### **5.1.1 Occupancy grids**

Their first appearance was back in the 1985 by [70] and it is one of the most popular type of maps. The representation is very simple, given a 2-D array each cell of that array holds the probability value that it is occupied by an object. The big advantage of these grids is that they do not rely on any predefined features. They also have the ability to easily represent unobserved areas and have the ability to offer a constant-time access to the grid. The only downside to this type of map is that they have a high memory requirement

and they could have discretization errors. A depiction of how this mapping method would look like is given in Figure 5.1 where a living room is depicted by a grid map. In the grid map of that picture the objects are depicted with a black cell depicting its occupancy and a white cell of no object resides there. Where this example uses a very low resolution which is not the most suitable to navigate in, the resolution can be increased significantly. The limit in terms of resolution will in that case be determined the memory usage by such a map, since only finite amounts of memory are available.

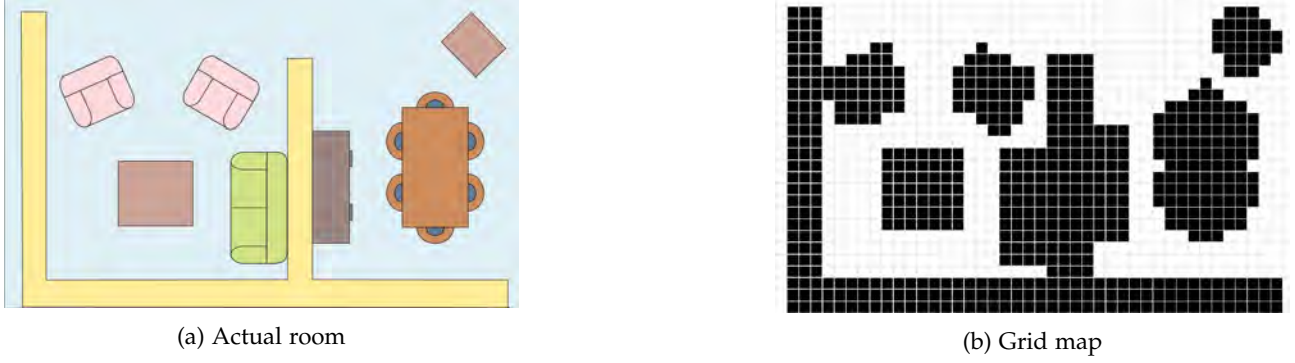


Figure 5.1: Grid map visualization 5.1b of room 5.1a. [10]

To get to theory behind the occupancy grids, the first thing is that it consists of  $n$  cells in a two-dimensional grid,  $m_1, \dots, m_n$ . Occasionally this method is used in with a three-dimensional grid. A single cell in this map represents a perfect square in the real world, each being a certain predetermined size. All of these cells is assumed to be independent of one another. Despite the fact that in the real world object normally span multiple cells, as could be seen in Figure 5.1 where for example the table occupies more than one cell, this method works successfully in robots [3].

By now using the independence assumption of each cell, the formula for calculating the occupancy probability  $p(m_i|x_{1:t}z_{1:t})$  that cell  $m_i$  is occupied given  $p(m_i|x_{1:t-1}z_{1:t-1})$  and new observation  $z_t$  at in state  $x_t$  can be given.

$$p(m_i|x_{1:t}z_{1:t}) = \left( 1 + \frac{(1 - p(m_i|x_t, z_t))}{p(m_i|x_t, z_t)} \cdot \frac{p(m_i)}{1 - p(m_i)} \times \frac{(1 - p(m_i|x_{1:t-1}z_{1:t-1}))}{p(m_i|x_{1:t-1}z_{1:t-1})} \right) \quad (5.1)$$

Equation 5.1 can be made to look a lot easier by introducing the often used *odds* notation for a binary Bayes filter, which is shown in line 1 of the algorithm for occupancy grids, Algorithm 5.

---

**Algorithm 5** posterior occupancy probability of cell  $m_i$ , given observation  $z_t$  in state  $x_t$  [3]

---

- 1: **procedure** ODDS( $p(m_i|x_{1:t}z_{1:t}), p(m_i), p(m_i|x_{1:t-1}z_{1:t-1})$ )
  - 2:      $odds(x) = \frac{p(x)}{1 - p(x)}$
  - 3:      $odds(m_i|x_{1:t}z_{1:t}) = \frac{odds(m_i|x_t z_t)}{odds(m_i)} \times odds(m_i|x_{1:t-1}z_{1:t-1})$
  - 4: **end procedure**
- 

By instantiating the prior of  $p(m_i)$ , also known as  $p_{prior}$ , as 0.5, in the first case when the robot starts building the map, the  $odds(m_i)$  defaults to 1 thereby vanishing from the equation. In that case the map depicts the



unknown occupancy of a cell, since a value of 0.5 says nothing about the actual occupancy or vacancy. Thus a cell holds a value for which the following holds  $0 \leq p_{vacant} < p_{prior} < p_{occupied} \leq 1$

Describing how to compute the occupancy probability  $p(m_i|x_tz_t)$  of a grid cell given a single observation has not been given. This is due to the fact that this differs per sensor type and how they observe the world. This implies that the tuning of the parameters of those models needs to be specifically for to the characteristics of the specific sensors.

### 5.1.2 Line maps

This map type is based on the same assumption as the occupancy grids, namely that indoor structured environments have lots of linear structures such as lines and planes. The advantage of line maps however is that they allow the robot to use substantially less memory to represent the environment. Moreover they allow the environment to be depicted in memory more precisely since the lines can be drawn where ever objects reside instead of having to discretize them into one of the cells of the occupancy grid. A visualization of how these line maps could be depicted is seen in Figure 5.2. In the left we see in Figure 5.2a the laser scan data, this image is a collection of numerous points from the sensor. The line map drawn from all of these points is given in Figure 5.2b. In this figure an algorithm by [11] is used to extract these line segments.



(a) Laser scan data



(b) Line map

Figure 5.2: Line map visualization 5.2b of laser scan data 5.2a. [11]

As seen, line maps are a big collection of  $n$  pairs of Cartesian coordinates  $(x_i, y_i)$ . Where it is easy to draw one line fitting those coordinate pairs with the use of the least squares technique this of course would not be a good depiction of reality. To solve the problem of correctly drawing multiple lines from the set of points two questions need answering. Firstly the map needs to be divided into several lines, thus the question of how many lines should be drawn has to be answered. Secondly if the first question has been answered then it comes down to the question of which points then belong to which lines.

Besides the algorithm by [11] used in Figure 5.2 various algorithms have been made to solve this problem.

The most popular approach to this is the so-called split-and-merge algorithm by [71]. In this method a line is drawn according to the least squares technique. Then the point furthest away is chosen and if this distance is above a certain threshold the drawn line is cut in half at the point that was furthest. After that there are 2 lines, from the original start point to the point that was furthest away and from that point to the original end point. This process is repeated until the distance from every point to its corresponding line is below the set threshold. Where this method is very efficient and quickly, it does not however always give the best results. Another much used, but thus more computationally expensive, method is the Expectation Maximization method [72]. This method can be regarded as a variant of the fuzzy-k means clustering algorithm [73]. For the purpose of this thesis it is not useful to go into more details about the workings of all the other algorithms.

### 5.1.3 Topological maps

Topological maps look at the surrounding world in a different way, namely by representing it as a graph. The pioneering work for these maps has been done by [74]. In their work they represented the map with a graph-like structure, in which each of the nodes is a locally distinguishable place. These nodes together form the travel edges along which the robot can move between. Distinguishable places are identified according to the distance they have to nearby places. These places were defined by [75] as the meet-points in generalized Voronoi graphs. A generalized Voronoi graph is the set of point which is equidistant from the two (or more) closest obstacle boundaries. This approach means they are a very good depiction of the topological structure of the environment and can be considered as road maps. Using these kind of maps as road maps is very simple since the robot can simply plan a path to one of the points on the generalized Voronoi graph. Then simply following the graph as close as possible to the desired destination, at which the robot stops following the graph and moves to its goal.

A depiction of these kind of graphs is shown in Figure 5.3. In this figure the easy path planning that these maps enable is shown clearly by seeing that these graph lines are equidistant to the obstacle boundaries. Thereby enabling the robot to move freely along these lines without danger of collision.

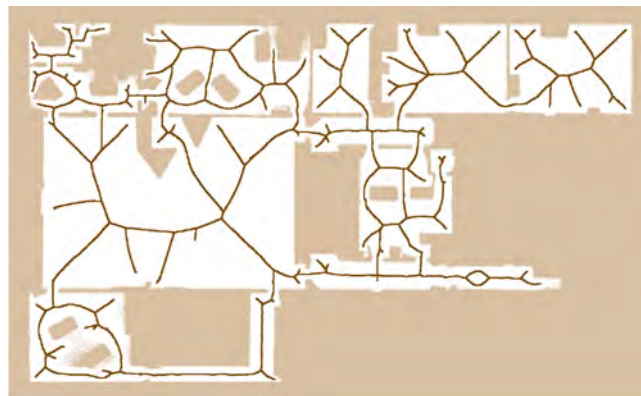


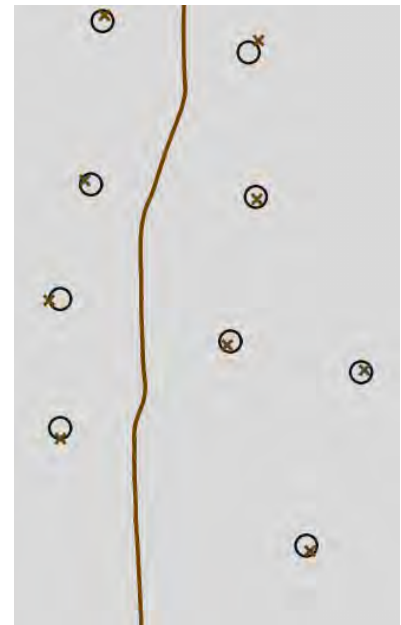
Figure 5.3: Example of a generalized Voronoi graph. [3]

### 5.1.4 Landmark-based maps

When it comes down to environments that have locally distinguishable places, landmark-based maps are used extensively. A locally distinguishable place is actually quite common, especially nowadays when camera's are used on robots, since for a painting or couch somewhere in a room is nearly always distinguishable from other places. A visualization of this kind of map is given in Figure 5.4 where a mobile robot is mapping a set of stones.



(a) Robot mapping the stones



(b) Estimated path and landmark positions (manually determined positions given by circlxy)

Figure 5.4: Visualization of a landmark-based map. [3]

When using the assumption that the robot pose is known, it simply remains to estimate and maintain a position for each landmark over time. Landmark-based map  $m$  can then be represented by  $N$  two-dimensional Gaussians, where  $N$  equals the number of landmarks, in a planar environment. Various approaches to these maps can be used, one of which is using the EKF for landmarks. This approach would result in having  $2N + 3$ -dimensional state vector ( $2N$  two dimensional Gaussians for the landmarks and 3 for the robot pose). Another approach that is FastSLAM [76] which reduced the required dimensions to just  $N$  two dimensional Gaussians and then using localization to determine the robot's pose. This algorithm is an implementation of Simultaneous Localization and Mapping (SLAM), which will be explained more thoroughly in a later section.

## 5.2 Natural environments

In the case of natural environments it is no longer appropriate to project all of the sensed data into a 2-D map. In these environments the robot needs to be able to store elevation changes. Overhanging structures such as three canopies are hard to depict by using just elevation, therefore more advanced true 3-D map types are

also available. The need to depict overhanging structures has become more apparent recently by the arrival of drones, which need to be able to not only collide with the ground, but in some occasions need to be able to go through holes in walls or underneath trees.

The most used types of maps are as follows [3]:

- **Elevation grids.** This type is simply an extension of the occupancy grids for indoor environments because it also holds elevation data. The most simple depiction of this type can be seen as a wire mesh. It is easy to implement but quickly becomes obsolete in environments where vertical surfaces or overhangs reside due to its incompatibility with these types of structures. Extended versions of these maps have been made to work around their limitations in the form of multilevel surface maps where they allow the storage of multiple surfaces in each cell of the grid [77] or extended elevation maps where certain classes are given to the cell to depict vertical objects [78]
- **3-D grids and point clouds/sets.** When structures such as trees have to be depicted in maps, these types are good options. The most easy implementation of these is to use the occupancy grid approach but then expand it into the third dimension. The downside however is that it quickly becomes very memory hungry since the array's used to store the data become humongous when a high resolution is chosen in outside environments. A solution for this is given in [79] where data structures are presented based on dynamic 3-D grids.
- **Meshes.** Both Elevation grids and 3-D grids and point sets thus have their respective limitations on the types of environments they can depict, which also makes them very suitable if they are indeed needed. Meshes however can be adapted to depict nearly any type of terrain. By using advanced algorithms [80] the amount of vertices in these maps can also be reduced to minimize the memory need. The only downside is the difficulty to correctly map the environment, since due to sensor noise the map might depict two separated surfaces as connected.
- **Cost maps.** These maps are an extension of elevation maps, because they also hold data for the cost of moving over that particular part of the map. Various algorithms, such as [81], have been designed to extract the cost of the cell from the slopes and textures of the terrain. More advanced algorithms also take into account the dynamics of the robot to better depict the cost.

### 5.3 Recommendation

Whilst mapping techniques will not be used for the defined navigation architecture so far, they are, however, to be used in conjunction with future work. Once higher levels of the proposed navigation architecture are implemented one of the methods from the Indoor Structured Environments is to be implemented first. The easiest would be to start with simple grid based maps, like the occupancy grids. When using occupancy grid the (later on) proposed radar sensors or collision detection from the IMU can be implemented. From there on the robot can be developed to work correctly before even higher levels are implemented. Once RGB-D camera's or laser range finders will be implemented a method from the Natural Environments can be chosen due to the introduction of 3D data. A good implementation to use in this situation in order to keep the

memory usage lower is landmark based maps (graph maps). Once detailed 3D data is acquired this could be stored in one of the landmarks which could then be used to make assumptions of the robots global position in reference to the map by comparing the current information to the landmarks. The usage of grid based maps to prevent collision with objects also becomes less important since the most recent data from the range finders and RGB-D camera's can be used for that goal. An example of this graph based mapping technique has been proposed by [12].

## Chapter 6

# Simultaneous Localization and Mapping

This chapter will be covering the problem of simultaneously building a map of the current surroundings as well as to localize the robot in this map. This described technique is known as Simultaneous Localization and Mapping (SLAM). Understanding SLAM is needed in order to solve the problem of autonomous driving for the proposed future work. For the proposed and researched navigation architecture this technique is not yet necessary.

The main idea behind SLAM is to create a map with the sensor data (from sensors looking to the environment) as input to localize the robot, then to translate this map after the robot has moved followed by doing those steps again with the current world perception as the start of the next iteration. This entire world is viewed as a stochastic model in which each perceived object and the current location is approximated. This problem can in a way be seen as a chicken-or-egg problem, the map is needed for localization, but the pose estimate is needed to build the map.

SLAM thus is a very important part in any robot in order for it to navigate. Because of this importance various methods have been researched by many, of which the most popular use localization techniques such as the Kalman- ([82] and [83]), information- and particle filters.

### 6.1 Taxonomy of SLAM

As mentioned there are a lot of solutions to the SLAM problem, each of which has its own characteristics and thus differences. These differences can be combined into a few topics which will be discussed here [3]. One of these differences, namely single-robot vs multirobot SLAM, will not be discussed since it has no use for this thesis' robot BB-8.

- **Volumetric - Feature-based.** Volumetric SLAM samples the map at a very high resolution, enough for a photo-realistic reconstruction of the environment. Feature-based SLAM extracts features from the

surroundings, like landmark based maps. The downside to feature-based SLAM is that it thus discards information but the upside is its increased efficiency.

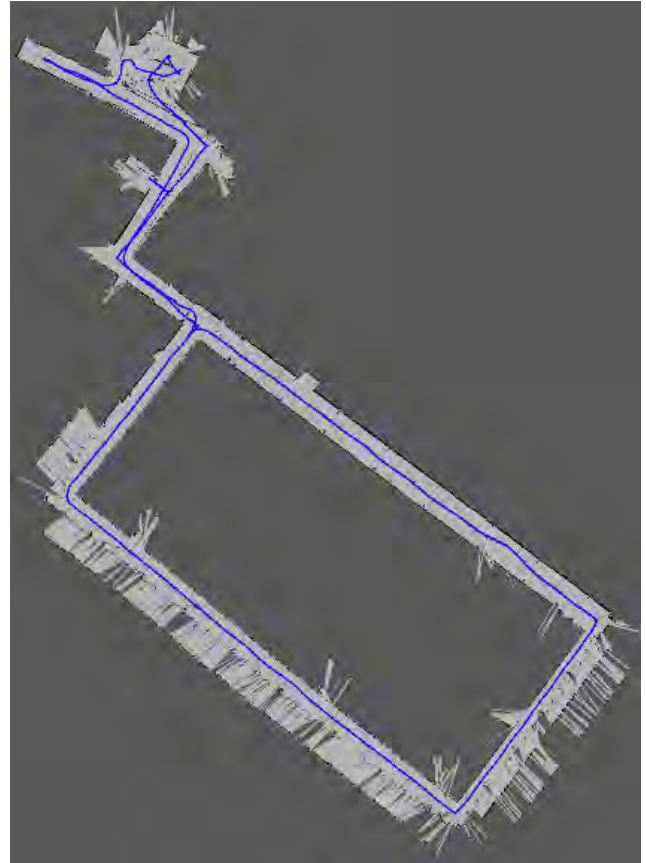
- **Topological - Metric.** Topological mapping techniques use a description of the environment, for example place a is adjacent to place b. Whereas metric mapping describes precise metric information of the relation between objects instead of just a description.
- **Known - Unknown correspondence.** This subproblem refers to relating sensed objects to previously sensed object, for example sensing a painting that is already a part of the landmark based map. More on this topic in the next subsection about loop closures (Section 6.2).
- **Static - Dynamic.** This has everything to do with the environment. Dynamic methods incorporate shifts in the environment, like a chair that has moved, while static methods cannot cope with this.
- **Small - Large uncertainty.** This covers the amount of uncertainty the method can handle when it comes to the robots position. Some incorporate loop closures (Section 6.2) methods which allow more uncertainty in the location after a certain time, only to be reduced significantly at sensing a known position, whereas others could use absolute positioning systems such as GPS to keep the uncertainty small.
- **Active - Passive.** Active SLAM methods the method itself controls the movement of the robot, with the goal of pursuing the most accurate possible map. In passive SLAM methods the SLAM method is simply an observer which senses the environment and places the robot somewhere. Another part of the robots program can then control the movements. Hybrid approaches are possible where the active part points the sensors in a direction and the passive part moves the entire robot.

## 6.2 Loop Closures

In each SLAM solution a key feature is the ability to recognize previously visited locations, this process is called loop closure detection. The name loop closure detection implies that coming back (after making a loop) to a previously visited location will allow the robot to detect this location with such a visited place. As mentioned, most of the slam solutions are based on probabilistic models of the world. For these solutions the loop closure detection methods are all done locally, meaning that loop closures are found between the current (new) observation and a small part of the current map. This small part of the map is determined by the robot never fully knowing where its position is, it only has an estimated position around which each of the other positions also is a possibility. These probabilistic based loop closure detection methods ([84], [85] and [86]) have the advantage that all of the information can be processed near real-time. The downside to these methods is that once the estimated position is not valid, they all cease to work because they can no longer give good estimates of loop closures. The fact that estimated positions cannot be guaranteed to be valid is a serious problem [87]. A visualization of what loop closure exactly does is given in Figure 6.1.



(a) The current map is visible with the detected loop closures in red.



(b) The optimized map after all of the detected loop closures have been applied.

Figure 6.1: Loop closure example. [12]

The problem in the case of BB-8 however is that it will be operating in a large scale environment (an entire office building) and that it will do this for up to 4 hours or more. The complexity of the loop closure problem then becomes a lot harder to solve, subsequently increasing the memory and time required to process new observations due to the increased number of locations in the map. A solution to this is given in [88], this solution is not elaborated on further but could be used in the future when further development is done to extend BB-8's capabilities beyond the scope of this thesis.



## Chapter 7

# Motion Planning

This chapter will go a bit into detail about motion planning and what techniques exist. Since this is less important for the currently proposed navigation architecture most of the mathematics behind each of the reviewed methods is omitted. However, if higher levels of the navigation architecture as defined later on are implemented a motion planning algorithm becomes essential. Firstly a few motion planning algorithms will be reviewed before a recommendation is made about which algorithm will be useful and with what navigation architecture level it should be used.

The problem of motion planning is of course at the basis of autonomous robotics since each robot accomplishes tasks by moving in the real world. To achieve these tasks a motion planning algorithm should try to do the following two things [13]:

- Get to the destination with a collision-free trajectory
- Calculate a trajectory to the destination which is the shortest possible trajectory

In order for the algorithm to do this it needs to know a few things in order to successfully calculate a path [13]:

- Current/start pose of the robot
- Desired pose at the goal
- A description of the robot with its geometric shape
- A map of the world

While all of the actions that a motion planning algorithm provides are executed in the real world, the actual problem of motion planning lives in the Configuration Space (C-space). The C-space can be defined in a varying number of dimensions. The amount of dimensions corresponds to the degrees of freedom the robot has, which is three dimensions,  $(x, y, \theta)$ , when a robot is used in a 2D plane. To obtain the C-space the robot's radius is used to "blow up" the edges of obstacles present in the map, called work space, by sliding this radius along all edges of all obstacles. A visualization of this process is given in Figure 7.1. In this figure a motion planning algorithm creates the configuration space from the work space and calculates a path from a reference point to an endpoint. A noteworthy detail is that the robot in question does not have the same radius along all sides. This can be seen as the grey area around the black obstacles which is not equally large

along all edges. If this example would be extrapolated to a third dimension (for orientation) this would lead to very strange 3d shapes of the C-space due to the odd shape of the robot.

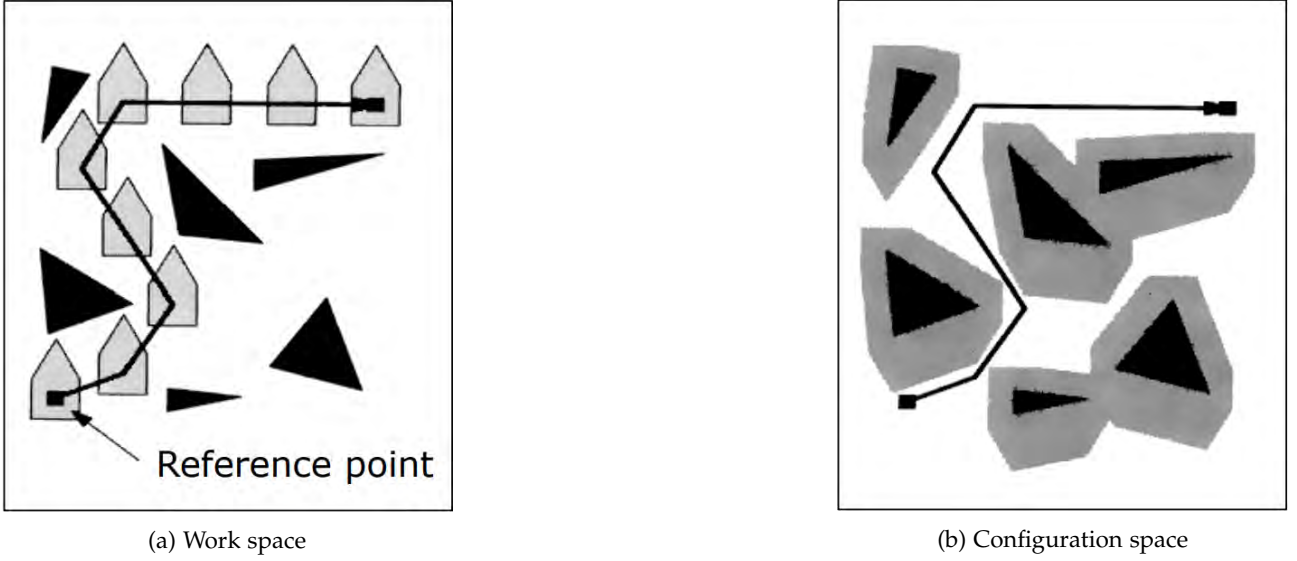


Figure 7.1: Visualization of obtaining configuration space from work space [13]

Before a detailed explanation of motion planning algorithms using this C-space are given, a little more light is shed on the formal notation of the C-space. As the C-space is now constructed by blowing up the work space the remaining areas can be named in a mathematical manner as a set of points. Once formally defined the following definition of the C-space remains for a rigid body that can translate and rotate:

$$C = \mathbb{R}^2 \times S$$

The work space is denoted as either  $W = \mathbb{R}^2$  or higher orders of  $\mathbb{R}$  depending on the degrees of freedom of the robot. In this work space an obstacle region  $O \subset W$  is defined. Besides the obstacle region a rigid body  $A \subset W$  is also defined, commonly denoting the robot itself. Then let  $q \in C$  denote the configuration of  $A$  where  $q = (x_t, y_t, \theta)$ . The obstacle region  $C_{obs}$  is then defined as:

$$C_{obs} = \{q \in C | A(q) \cap O \neq \emptyset\}$$

By this definition  $C_{obs}$  is the set of all configurations of rigid body  $A$ , where this rigid body intersects the obstacle region,  $O$ . Lastly the leftover regions are defined as  $C_{free}$ :

$$C_{free} = C \setminus C_{obs}$$

$C_{free}$  can be seen in Figure 7.1b as the white area in which the robot can move.  $C_{obs}$  holds both the black and grey areas from the same figure where the robot is unable to move. To formally denote the problem of motion planning two more definitions are introduced. First of which is the initial configuration  $q_1 \in C_{free}$ , followed by  $q_G \in C_{free}$  denoting the goal configuration. All of the definitions combined are shown in a visual representation in Figure 7.2, where the work space  $W$  encompasses all elements.

The problem definition for motion planning algorithms is then as follows [14]:

1. Given a world,  $W$

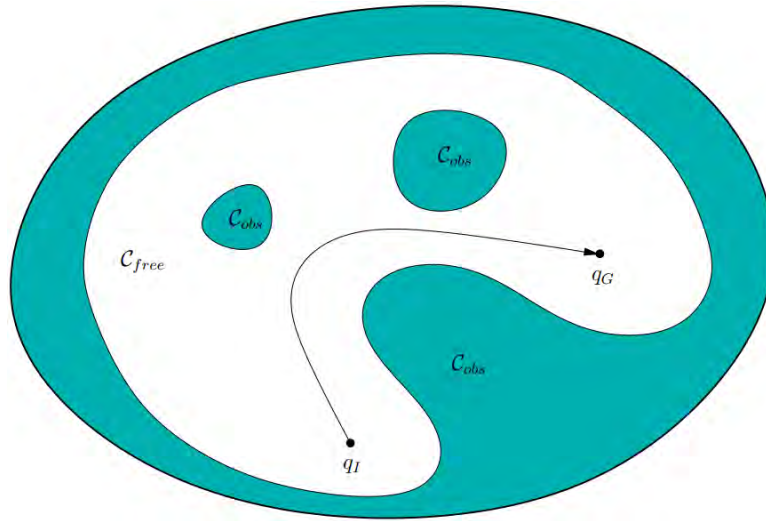


Figure 7.2: Configuration space definitions [14]

2. Given an obstacle region  $O \subset W$
3. Given a rigid body  $A$  which is defined in  $W$
4. Given a configuration space  $C$ , from which  $C_{obs}$  and  $C_{free}$  are derived
5. Given an initial configuration  $q_1$  and a goal configuration  $q_G$ , together denoting the query pair  $(q_1, q_G)$
6. Compute a path  $t : [0, 1] \rightarrow C_{free}$ , where  $q_1 = t(0)$  and  $q_G = t(1)$  or correctly report that there is no such path

For algorithm to solve this problem definition, they need to discretize the C-space. This is needed to reduce the map size making it suitable for processing in a manner that is quick enough for real time motion planning. There are two general approaches available to discretize the C-space, each of which has its own set of different algorithms which will be explained later. The two main approaches are [13]:

- Combinatorial planning
- Sample based planning

## 7.1 Combinatorial planning

In order to solve the problem definition for the motion planning algorithms combinatorial planning methods find paths in the C-space without resorting to approximations. Given this property they are often referred to as exact algorithms, which are also complete. Being a complete algorithm defines them as being able to find a path in any problem instance or correctly reporting no such path exists. Some of the algorithms are even optimal, meaning that the path found is the best possible path to take (e.g. shortest or quickest).

The downside of these kinds of planning algorithms however is that they become intractable quickly when higher order dimensions are used. This is due to the fact that all of these methods report failure by exhaustively searching the C-space until no more options remain. Another downside to these methods in combination with higher order dimensions is that they explode in terms of facets to represent  $A$ ,  $O$  and  $C_{obs}$ . This

is especially apparent once rotations are taken into account, resulting in a C-space resembling a nontrivial manifold [14].

All of the methods that fall under this technique produce a road map [13]. This road map is a graph in  $C_{free}$  in which each vertex is a configuration in  $C_{free}$  and the edges are collision free paths through  $C_{free}$ . A few of the basic algorithms that fall under this type are:

- Visibility graphs
- Exact cell decomposition
- Approximate cell decomposition
- (Generalized) Voronoi graphs

All of the methods above will be discussed except the (generalized) Voronoi graphs, since these have already been defined in Section 7.1.

### 7.1.1 Visibility graphs

Visibility graphs are one of the earliest path planning methods. Their idea is to construct a path as a polygonal line connecting  $q_1$  and  $q_G$  through the vertices of  $C_{obs}$ , thereby only narrowly avoiding collision. The advantage of constructing a path in this manner is that it results in an optimal path [89, Chapter 15]. To compute a path this method sees all obstacles in the world as closed polygons. These polygons are made up of multiple vertices and edges. Some of these vertices are called reflex vertices and play an important role in the computation of the path. A reflex vertex is a polygon vertex for which the interior angle (residing in  $C_{free}$ ) is greater than 180deg. All vertices of the computer roadmap from which a path is calculated is a reflex vertex.

The shortest path roadmap  $G$  is then constructed according to Algorithm 6. A visualization of this algorithm can then be found in Figure 7.3.

---

#### Algorithm 6 Visibility graph path planning pseudo code

---

- 1: Create empty graph  $G$
  - 2: Add  $q_1$  and  $q_G$  as vertices in  $G$
  - 3: Add polygons from  $C_{obs}$  to  $G$
  - 4: Place all reflex vertices from all polygons in list  $V$
  - 5: **for**  $i = 0$  to  $n$  **do**
  - 6:     Connect  $V_i$  to  $q_1$  or  $q_G$ , if the constructed edge does not intersect any polygon add it to  $G$
  - 7:     Connect  $V_i$  to all other vertices in  $G$  and add this edge if it also has no intersections
  - 8: **end for**
  - 9: Run a graph planner (like Dijkstra's or A\*) on  $G$
-

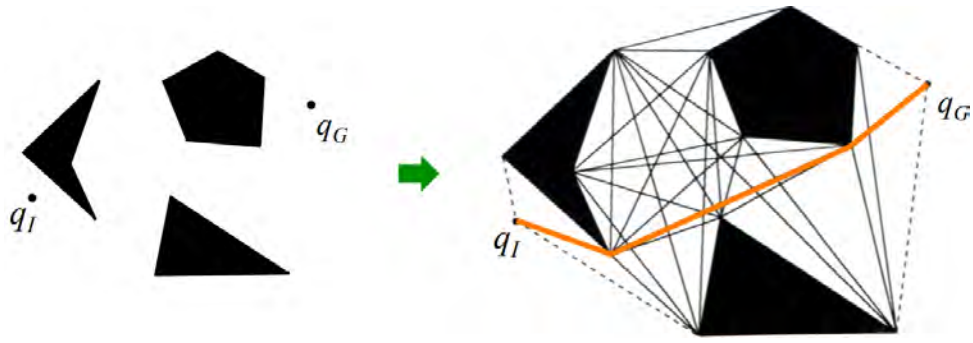


Figure 7.3: Visibility graph algorithm visualization [14]

### 7.1.2 Exact cell decomposition

Whereas visibility graphs look at the edges of  $C_{obs}$  only narrowly avoiding them, exact cell decomposition looks at  $C_{free}$ .  $C_{free}$  is decomposed into non-overlapping cells. This method does not however compute the optimal path. In order to explain this algorithm no more definitions are required besides the ones given in the the introduction of this chapter. Thus below the psuedo code is found in Algorithm 7 with its visualization in Figure 7.4. This example focuses on the trapezoidal decomposition method. Other implementations of this algorithm are possible [13], but for the sake of simplicity omitted here.

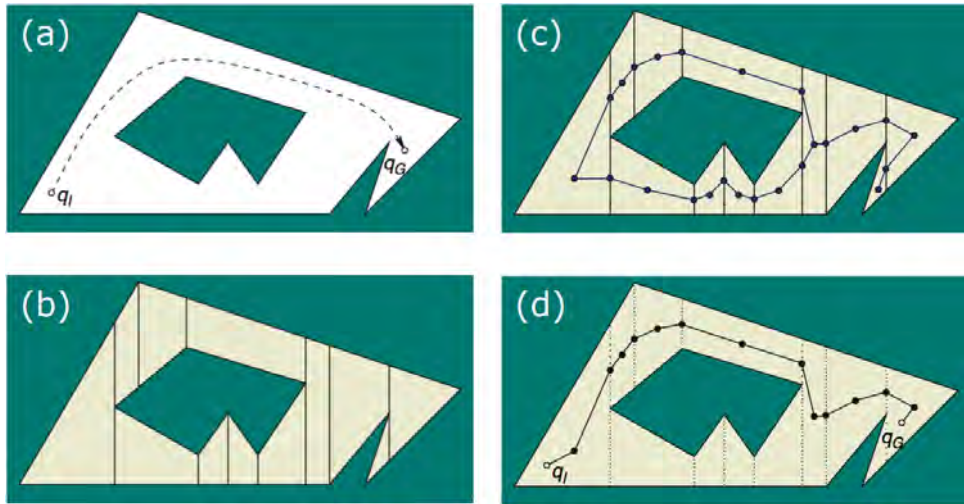


Figure 7.4: Exact cell decomposition algorithm visualization [13]

---

**Algorithm 7** Exact cell decomposition path planning pseudo code

---

```
1: Create empty graph  $\mathbb{G}$ 
2: Add  $q_1$  and  $q_G$  as vertices in  $\mathbb{G}$ 
3: Place all vertices from all polygons in list  $V$ 
4: for  $i = 0$  to  $n$  do
5:   Shoot a vertical line from  $V_i$  both upwards and downwards, add the one or more created trapezoids to  $T$ 
6:   Place one vertex on the drawn line, e.g. the centroid and add it to  $\mathbb{G}$ 
7: end for
8: for  $j = 0$  to  $n$  do
9:   Place one vertex in trapezoid  $T_j$ , e.g. the centroid and add it to  $\mathbb{G}$ 
10: end for
11: Connect all vertices in  $\mathbb{G}$  with edges as long as each edge does not touch or intersect  $C_{obs}$ 
12: Run a graph planner (like Dijkstra's or A*) on  $\mathbb{G}$ 
```

---

### 7.1.3 Approximate cell decomposition

The downside of exact cell decomposition methods is that they use odd sized shapes in order to calculate path. This leads to inefficient and complex problems that are to be solved. The approximate cell decomposition techniques however use the same simple predetermined shapes making them more efficient and easier. The downside of the approximation however is that these methods can be complete, but they have to be made so [14].

A few methods fall under this type, the example that is given in Figure 7.5 is in the form of quad tree representation. This method can sometimes also be referred to as an adaptive cell decomposition method. This is due to the fact that whilst all of the cells have the same shape, they do not, however, have the same size. Other form such as equal sized squares can also be used. Pseudo code for this type of path planning can be found in Algorithm 8.

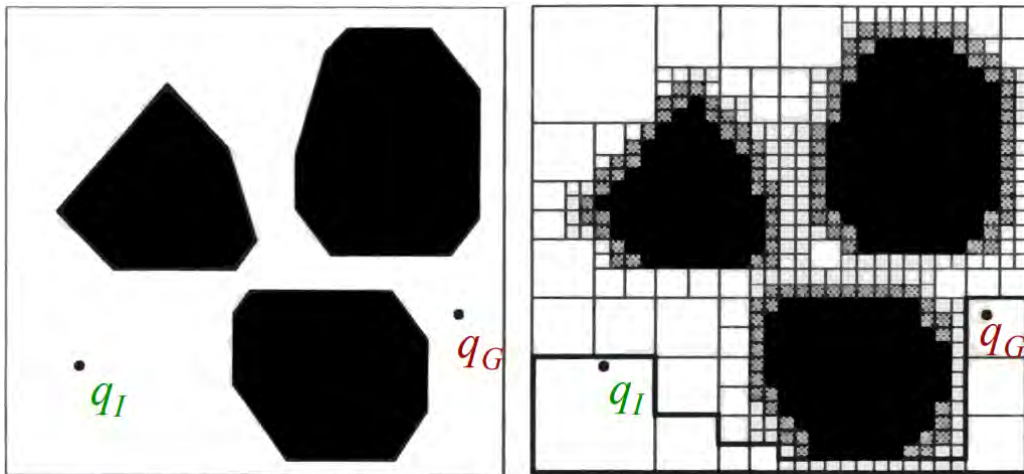


Figure 7.5: Approximate cell decomposition algorithm visualization [13]

---

**Algorithm 8** Approximate cell decomposition path planning pseudo code

---

```
1: Discretize given C-space into smaller fixed shape areas and add to  $D$ 
2: for  $i = 0$  to  $n$  do
3:   Mark  $D_i$  as passable if  $D_i \cap C_{obs} = \emptyset$ 
4:   If passable add vertex in centre of cell and add to  $\mathbb{G}$ 
5: end for
6: Mark the vertex residing in cell where  $q_1$  resides as start
7: Mark the vertex residing in cell where  $q_G$  resides as goal
8: Connect all vertices  $\in \mathbb{G}$  to the vertices from neighbouring cells
9: Run a graph planner (like Dijkstra's or A*) on  $\mathbb{G}$ 
```

---

## 7.2 Sample based planning

The main idea behind sample based planning is to search the C-space with a sampling scheme, explicitly avoiding the construction of  $C_{obs}$ . Probing the C-space in this way is done with the aid of a collision detection module, which acts as a sort of black-box. This module checks if the given configuration are in fact in  $C_{free}$ . In this manner the development of planning algorithms that are independent of the particular geometric models is enabled. Whilst this results in a weaker conclusion of a good path it is however far more efficient by not exhaustively searching the C-space. To compute a path these algorithms all use a random element whilst going through the C-space trying to find a vertex residing in  $C_{free}$ .

The advantage of Sample based planning methods is that they are quickly able to come up with a solution in a higher order dimension. The solution that is presented is usually far from optimal, but given enough time can approach optimality. Most of the robots nowadays use these kind of methods due to their high dimensionality, often having arms or multiple joints which result in more degrees of freedom.

Just as with combinatorial motion planning algorithms a lot of methods have been defined which are part of the sample based planning methods. The two methods that will be shown are:

- Probabilistic Road Maps (PRM)
- Rapidly Exploring Random Trees

### 7.2.1 Probabilistic Road Maps

PRM's are very quick in finding solutions to higher order dimensions but have some downsides. Due to their random nature in picking vertices in the C-space they are not well suited for C-spaces in which narrow corridors reside. It is improbable that a vertex is randomly selected in those corridors knowing the possible configuration from the entire C-space. Whilst this is a downside these methods are however probabilistically complete. Given enough time and resources the random nature of this methods leads to all configurations being exhaustively searched and a solution to be found optimal.

To solve the downsides of the random nature various extensions of PRM's have been defined in more recent literature. For example these extensions can add advanced sampling strategies close to objects, increasing the

chance so solve problems when narrow corridors are in the C-space. All of these extensions are, however, out of scope for this thesis. Once again a figure to illustrate its workings as well as a psuedo code algorithm are given in Figure 7.6 and Algorithm 9.

---

**Algorithm 9** Probabilistic Road Map path planning pseudo code

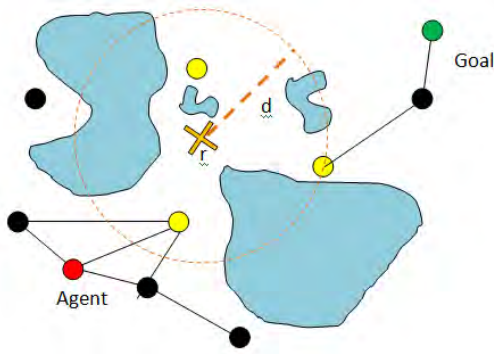
---

```

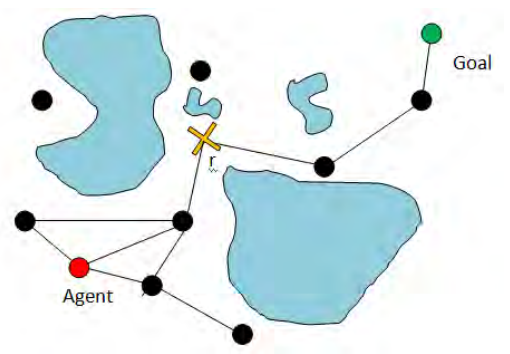
1: Create empty graph  $G$ 
2: Add  $q_1$  and  $q_G$  as vertices in  $G$ 
3: for  $i = 0$  to  $n$  do
4:   Sample a random configuration,  $q_r$  in C-space
5:   if  $q_r \notin C_{obs}$ , checked by collision detection module then
6:     Add  $q_r$  to  $G$ 
7:     for  $n \in \text{neighbourhood}(q_r, G)$  do
8:       Draw edge between  $q_r$  and  $n$  if edge does not cross  $C_{obs}$ 
9:     end for
10:  end if
11: end for
12: Run a graph planner (like Dijkstra's or A*) on  $G$ 

```

---



(a) New sample of point in C-space



(b) Newly added edges in  $G$

Figure 7.6: Probabilistic Road Map algorithm visualization [15]

### 7.2.2 Rapidly Exploring Random Trees

Rapidly exploring Random Trees (RRT's) are a very easy method to get to a solution is to randomly start moving from  $q_i$  in order to eventually get to  $q_G$ . The path is build up in a random order and every  $n$  cycles the randomly chosen vertex is forced to be  $q_G$  in order to see if the goal is already reachable. These algorithms can be expanded by both growing a RRT from both  $q_i$  as well as  $q_G$  and try and connect them every  $n$  cycles. These kinds of RRT's are called Bidirectional RRT's.

Just as with PRM's these RRT's have their disadvantages due to their random nature. Furthermore also RRT's are probabilistically complete, as long as the tree is continued to be expanded all configurations are reached. One easy implementation of RRT's is discussed here with more advanced method being omitted for research in future work. A visualization of the pseudo code in Algorithm 10 can be found in Figure 7.7.



---

**Algorithm 10** Rapidly Exploring Random Trees path planning pseudo code

---

```
1: Create empty graph  $\mathbb{G}$ 
2: Add  $q_1$  and  $q_G$  as vertices in  $\mathbb{G}$ 
3: for  $i = 0$  to  $n$  do
4:   Sample a random configuration,  $q_r$  in C-space
5:   Find node  $q_n \in \mathbb{G}$  closest to  $q_r$ 
6:   while edge  $(q_r, q_n)$  crosses  $C_{obs}$  do
7:     Move  $q_r$  closer to  $q_n$  along edge  $(q_r, q_n)$ 
8:   end while
9:   Add  $q_r$  to  $\mathbb{G}$ 
10: end for
11: Run a graph planner (like Dijkstra's or A*) on  $\mathbb{G}$ 
```

---

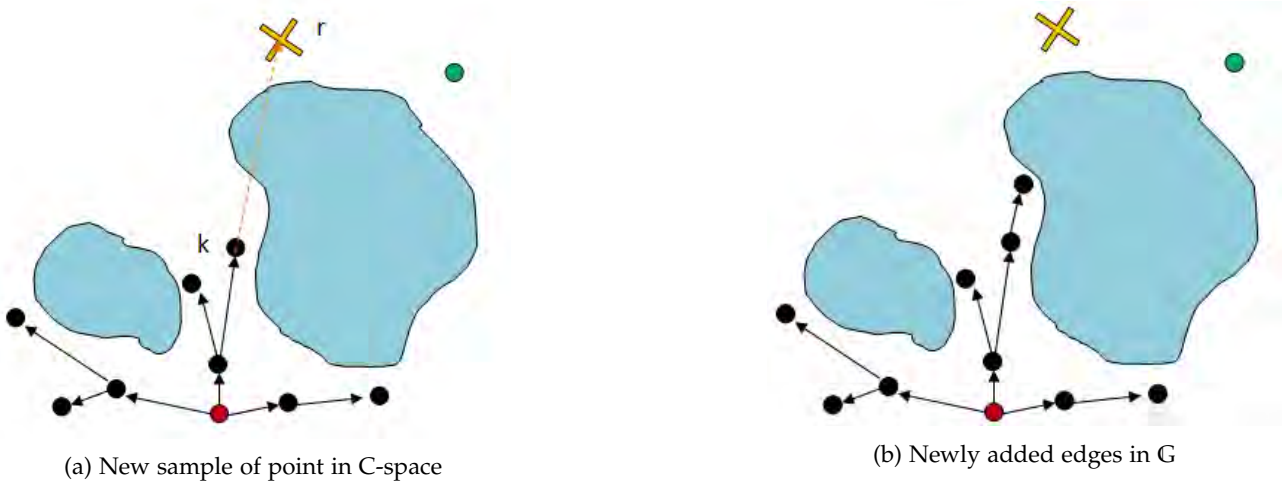


Figure 7.7: Rapidly Exploring Random Trees algorithm visualization [15]

### 7.3 Recommendation

Various path planning algorithms have now been reviewed. Whilst many more variants of the reviewed methods remain the basic concepts are the same. Whilst out of scope for this thesis since the proposed navigation architecture is not in need of a path planning algorithm. This is also due to its inability to produce a map which is needed as per de given definition of the path planning problem.

In the proposed future work the approximate cell decomposition methods can be used once IMU's are used for collision detection due to the proposed use of occupancy grids. These grids closely resemble the needed map reducing computational time needed to compute a path. Once higher levels of the navigation architecture are implemented more degrees of freedom are added to the robot due to the addition of the head. In that case a method from the sample based algorithms could be useful. Due to the eventual goal of giving a guided tour the recommendation is to use Bidirectional RRT's in order to come up with a solution. If however in the future work occupancy grids are still used approximate cell decomposition can remain useful. Or if a graph based mapping technique is used most of the planners can be evaded an Dijkstra's or A\* can simply be used as this is the last step for most of the planners that create graphs.

## Chapter 8

# Robot Operating System

For any robot to work it needs a large code base to control each and every sensor and motor. For this purpose various frameworks have been created over the years. The one that is used in the case of BB-8 is the Robot Operating System (ROS). It has a very large community that makes use of it and thus has an enormous amount of documentation available on the internet to speed up the development process. In order to understand the implementation of the proposed navigation architecture it is essential to understand the basics of this framework.

### 8.1 Overview

ROS is a completely open source framework used in robotics applications, offering many communication tools between processes of the robot or multiple machines on which the robot runs through a TCP/IP network. ROS is mainly split up into two parts:

- **Roscore.** This is the core of the framework, for any of its components to work it needs this part. It offers the various scripts and command-line programs to control and monitor everything that happens whilst the robot is running. This core is maintained by the creators of the ROS framework: Willow Garage.
- **Packages.** The core itself does nothing but enable various programs to use its services. To have the robot actually perform anything a combination of packages is needed. These are all open source and maintained by numerous people from throughout the community. Each of these is tailor made for a specific purpose or robot. Each of these packages runs as a stand alone program, all communicating via the roscore, which is explained in the next subsection.

Another huge benefit of ROS is the ability to write the software in various languages. It supports C++, Python, Lisp officially and has experimental builds available for Java and Lua. Since programs that are written in any of these languages all communicate via the TCP/IP network with the roscore any package of the robot can be written in any of these languages and cause no problems at all. Besides the ability to support many programming languages it is also available on various architecture due to the fact that it runs mainly on

top of Ubuntu and Mac OS, but with the aid of the community various ports have been created to other linux distributions. This enables it to run on both ARM and X86 architectures and more, offering even more variation in which hardware can be chosen for the robot.

## 8.2 Communication

The real power of ROS is in its ability to enable communication between all of the packages. Each of these packages is called a node once run and each of these nodes then can communicate with another via the TCP/IP protocol. This is done by letting the nodes connect to another node via sockets, where each node has its own port. The roscore keeps track of every node that exists and with which nodes they communicate via the so called topics or services. All of the message passing is done via roscore which also enables the message passing to be asynchronous, it also provides nodes the ability to read previous messages because the roscore keeps a short history of the sent messages. The communication between nodes is also completely anonymous, since a node does not know to which other nodes its messages are sent, only the roscore has that information. As mentioned ROS also provides the ability to use the same TCP/IP protocol to send messages over WiFi or LAN to other devices on which a different part of the robot runs. ROS however also has the option to provide synchronous communication via services. Because of the fact that ROS has been designed for robots it provides only standard message definitions using simple data types.

A few key concepts that are vital to understand how the communication in ROS happens follows:

- **Nodes.** These are the processes that come from the packages that are run. Each of these does a specific task for the robot and communicates via the network. On startup it will register itself with the roscore using an unique id and also provides information on which topics it subscribes to and publishes messages on. Besides the topics it also passes information to the roscore which services it can provide for other nodes to call. As mentioned these can be written in various languages offering large compatibility.
- **Master.** This is a special node set up by the roscore, which does everything mentioned above like handling the registration, subscriptions, publications and disconnections of each node. Thus it also handles every message that is passed on by a publisher to each of the subscriber nodes on the given topic and storing them for a short time allowing the subscribers to also retrieve messages from the past in case that is necessary.
- **Topics.** Every node that communicates does so via a topic, they are the core of ROS' communication system. Each of these topics has its own unique id and unique message type. Thus every node knows exactly what data type or object with data types it has to publish or receives, preventing possible errors since nothing else can be sent through these. Each node can connect to the topics using its name when calling the master. There is no limit on how many subscribers and publishers can exist for a topic.
- **Messages.** These are the actual packets that get sent by the publishers via the network. Each of these is defined by using other message types or simple data type. The available data types are, by default, the following: boolean, string, floats (32 + 64 bits), (unsigned) integers (8 + 16 + 32 bits). Besides these standard data types ROS has two more specifically for ROS: time and duration. By combining these

into messages an object is created, these objects can then be implemented by other messages to create a more complex message type. The power of these thus is that a developer is given the ability to create its own messages that need to be passed via the network.

- **Services.** Where the topics allow for asynchronous message passing services offer synchronous ways to do so, letting nodes call each other directly instead of through the master. A node that provides a service returns data specifically for the query it received by another node. Since a service is synchronous the node providing this service can only handle one request at a time, thus limiting the use of it. Each node that provides a service also advertises it via a topic, providing a unique name for other nodes to call. Thus each node also has a specific message type for the request and response data.

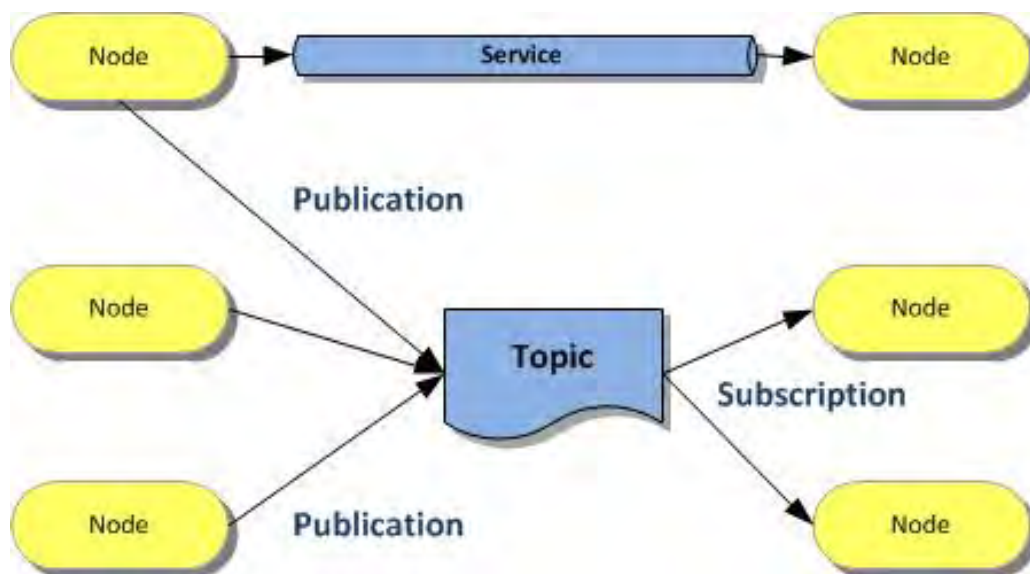


Figure 8.1: ROS communication example [16]

In Figure 8.1 the communication system is visualized. The three nodes on the left publish information on a topic, to which the two bottom nodes on the right are subscribed. These subscribed topics have no way of knowing when a new message might come, they just receive it from the topic via the master when the publishers have one ready. To get data straight away the top right node can request a service from the top left node, thus getting a direct response to its query.

## Chapter 9

# Navigation Architecture

By default a robot would not be able to do anything inside of any environment without the use of the right sensors. Therefore a navigation stack is needed to provide various sensors in order for a robot to be able to navigate autonomously. In this chapter a navigation architecture will be given for the BB-8 in order to provide it with increasing levels of autonomous driving after the adding of each new sensor.

The navigation architecture will be built up in the following order:

1. Wheel encoders - extract rotation of wheels from the motors, thereby calculating the travelled distance. This can be achieved by using the stepper motors completed steps since they have a fixed 200 steps per full rotation.
2. Inertial measurement unit (IMU) - encompasses an accelerometer, gyroscope and compass whose data can be extracted and used for orientation or the traveled distance. With the accelerometer data collisions could be detected by sudden spikes in the data, thereby sensing where objects are.

Future work to increase the effectiveness of this navigation architecture could be built up in the following order:

1. Bluetooth positioning - use Bluetooth as an indoor GPS equivalent by triangulating the robot's location from the calculated distance to various beacons each few seconds. Thus acquiring our absolute position in a building at low frequency, enough to compensate the IMU's habit of drifting values over time.
2. Radar sensor - use radar sensors to peek through the shell in order to find objects in close proximity, eliminating object collision at slow speeds and short distance (under 50cm).
3. Kinect V2 - using the RGBD camera from the Xbox One's Kinect V2 long range (60cm until 4.5m) objects and humans can be detected and thus better evaded, even at higher speeds due to its increased range.

## 9.1 Wheel encoders

As described in the background information, the robot's internal drive platform consists of four omni-wheels, each together with its own stepper motor. Each stepper motor then has its own stepper motor driver to provide the current that is needed for the motor to move at the right time. These stepper motor drivers have built-in registers in which they automatically keep track of how many times current has been sent through the motors, thus keeping track of exactly how many steps have been made. However, steps can of course be lost due to a force that counteracts the motor's movement by, for example, having the robot try to move up a hill. In the case that the hill becomes too steep, the motor will not be able to provide enough force with the wheels on the sphere via its internal magnetic spools to overcome gravity. In the case that happens, the stepper motor will have a so-called step-loss. Luckily, the used stepper motor drivers are able to detect these most of the time and should be able to keep the internal registers at the right value.

The downside to this method is that, although the rotations from each of the motors and wheels can now be calculated, no information is gained for the rest of the robot's movements. Thus the limiting factor here is that if the wheels slip, we do see steps that have been successfully completed but in the absolute motion of the entire robot, no movement has occurred. This means that the wheel encoders can not be more than a good estimate to determine how much the robot has moved. This is, however, a good starting point for the entire navigation stack. Since when the environment is sensed by the robot, it needs to know how to translate the gained information in a map after movement to compare the new sensor data it has then acquired as described in Chapter 4.

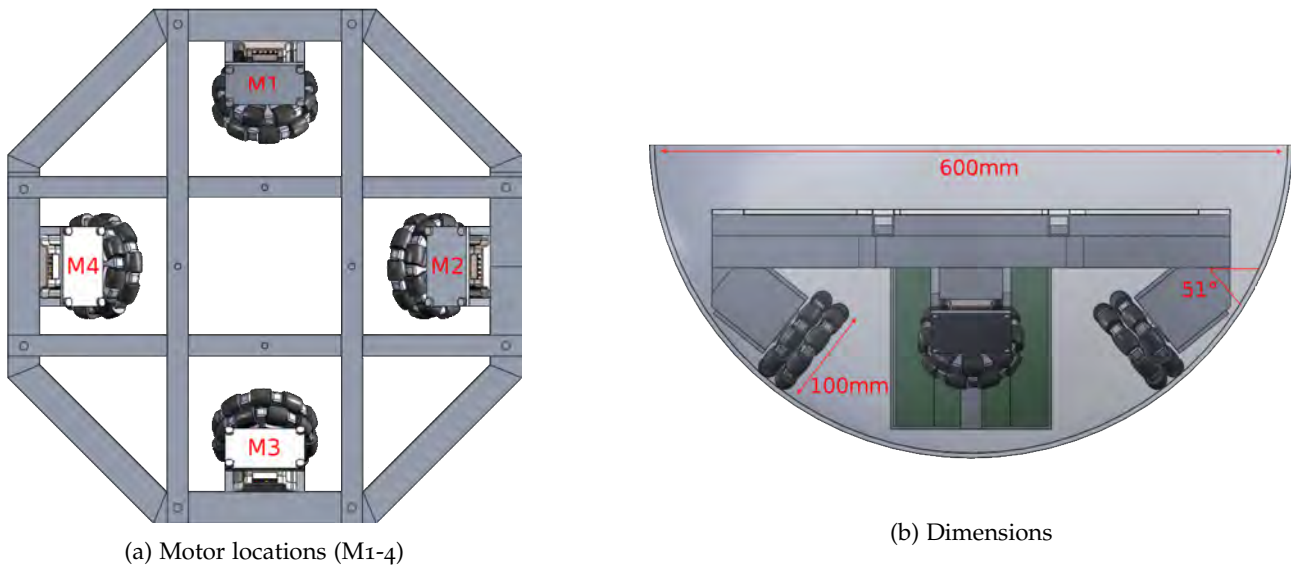


Figure 9.1: Driving mechanism of BB-8

In order to use the wheel data, a simple algorithm is needed to update our position. The pseudo code can be found in Algorithm 11 where the 'getOdometryMeasurements' is the implementation of the algorithms explained below. After the data has been updated, the current position and last known change in position are published in order to use it in the localization step.

---

**Algorithm 11** Odometry algorithm pseudo code

---

```
1:  $x = y = \theta = 0$ 
2: while true do
3:    $\delta x, \delta Y, \delta \Theta = \text{getOdometryMeasurements}$ 
4:    $x = x + \delta x$ 
5:    $y = y + \delta y$ 
6:    $\Theta = \Theta + \delta \Theta$ 
7:    $\text{publishOdometry}(x, y, \Theta, \delta x, \delta Y, \delta \Theta)$ 
8: end while
```

---

### 9.1.1 Four omni-wheel setup

Knowing the amount of steps does not help us get the movement of the entire robot yet, for this we need to combine the data from all four of the motors in order to calculate both angular and linear displacement, also called odometry. By having the ability to calculate the odometry the first problem of the localization step can be solved, by being able to track our position relative to the begin position. In the case of the BB-8 robot the base platform consists of 4 separate motors which are mounted on 2 axis. This means that the entire system could be described as a dual differential drive system in which the axis are orthogonal to each other. For a single differential drive system there are a few formulas for the odometry calculations, as described in [90] and [91]. We describe each motor as  $M_1$ ,  $M_2$ ,  $M_3$  and  $M_4$  and combine them per axis in the groups  $(M_1, M_3)$  and  $(M_2, M_4)$ , as seen in Figure 9.1.

Firstly the simple conversion from steps to distance for each individual motor:

$$\Delta s_i = \frac{2\pi n_i}{N} * R_i \quad (9.1)$$

In this equation  $\Delta s_i$  is the calculated displacement for the motor, with  $1 \leq i \leq 4$  (since there are four separate motors);  $n_i$  is the number of steps in the sampling period;  $N$  the number of steps per full rotation which is 200 in this case; and lastly  $R_i$  the radius of the wheels which is 0.05.

Then to combine the displacement of two wheels to calculate the displacement and rotation for the appropriate axis, which gives the following formulas for the axis with wheel pair  $(M_1, M_3)$ :

$$\Delta s_{(M_1, M_3)} = \frac{\Delta s_1 + \Delta s_3}{2} \quad (9.2)$$

$$\Delta \theta_{(M_1, M_3)} = \frac{\Delta s_1 - \Delta s_3}{b} \quad (9.3)$$

Where  $\Delta s_1$  and  $\Delta s_3$  are calculated as per Equation 9.1; and  $b$  is the length of the wheel base which is 0.60, or simply said the distance between two of the wheels. Equations 9.2 and 9.3 can be changed to get the corresponding values for the other axis,  $(M_2, M_4)$ , by substituting  $M_1$  with  $M_2$  and  $M_3$  with  $M_4$ , thus also substituting  $\Delta s_1$  with  $\Delta s_2$  and  $\Delta s_3$  with  $\Delta s_4$  (given that for the other axis  $b$  does not change).

One thing that should not be forgotten and added to Equations 9.2 and 9.3 in the case of BB-8 is the transmission. Since the driving mechanism is inside a ball, the distance that the wheels travel is not equal to the absolute distance that the robot has traveled. The wheels are as said located on a  $51^\circ$  angle with the horizon, as seen in Figure 9.1. This means that whenever the wheels move inside of the ball they travel a shorter

absolute distance then the robot as a whole. To calculate this offset we take the circumference of the entire ball with radius  $0.3m$ :  $0.3m * 2 * \pi = 1.885m$ . Then to take the distance the wheels travel in a full rotation we use our angle but then with the vertical axis, which is then  $39^\circ$ :

$$0.3 * \cos(39^\circ) * 2 * \pi = 1.465$$

This ends up giving us a transmission of

$$\frac{1.885}{1.465} = 1.287$$

So now to incorporate this in the formula we get for wheel pair  $(M_1, M_3)$  the new formula for the traveled distance:

$$\Delta s_{(M_1, M_3)} = \frac{\Delta s_1 + \Delta s_3}{2} * 1.287 \quad (9.4)$$

Now that the equations for the differential drives have been explained, the two axis can be combined to equate the odometry for the entire robot as described in [92]. First to take the simple average of the two rotational components, as per Equation 9.3 and its substituted version for the other axis, the rotational difference for the base frame is computed:

$$\Delta \theta = \frac{\Delta \theta_{(M_1, M_3)} + \Delta \theta_{(M_2, M_4)}}{2} \quad (9.5)$$

To combine all of the now available data to get the new pose of the BB-8 in total:

$$x_n = x_{n-1} + \Delta s_{(M_1, M_3)} \cos\left(\theta_{n-1} + \frac{\Delta \theta}{2}\right) + \Delta s_{(M_2, M_4)} \cos\left(\theta_{n-1} + \frac{\pi}{2} + \frac{\Delta \theta}{2}\right) \quad (9.6)$$

$$y_n = y_{n-1} + \Delta s_{(M_1, M_3)} \sin\left(\theta_{n-1} + \frac{\Delta \theta}{2}\right) + \Delta s_{(M_2, M_4)} \sin\left(\theta_{n-1} + \frac{\pi}{2} + \frac{\Delta \theta}{2}\right) \quad (9.7)$$

$$\theta_n = \Delta \theta + \theta_{n-1} \quad (9.8)$$

Now that the odometry is known as per Equations 9.6, 9.7 and 9.8, the continued updates can be used by the system to get the traveled distance and update its position over time. The complete algorithm for odometry updates in the case of a four wheel setup can be seen in Algorithm 12.

---

**Algorithm 12** Four wheel odometry

---

```

1:  $x = y = \theta = 0$ 
2: while true do
3:    $d_1, d_2, d_3, d_4 = \text{getWheelMovement}$ 
4:    $\delta \theta = \frac{\frac{d_1 - d_3}{2} + \frac{d_2 - d_4}{2}}{2}$ 
5:    $\delta x = 1.287 * \left( \frac{d_1 + d_3}{2} \cos\left(\theta_{n-1} + \frac{\delta \theta}{2}\right) + \frac{d_2 + d_4}{2} \cos\left(\theta_{n-1} + \frac{\pi}{2} + \frac{\delta \theta}{2}\right) \right)$ 
6:    $\delta y = 1.287 * \left( \frac{d_1 + d_3}{2} \sin\left(\theta_{n-1} + \frac{\delta \theta}{2}\right) + \frac{d_2 + d_4}{2} \sin\left(\theta_{n-1} + \frac{\pi}{2} + \frac{\delta \theta}{2}\right) \right)$ 
7:    $x += \delta x$ 
8:    $y += \delta y$ 
9:    $\Theta += \delta \Theta$ 
10:   $\text{publishOdometry}(x, y, \Theta, \delta x, \delta y, \delta \Theta)$ 
11: end while

```

---



### 9.1.2 Alternative three omni-wheel setup

Another setup that can be used is the one with three omni-wheels. Due to the way the platform of the robot is setup in its octagonal shape the three omni wheels cannot be placed in perfect 120 degree angles in relation to each other. The advantage of this setup is that while with 4 motors more power is available they also have a higher possibility of slipping. With this setup that risk is greatly reduced since at all times a part of the robot weight rests on one of the wheels, all bearing a part of the total weight, thus resulting in grip. With a 4 motor system this could only be achieved with suspension that will push the wheels down at all times, but that would be harder to implement.

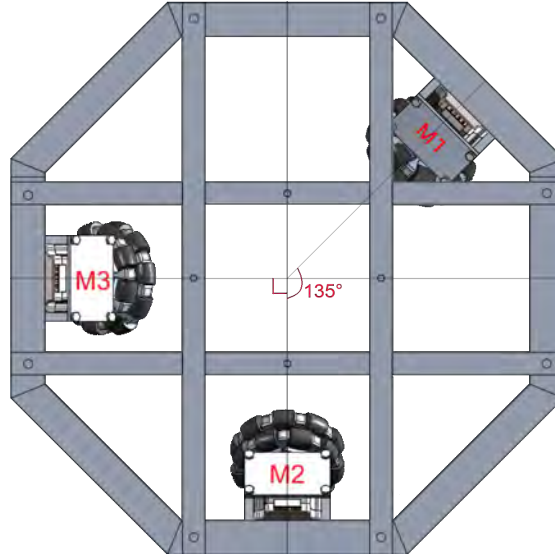


Figure 9.2: Driving mechanism of BB-8 with three omni-wheels

In Figure 9.2 the setup is shown with three omni-wheels. As can be seen the used frame is sub-optimal for this setup, but the extra grip might prove useful. The wheels and motors are positioned in a setup in which M1 is rotated  $135^\circ$  in reference to M2 and M3 which are perpendicular to each other. The motors and wheels used in this setup are the same as with the 4 motor setup. A mathematical explanation for the odometry information will now be given.

To see how the kinematic model of this setup would be the kinematic model from [17] is used and applied on to our setup. Their model has to be altered to accompany for the different angles that are used in this system, they use the perfect separation of  $120^\circ$  between all wheels. In Figure 9.3 the graphic diagram which is used for this model is displayed. In this diagram  $v_1, v_2, v_3$  are the velocities of each of the wheels (M1, M2 and M3), which are shown on the projections of the direction of each wheel. In this diagram the dashed lines ( $r_1, r_2$  and  $r_3$ ) show the normal direction to each of the motors axis'. The lines W1, W2 and W3 indicate the positive direction of rotation of each wheel and on these lines the speed of each separate wheel ( $v_1, v_2$  and  $v_3$ ) is projected.

Now to examine the equations used to calculate the projection line of each wheel  $r_1, r_2$  and  $r_3$ :

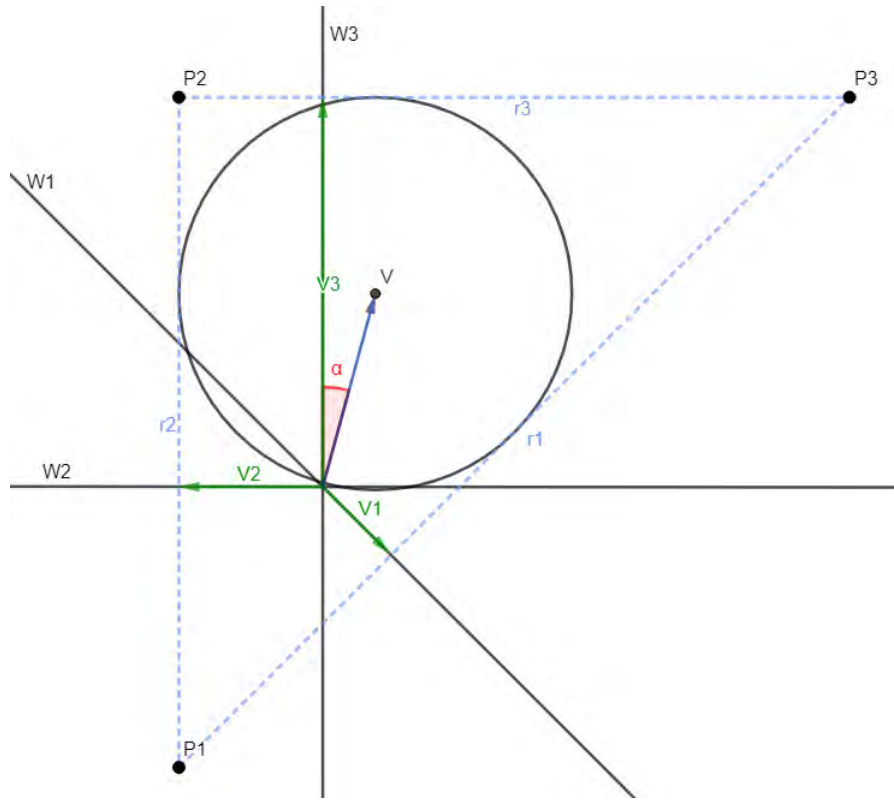


Figure 9.3: Graphic diagram for three wheeled kinematic model, adjusted from [17] for the current setup

$$r1 : y = \tan\left(\frac{1}{4}\pi\right) * x + ((-v1) * \cos\left(\frac{1}{4}\pi\right) + (-v1) * \tan\left(\frac{1}{4}\pi\right) * \sin\left(\frac{1}{4}\pi\right))$$

$$y = x - \sqrt{2} * v1$$

$$x = y + \sqrt{2} * v1 \quad (9.9)$$

$$r2 : x = -v2 \quad (9.10)$$

$$r3 : y = v3 \quad (9.11)$$

Next these three Equations 9.9, 9.10 and 9.11 can be used to obtain the cutoffs of  $r1$ ,  $r2$  and  $r3$ :

$$r1 \leftrightarrow r2 : P1 = (-v2, -v2 - \sqrt{2} * v1) \quad (9.12)$$

$$r2 \leftrightarrow r3 : P2 = (-v2, v3) \quad (9.13)$$

$$r3 \leftrightarrow r1 : P3 = (v3 + \sqrt{2} * v1, v3) \quad (9.14)$$

Combining these three cutoff points from Equations 9.12, 9.13 and 9.14 results in the centroid of the triangle  $V$ :

$$V = \left( \frac{P1_x + P2_x + P3_x}{3}, \frac{P1_y + P2_y + P3_y}{3} \right) = \left( \frac{\sqrt{2} * v1 - 2 * v2 + v3}{3}, \frac{-\sqrt{2} * v1 - v2 + 2 * v3}{3} \right) \quad (9.15)$$

Equation 9.15 now gives us the  $x$  and  $y$  displacement of the robot. Now to be able to display the robot in polar coordinates we need velocity  $v$ , angular orientation  $a$  and angular rotation  $w$ .

$$v = \sqrt{V_x^2 + V_y^2} = \sqrt{\left( \frac{\sqrt{2} * v1 - 2 * v2 + v3}{3} \right)^2 + \left( \frac{-\sqrt{2} * v1 - v2 + 2 * v3}{3} \right)^2} \quad (9.16)$$

$$a = 90 - \tan^{-1} \left( \frac{V_y}{V_x} \right) = 90 - \tan^{-1} \left( \frac{-\sqrt{2} * v1 - v2 + 2 * v3}{\sqrt{2} * v1 - 2 * v2 + v3} \right) \quad (9.17)$$

$$w * R = Gx - r2 = \frac{\sqrt{2} * v1 - 2 * v2 + v3}{3} - (-v2) = \frac{\sqrt{2} * v1 + v2 + v3}{3}$$

$$w = \frac{\sqrt{2} * v1 + v2 + v3}{3 * R} \quad (9.18)$$

All combined into one algorithm is found for the case of this thesis' robot, which can be seen in Table 13. In this algorithm the traveled distances of each wheel is used ( $d_i$ ) and the radius of the circle on which the platform rests ( $R$ ).

---

**Algorithm 13** Three wheel odometry

---

```

1:  $x = y = \theta = 0$ 
2: while true do
3:    $d_1, d_2, d_3 = \text{getWheelMovement}$ 
4:    $\delta x = \frac{\sqrt{2} * d_1 - 2 * d_2 + d_3}{3}$ 
5:    $\delta y = \frac{-\sqrt{2} * d_1 - d_2 + 2 * d_3}{3}$ 
6:    $\delta \Theta = \frac{\sqrt{2} * d_1 + d_2 + d_3}{3 * R}$ 
7:    $x+ = \delta x$ 
8:    $y+ = \delta y$ 
9:    $\Theta+ = \delta \Theta$ 
10:   $\text{publishOdometry}(x, y, \Theta, \delta x, \delta y, \delta \Theta)$ 
11: end while

```

---

### 9.1.3 Calibration

No odometry system is perfectly aligned and calibrated out of the box, they all have slight deviations on many fronts. To accompany for the errors in an odometry system there is a need to calibrate it as precise as possible. For the three wheeled setup an adaption of the calibration method provided by [4]. Their method is quite simple and consists of several test runs to gather some data of the approximate errors, which are then fed into a few formulas which give the calibration values. The steps to follow are:

1. Setup robot at the start position. Make sure the orientation angle is also zero.

2. Let the robot move along a straight line straight forward, stopping after a predetermined length of travelling.
3. Once it has moved, measure the robots absolute position with respect to the reference coordinates it was supposed to move to.
4. Repeat steps 1-3 a few times for some more data to improve calibration result.
5. Calculate longitudinal, lateral and rotation errors for each test run.
6. Calculate the correction matrices as defined by [4] and seen below in Algorithm 14.
7. Adjust each of the motors rotational speed with the resulting correction matrices from step 6.

---

**Algorithm 14** Odometry calibration [4]

---

$$\begin{aligned}
1: R(\theta) &= \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
2: J_{n \times 3}^{-1} &= \text{diag} \left[ \frac{1}{r_1 \cos(\gamma_1)} \cdots \frac{1}{r_n \cos(\gamma_n)} \right] \begin{bmatrix} \sin(\alpha_1 + \beta_1 + \gamma_1) & -\cos(\alpha_1 + \beta_1 + \gamma_1) & -l_1 \cos(\beta_1 + \gamma_1) \\ \vdots & \vdots & \vdots \\ \sin(\alpha_n + \beta_n + \gamma_n) & -\cos(\alpha_n + \beta_n + \gamma_n) & -l_n \cos(\beta_n + \gamma_n) \end{bmatrix} R(\theta) \\
3: \begin{bmatrix} \dot{\phi}_{e,1} \\ \dot{\phi}_{e,2} \\ \vdots \\ \dot{\phi}_{e,n} \end{bmatrix} &= J^{-1} \begin{bmatrix} 0 \\ 0 \\ -\dot{\theta}_e \end{bmatrix} \\
4: F_{lat} &= \text{diag} \left[ 1 + \frac{\dot{\phi}_{e,1}}{\dot{\phi}_1} \cdots 1 + \frac{\dot{\phi}_{e,n}}{\dot{\phi}_n} \right] \\
5: F_{lon} &= \frac{L}{\sqrt{(L + \bar{x}_e)^2 + (\bar{y}_e)^2}} \\
6: J_c^{-1} &= F_{lon} F_{lat} J^{-1} \text{diag} \left[ \left( 1 + \frac{\dot{\phi}_{e,1}}{\dot{\phi}_1} \right) \frac{1}{r_1 \cos(\gamma_1)} \cdots \left( 1 + \frac{\dot{\phi}_{e,n}}{\dot{\phi}_n} \right) \frac{1}{r_n \cos(\gamma_n)} \right] \times \\
&\quad \begin{bmatrix} F_{lon} \sin(\alpha_1 + \beta_1 + \gamma_1) & -F_{lon} \cos(\alpha_1 + \beta_1 + \gamma_1) & -F_{lon} l_1 \cos(\beta_1 + \gamma_1) \\ \vdots & \vdots & \vdots \\ F_{lon} \sin(\alpha_n + \beta_n + \gamma_n) & -F_{lon} \cos(\alpha_n + \beta_n + \gamma_n) & -F_{lon} l_n \cos(\beta_n + \gamma_n) \end{bmatrix} R(\theta)
\end{aligned}$$


---

To explain the formula each line from the algorithm is analyzed, for a complete mathematical deduction see the referenced paper by [4]. In the first line  $R(\theta)$  is simply the rotation matrix used to convert the robot's movements from its current orientation,  $\theta$ , to the global reference frame. Next the inverse Jacobian is given, in this formula the  $n$  (number of wheels) is to be filled. A few more parameters per wheel are to be filled in as well, starting with the orientation of each wheel in reference to the robot's frame,  $\alpha_i$ .  $\beta_i$  denotes the angle of the wheel relative to the robots main body.  $\gamma_i$  is the angle between the rotation axis of the smaller circumferential rollers and the wheels main plane. And lastly  $r_i$  is the radius of the wheel. For line 3 the only parameter left to explain is the  $\dot{\theta}_e$ , which is simply obtained by taking the measured error angle  $\theta_e$  and dividing it by the time  $t$  taken to complete the calibration run. Line 4 uses the outcome of line 3 together with the nominal angular velocities per wheel,  $\dot{\phi}_i$ . In line 5 the longitudinal error is calculate by taking the x and y errors,  $\bar{x}_e, \bar{y}_e$ , together with the length of the traveled calibration trajectory,  $L$ . Then in line 6 the outcome of all of the other parts of the calibration formula is taken together to get the corrected inverse Jacobian matrix,  $J_c^{-1}$ . This  $J_c^{-1}$  is to be applied to the given movement vector to correct the wheel speeds to let the robot actually follow the movement vector as precise as possible. Some of these variables can already be given for the robot.

These variables, found in Table 9.1, can be applied to the calibration algorithm above.

Table 9.1: Calibration parameters

Specification	Dimension
Number of wheels, $n$	3 or 4
Wheel radius ( $r_i$ )	0.05
Wheel steering angle ( $\beta_i$ )	0
Roller axis angle ( $\gamma_i$ )	0
Wheel shaft angle ( $\alpha_i$ ) in 4 wheel setup	$0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}$
Wheel shaft angle ( $\alpha_i$ ) in 3 wheel setup	$0, \frac{\pi}{2}, \frac{5\pi}{3}$

## 9.2 Inertial measurement unit

Having just the computed odometry from the wheel encoders gives a lot of information of the robots new position over time, however it is unable to detect any form of slippage in an easy manner. Another deficit the robot's shape brings with it is that due to its single point of contact with the ground it might rotate over time which it would be unaware of looking at just the encoder data. To compensate for this extra sensors are needed which try to keep track of the robots position by measuring external forces such as gravity and magnetism. For this the Inertial Measurement Unit (IMU) has been created, which hosts an accelerometer, a gyroscope and a compass.

These sensors are available in various forms and range from tens up to thousands of euro's. The more expensive ones are capable of more accurate and less noisy measurements in combination with a higher read out frequency. The only downside they all have is their tendency to drift over time, meaning if the robot remains stationary the IMU sensor might show a (very) slow continued movement. This would eventually lead to thinking the robot might be in an upside down position, this effect is once again reduced when more expensive and precise IMU's are used. This comes from the more precise measurements provided by them which reduce the error introduced when the acceleration data is integrated. An overview is given in Table 9.2 where the various grades of quality of IMU's are given with their respective properties.

Seeing all of the various grades in Table 9.2 and furthermore keeping in mind the fact that the robot is to be used in an indoor environment a tactical or navigational grade IMU would be preferable. Going any lower in grade will dramatically increase our errors, going to over 120km/h of positioning error on automotive grade or 180km/h or more on consumer grade IMU's. Whereas the navigation and tactical grade IMU's offer the best precision they are, as mentioned above, very expensive going to thousands of euro's quickly. Therefore this would not be an option, so the higher end automotive grade IMU's are used, which in the case of BB-8 would be the Phidgets 1044 Spatial Precision 3/3/3 High Resolution IMU. This IMU offers the precise, high end automotive grade, sensors combined with the lesser automotive grade sensors as backup in case the change in speed becomes too high to correctly measure with the high precision sensors. This also enables the robot to keep working under those conditions by using the lower precision sensors as a fail-safe.

Table 9.2: Performance of different grades of inertial sensors. [2]

Grade	Navigation		Tactical		Automotive		Consumer	
Sensor type	Gyro	Accel.	Gyro	Accel.	Gyro	Accel.	Gyro	Accel.
Bias	0.005 - 0.01 (deg/hr)	5 - 10 ( $\mu$ g)	1 - 10 (deg/hr)	200 - 500 ( $\mu$ g)	150 - 180 (deg/hr)	1200 ( $\mu$ g)	360 (deg/hr)	2400 ( $\mu$ g)
Scale Factor	5 - 50 ppm	10 - 20 ppm	200 - 500 ppm	400 - 1000 ppm	—	—	—	—
Noise	0.002 - 0.005 deg/hr/Hz	5 - 10 g /hr/Hz	0.2 - 0.5 deg/hr/Hz	200 - 400 g /hr/Hz	—	—	—	—
Positioning Error	$\sim 2$ (km/hr)		$\sim 20-40$ (km/hr)		$\sim 2$ (km/min)		$\sim 3$ (km/min)	

### 9.2.1 Madgwick

As mentioned the IMU's present quite noisy data, to compensate for this and get good results they will have to be passed through a filter. The go to filter was a Kalman filter for many years, however these become quite heavy in situations where multiple data sources need to be combined. This meant that for IMU's, which give up to 9 types of data (3 axis for each of the included sensors), another solution needed to be found. This has been done in the form of the Madgwick filter [93] which is a simpler filter specially optimized for IMU's. Another benefit this optimization brought with it is the fact that it is even more precise than the usual Kalman filters, producing only a error of up to 0.7 degrees compared to the Kalman filters maximum of 1.3 degrees of error.

This filter has the option to be implemented with or without the use of the compass. By using the compass it could even better accomplish its task of eliminating gyroscope drift. However due to the fact that this robot has stepper motors which create temporary magnetic fields this might not be useful. This has to be seen during calibration if these temporary magnetic fields can be eliminated or the magnetic sensor should not be used. To accomplish the implementation of this filter the use of the ROS package `imu_filter_madgwick` is used. To see the implementation of this package and how it is connected to services and topics in ROS please go to Appendix B.

### 9.2.2 Sensor fusion

Now that there are multiple data sources for the robot's position and heading a way to fuse these correctly is necessary. This is typically the area in which Kalman filters are used, specifically an Extended Kalman Filter which can use non linear measurements. This filter gives the robot the possibility to fuse various data

sources, each of which of course is more or less noisy, to get a better, less noisy, pose estimate. To recall from Section 4.4: EKF can be used for any configuration of sensors, only the state transition model has to be adjusted to allow the correct use of every data source. The use of the Extended Kalman Filter in this case would be to take the orientational data from the IMU to compensate for the wheel encoders inability to correctly determine slight rotations after movement. It's accelerometer data could also be used to check the wheel encoder's velocity data by integrating it, and once more its actual position by integrating the velocity data. This last part however could turn out to not be useful because of the error introduced in the (double) integration.

All of this is possible with the use of the `robot_localization` package [94], which when set up correctly can take in numerous data sources. For each of these data sources exactly which data is fed into the filter can be configured and thus used in correctly estimating our pose. In the case of the wheel encoders it uses the  $x$  and  $y$  position and the  $\dot{x}$  and  $\dot{y}$  speeds. This is then combined with the *pitch*, *roll* and *yaw* of the IMU, together with their corresponding speeds *pitch*, *roll* and *yaw*. Lastly the acceleration in the  $x$  and  $y$  plane,  $\ddot{x}$  and  $\ddot{y}$  is used by integrating it to check the speeds and once again by integrating the speeds to check the position. As with the Madgwick filter the implementation of this filter into the system can be seen in the Appendix B.

### 9.2.3 Calibration

As with any sensor calibration is needed, especially once forces that differ all over the world are measured, as is the case with an IMU which measures gravity and earths magnetic field. The way to calibrate this is rather simple, the earths gravitational acceleration is to be measured for the used IMU in each for each of the 3 axis during stand still. To do so each of these three axis is to be put perpendicular to the ground which can simply be done by using a plumb to which the IMU is mount perfectly parallel to. After having done so each of the axis acceleration measurement can be read out before being rotated  $180^\circ$  to measure the respective axis negative acceleration measurement. Using both the negative and positive readouts of each of the accelerometers axis the bias of each axis can be determined by the simple average of those two readings. Each of these rotations is done in a particular order as to incorporate the gyroscope calibration at the same time. To get the gyroscope bias the robot is to remain perfectly still, then after a short while the value can be read out to see how much the gyroscope has drifted. This order allows each of the gyroscope axis to be tested in a good fashion to get the right gain values. The order to use is as follows (there are of course variations possible, as long as each axis of both sensors it put through its paces):

1. Place the IMU in the Zup orientation, record the Zup accelerometer measurement and gyro bias vector.
2. Rotate the IMU about the X axis 180 degrees to the Zdown orientation. Record the integrated gyro measurements during the rotation around X axis. Record the Zdown accelerometer measurement.
3. Rotate the IMU about the Y axis 90 degrees to the Xdown orientation. Record the Xdown accelerometer measurement.
4. Rotate the IMU about the Y axis 180 degrees to the XupP orientation. Record the integrated gyro measurements during the rotation around Y axis. Record the Xup accelerometer measurement.

5. Rotate the IMU about the Z axis 90 degrees to the Ydown orientation. Record the Ydown accelerometer measurement.
6. Rotate the IMU about the Z axis 180 degrees to the Yup orientation. Record the integrated gyro measurements during the rotation around Z axis. Record the Yup accelerometer measurement.

The calibration formula for IMU's are given by [5]. They express the true acceleration and angular velocity vectors as seen in Algorithm 15. In that formula the measured values from the IMU are corrected according to parameters acquired later in this section to give the calibrated outcome. To understand the actual calibration formula understanding the formula from Algorithm 15 is essential. The calibration algorithm they propose is a new one where Fourier Transforms are used to calculate the parameters. They have compared this method with the common Recursive Least Squares (RLS) and due to the simplicity and small differences in actual outcome after calibration the more easy RLS method will be used here to calibrate the IMU.

---

**Algorithm 15** Correction formula to calibrate and align IMU [5]

---

$$\begin{aligned}
F &= [F_x \quad F_y \quad F_z]^t \\
\delta F &= [\delta F_x \quad \delta F_y \quad \delta F_z]^t \\
S_f &= \begin{bmatrix} k_x^F & 0 & 0 \\ 0 & k_y^F & 0 \\ 0 & 0 & k_z^F \end{bmatrix} \\
M_f &= \begin{bmatrix} 0 & m_{xy}^F & m_{xz}^F \\ m_{yx}^F & 0 & m_{yz}^F \\ m_{zx}^F & m_{zy}^F & 0 \end{bmatrix} \\
B_F &= [b_x^F \quad b_y^F \quad b_z^F]^t \\
F &= (S_F + M_F)(\delta F - B_F) + w_F
\end{aligned}$$


---

To explain Algorithm 15, in line 1 the true acceleration or angular rate is defined as  $F$ . In line 2  $\delta F$  depicts the measured acceleration or gyro values from the IMU.  $S_F$  in line 3 is the scale factor, to accompany for wrong scaling at higher acceleration or angular rates. To make sure the acceleration and gyro sensors from the IMU are aligned to their correct axis  $M_F$  is defined in line 4. In line 5 the last parameter of significant interest is shown which is the bias  $B_F$ . Lastly in line 6 the entire correction formula is shown into which all the variables should be entered to get the corrected values from the IMU which are then as close as possible to the real  $F$  values. In this last line another parameter is introduced which adds the noise terms,  $w_F$ . These noise terms are assumed to be white Gaussian.

To actually get the parameters needed for the alignment formula given by Algorithm 15 the RLS method is to be applied on the acquired values from the measurements. In order to do this [5] rewrites line 6 from Algorithm 15 to the following:

$$\begin{aligned}
F &= (S_F + M_F)(\delta F - B_F) + w_F \\
&= A_F^S s_F + A_F^M m_F - B_F' + w_F \\
&= \begin{bmatrix} A_F^S & A_F^M & -I_{[3 \times 3]} \end{bmatrix} \begin{bmatrix} s_f & m_f & B_F' \end{bmatrix}^T + w_F
\end{aligned} \tag{9.19}$$



To explain the last line of this equation with known variables:

$$\begin{aligned}
A_F^S &= \begin{bmatrix} \delta F_x & 0 & 0 \\ 0 & \delta F_y & 0 \\ 0 & 0 & \delta F_z \end{bmatrix} \\
s_f &= \begin{bmatrix} k_x^F & k_y^F & k_z^F \end{bmatrix}^T \\
m_f &= \begin{bmatrix} m_{xy}^F & m_{xz}^F & m_{yx}^F & m_{yz}^F & m_{zx}^F & m_{zy}^F \end{bmatrix}^T \\
B_F' &= \begin{bmatrix} b_x^{F'} & b_y^{F'} & b_z^{F'} \end{bmatrix}^T
\end{aligned}$$

The standard RLS method for which line 6 of algorithm 15 is adjusted is as follows:

$$z = H\Theta \quad (9.20)$$

To show the relation between Equation 9.20 and 9.19 the following is given:

$$\begin{aligned}
z &= F \\
H &= \begin{bmatrix} A_F^S & A_F^M & -I_{[3 \times 3]} \end{bmatrix} \\
\Theta &= \begin{bmatrix} s_f & m_f & B_F' \end{bmatrix}^T
\end{aligned}$$

The estimated parameter matrix  $\hat{\Theta}$  is then given by [5] as follows:

$$\hat{\Theta}_n = \hat{\Theta}_{n-1} + K_n(z_n - H_n \hat{\Theta}_{n-1}) \quad (9.21)$$

In Equation 9.21 the actual parameters needed to enter into the correction formula are given. In this equation the following variables were used:

$$\begin{aligned}
K_n &= P_{n-1}^{-1} H_n^T (H_n P_{n-1}^{-1} H_n^T + I_{[3 \times 3]})^{-1} \\
P_n &= (I_{[3 \times 3]} - K_n H_n) P_{n-1}^{-1}
\end{aligned}$$

## Chapter 10

# Evaluation of dead reckoning capabilities

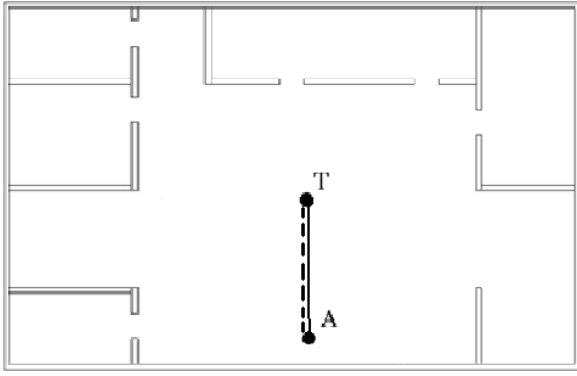
In this chapter the evaluation of the robot shall be explained. Firstly the test setups will be shown and how the robot should move through them in Section 10.1. After the various setups have been defined the parameters with which they are evaluated are explained in Section 10.2. Lastly the results of the various levels will be shown and compared to one another in section 10.3. In this comparison the goal is to show that every level adds to the robots overall perception of its pose. These levels will be compared when fused together, expecting each to increase the autonomous capabilities. In this way a good conclusion can be drawn as to if the level actually benefits the robot. When drawing a conclusion it is important to take in mind the research goals of defining an navigation architecture that is at the basis of an more elaborate architecture used for guided tours.

### 10.1 Experimental test setups

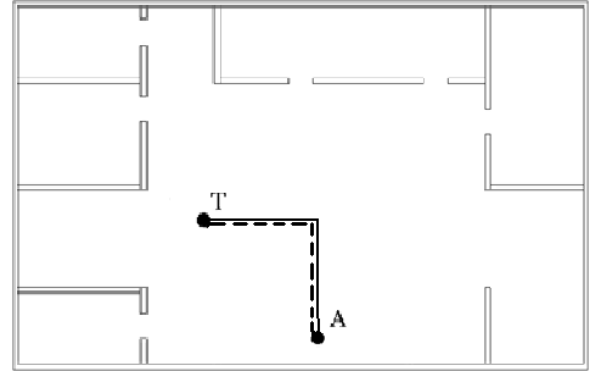
To compare the various levels of the navigation architecture a good test setup has to be defined. The test setup will compare the navigation architecture levels in various ways. To start of the robot will be set up at its home position with no rotation, meaning at position with its heading straight forward  $(x, y, \theta)$  is  $(0, 0, 0)$ . From this position the robot will be guided by hand, with the use of a Playstation 3 (PS3) controller connected to the robot over bluetooth, to a new position. From this new position it should then find its way back to its home position by using one or more of the navigation architecture levels. This new position will be called the 'let go' point for easy of use.

The tests will be built up in difficulty in the following order, ranging from simple to increasingly harder tasks:

1. Moving forward along a straight line.
2. Incorporating a turn into the path after which the robot now has to backtrack, this turn is made whilst stationary.



(a) Example of task 1, moving back from a straight line forward



(b) Example of task 2, backtracking the turn

Figure 10.1: Test setup examples.

Figure 10.1 shows the two test setups, each of these differing in their level of difficulty. In Figure 12.5a the first setup is shown where the robot moves forward in a straight line and has to backtrack it again. This should be relatively simple for the robot to accomplish since it can keep moving in a straight line without calculating turns etc. In Figure 12.5b turns are introduced, which the robot is to backtrack correctly.

Each of these tests will be conducted in order on a per architecture level basis. This means the first task is to be done by the odometry level alone, followed by introducing the IMU. Furthermore only the three wheeled setup is used when obtaining the results. The reason for excluding the four wheeled setup is given in the results section. For each of the tasks the length is increased to make the task harder in increments of one meter up to tree meter, thus one, two and three meter. These path lengths are the same for the task with turns but with a 90 degree turn followed by a the same length of the path driven before the turn. The length is increased in the same way as with the straight line, thus resulting in three different lengths of one, two and three meter. All of these test runs will be performed at a slow speed in order to reduce sudden stops and turns of the sphere resulting in even higher errors. Per architecture level a total of 10 test runs is performed per length of a setup. This sums up to a total of 60 test runs per setup with 120 runs in total.

## 10.2 Evaluation of experimental test setups

The defined experimental setups have to be evaluated in a well described manner in order to compare them. This evaluation is done for each of the setups individually. A few evaluation parameters can be defined here in terms of precision and difficulty. All parameters are listed here in Table 10.1.

Parameter	Explanation
Precision	$(x, y)$ deviation from the home position in m
Orientation	$\theta$ deviation from starting orientation in radians
Length of path	Length of the task that is completed

Table 10.1: Evaluation parameters

The precision of each of the levels is the most important parameter to compare them. The main goal is to achieve an precision as close as possible to being absolute. This is very important for the future work when it is needed in order for the robot to be able to drive through narrow spaces such as doors. A less important parameter, but still one that can be measured easily, is the orientation. An orientation the same as the starting orientation, namely  $\theta = 0$  is preferred. However due to the omni-directional capabilities of the robot it is insignificant which orientation the robot is in. Especially considering the future work the head is capable of rotating freely. This means that any sensors embedded there can observe every part of the world around the robot without it needing to be in a particular orientation. Lastly the length of the path is another important parameter, as a limit of the dead reckoning is expected due to no absolute positioning system being available now. The absence of an absolute positioning system would lead to an inevitable drift over time, which should thus limit the maximum length before the precision becomes awful.

## 10.3 Results

In this section the results of each of the navigation architecture levels will be given. In the same way as the levels were defined in the navigation chapter firstly the wheel odometry is tested. Afterwards the IMU is added and combined with the wheel odometry in order to increase the awareness of the robots orientation. Lastly these levels are compared to one another.

The results are given in tables with a graph illustrating the final position. In these graphs the position is shown as a small dot with the orientation being given by an arrow, which has its base at the dot. In all of the graphs the orange marker indicates the perfect orientation and home position to which the robot is to return in case of perfect autonomous navigation. Some overall remarks will be made after the results have been given since they apply for all results of one of the two navigation architecture levels. These remarks will explain some of the errors found in the results.

### 10.3.1 Straight lines

In this subsection the results for the straight line test are discussed. Starting with the odometry only results for all three lengths followed by the IMU results. The IMU results are then compared to the Odometry only results to see if any improvement has been made when the IMU is introduced.

To start of with the odometry only results for the path length of 1m from Table 10.2 depicted in red in Figure 10.2. It is clear to see that when the length of the path stays under 1m the this method is not optimal in terms of precision due to the outliers towards  $X_e$  0.2 and  $Y_e$  0.15. However nearly all of the results keep the orientation fairly directed at the let go point, meaning it does keep a sense of direction.

The odometry only results for a path length of 2m in Table 10.4, once again depicted in red in Figure 10.3 show a different story. These results show the loss of directional awareness with the orientation pointing in

various directions. Probably due to this loss of direction the precision is also lost. This loss of precision is clearly shown with some results closer to the let go point where to it was guided than the origin where it was supposed to go to. However, all of the results still ended up in the direction of the origin when looking from the let go point.

Lastly to discuss the odometry only results the longest path length of 3m in Table 10.6 depicted in red in Figure 10.4. These results show a complete loss of directional awareness with the orientation pointing in even more extreme directions when compared to the 2m path length test runs. This loss of direction is so extreme that in one case the robot ends up further away from its home position then the let go point is.

Now to get the results where the IMU was introduced to be combined with the odometry. The results a path length of 1m for this combination are shown in 10.3 depicted in blue in Figure 10.2. All of the resulting positions are within a few centimeters of the Y-axis, all resulting orientations are aiming towards the let go point. This shows a slight, but not significant, improvement over the odometry only method.

The IMU and odometry combination results for a path length of 2m are shown in 10.5 and depicted in blue in Figure 10.3. It is clear to see that in all cases the robot keeps a sense of direction with all resulting orientations pointing towards the let go point. Once again all resulting positions come close to the Y-axis within a deviation only in the X-axis of just over 0.5m. When comparing these results to the odometry only method a significant improvement can be seen. This is probably due to its ability to better track its orientation when the sphere itself starts to rotate.

Lastly the results for the IMU and odometry combination for a path length of 3m as seen in in 10.7 and depicted in blue in Figure 10.4. As with the previous two test run lengths the robot keeps its orientation pointed towards the let go point and the resulting positions coming close to the Y-axis. However, Figure 10.4 shows a deformed depiction of the results to the one massively erroneous result from the odometry only method around  $X_e$  1.5 and  $Y_e$  4.0. Thus the significant improvement over the odometry only method is slightly less obvious. This improvement can however be deducted from the near perfect following of the depicted circle, with a lot higher precision when looking towards the  $Y_e$ .

x error (m)	y error (m)	platform orientation (rad)
0.09	0.03	0.3
0.02	0.04	0.1
0.17	0.18	0.3
0.02	0.03	0.2
-0.03	0.02	0.0
0.21	0.15	0.2
0.06	0.00	0.2
0.18	0.14	0.2
-0.05	0.04	-0.2
0.17	0.16	0.5

Table 10.2: Results straight line 1m odometry only

x error (m)	y error (m)	platform orientation (rad)
0.05	0.02	-0.1
-0.05	0.04	-0.2
-0.03	0.01	0.0
0.09	0.01	0.1
0.19	0.03	0.3
0.15	0.02	0.0
0.09	0.03	0.1
0.09	0.02	0.1
0.14	0.02	0.1
-0.04	0.01	-0.1

Table 10.3: Results straight line 1m IMU + odometry

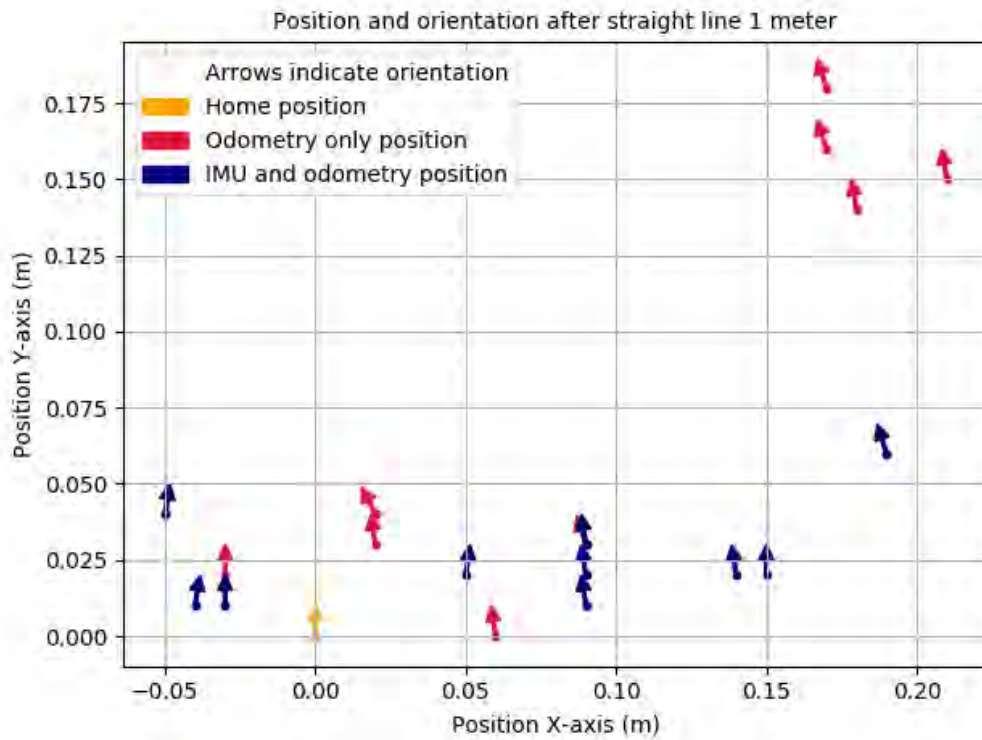


Figure 10.2: Visualization of Tables 10.2 and 10.3 for first test setup at length 1m

x error (m)	y error (m)	platform orientation (rad)
-1.13	1.49	1.2
-0.31	1.09	-1.5
-0.36	0.71	0.5
0.12	0.13	-0.7
0.10	-0.20	-0.5
0.86	1.53	-0.9
0.69	0.93	0.8
0.57	0.28	0.5
-0.89	0.29	-0.6
-0.74	0.16	-0.4

Table 10.4: Results straight line 2m odometry only

x error (m)	y error (m)	platform orientation (rad)
-0.39	0.12	-0.5
0.13	0.07	0.1
0.25	-0.08	0.2
0.06	0.07	-0.1
-0.57	0.13	-0.1
0.12	0.07	0.0
-0.16	0.09	0.0
0.13	0.11	0.1
-0.02	-0.01	0.0
-0.18	0.05	-0.1

Table 10.5: Results straight line 2m IMU + odometry

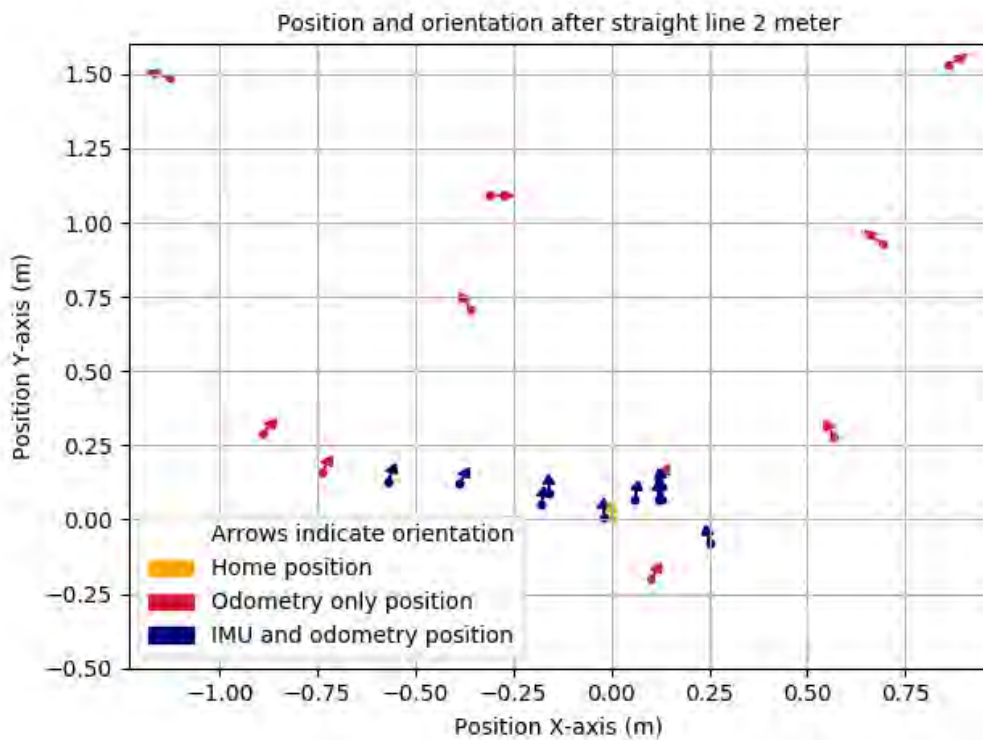


Figure 10.3: Visualization of Tables 10.4 and 10.5 for first test setup at length 2m

x error (m)	y error (m)	platform orientation (rad)
-1.13	4.12	-1.6
1.54	0.63	0.6
-1.17	0.70	-1.4
-0.76	0.66	-0.9
0.61	0.88	-0.3
-1.37	1.28	0.8
0.99	1.12	1.1
-0.32	0.92	-0.7
0.42	1.87	-2.5
0.32	0.43	0.4

Table 10.6: Results straight line 3m odometry only

x error (m)	y error (m)	platform orientation (rad)
-0.16	0.09	0.3
1.41	0.33	0.3
-0.60	0.13	-0.3
0.22	0.17	-0.2
0.46	0.18	-0.1
0.92	0.29	0.1
-0.33	0.08	-0.2
0.71	0.11	-0.1
-0.45	0.26	0.1
-0.86	0.34	0.4

Table 10.7: Results straight line 3m IMU + odometry

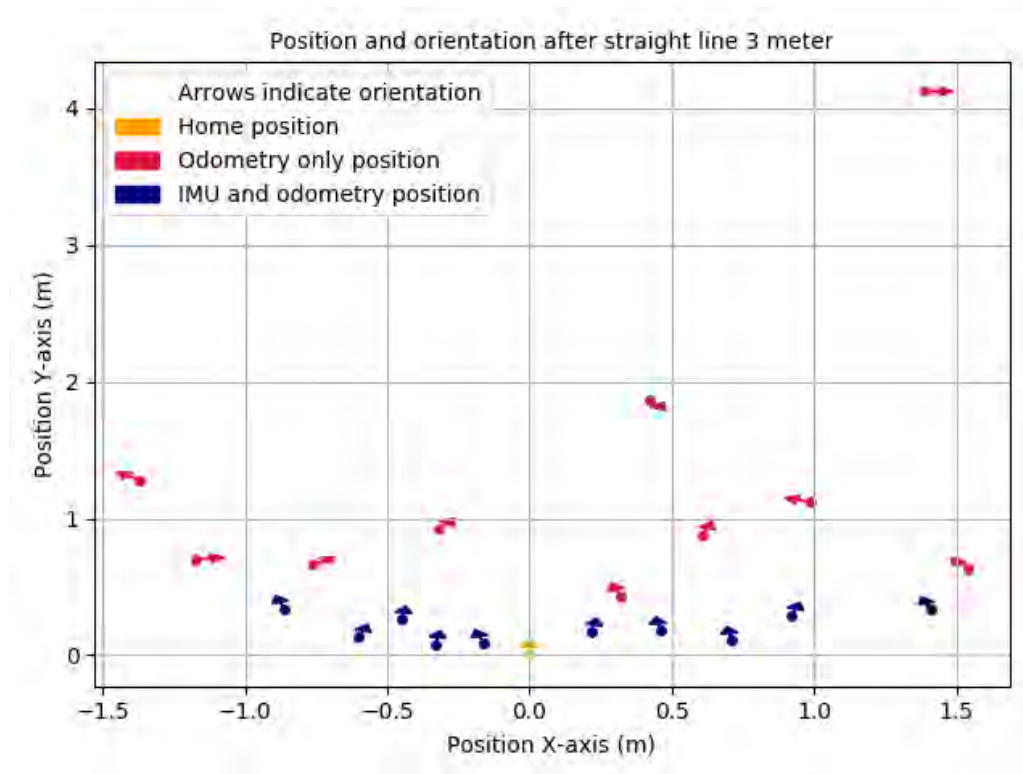


Figure 10.4: Visualization of Tables 10.6 and 10.7 for first test setup at length 3m



### 10.3.2 Turning

In this subsection the results for the test run with the 90 degree turn are given. As in the previous subsection the odometry only results are discussed first, followed by the IMU and odometry results before being compared to one another.

In Table 10.8 the results for the odometry only are shown for the shortest length. These results are depicted in red in Figure 10.5. As with the results from the straight line test the odometry only method is able to keep a sort of sense of direction. Although this sense of direction is worse then in the straight line test with less of the results pointing towards the point where the turn was made. What is less obvious but still seems to be a pattern is the precision errors following a line from the third quadrant to the first (negative x and y towards positive x and y). This could be caused by the rotation of the sphere being higher in one particular direction during the movement.

The results for the odometry only method for the 2m path lengths with a turn is shown in Table 10.10, being depicted in red in Figure 10.6. Once again the robot starts to lose its orientation when path lengths exceed 1 meter. Remarkable however is that in this case the final positions do not follow the line from the third quadrant to the first. This is seen in the two outliers in the second quadrant. It is possible that these two outliers are caused by turning back to far due to the sphere moving with the robot itself.

Lastly the odometry only results for the 3m path lengths with a turn. These results can be found in Table 10.12, being depicted in red in Figure 10.7. As in the shortest path lengths these results seem to follow the mentioned line from the third to the first quadrant. Furthermore the orientation is no longer correct.

To get to results of the the IMU and odometry combination. All three lengths can be discussed at once here since the all seem to follow the same pattern. These results can be found in Tables 10.9, 10.11 and 10.13. They are depicted in blue in Figures 10.5, 10.6 and 10.7. In all results the robot keeps a much better orientation when compared to the odometry only method. The precision is also a lot better, being more clustered. Some precision is of course lost when the traveled path increases, but still remains significantly better in all results.

x error (m)	y error (m)	platform orientation (rad)
-0.23	-0.23	0.5
-0.16	-0.32	-0.6
-0.58	-0.17	-0.5
0.47	0.29	-0.3
0.34	0.18	0.3
-0.45	-0.23	0.4
0.13	0.11	0.1
-0.12	0.14	-0.5
-0.18	-0.08	-0.4
-0.38	-0.27	-0.2

Table 10.8: Results two straight lines 1m with 90 degree turn in between odometry only

x error (m)	y error (m)	platform orientation (rad)
-0.26	0.13	0.0
-0.24	-0.01	-0.2
0.22	-0.07	0.0
0.03	-0.13	0.0
-0.18	-0.05	-0.1
0.16	0.12	0.2
-0.06	0.05	0.1
0.12	-0.08	0.0
0.16	0.16	0.0
0.23	0.15	0.2

Table 10.9: Results two straight lines 1m with 90 degree turn in between IMU + odometry

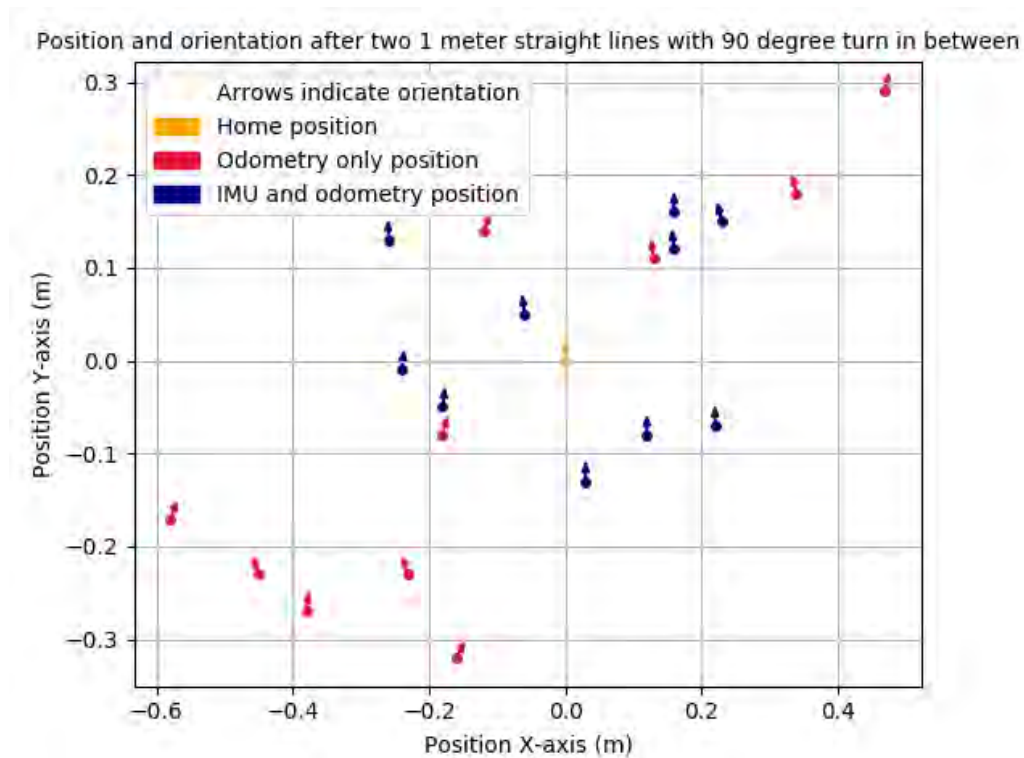


Figure 10.5: Visualization of Tables 10.8 and 10.9 for second test setup at two times length 1m with 90 degree turn in between

x error (m)	y error (m)	platform orientation (rad)
-1.38	0.34	-0.5
0.19	0.53	0.3
-0.73	-0.34	0.6
-1.06	1.67	-1.3
-0.94	1.34	-1.5
1.53	1.43	2.2
-0.43	-0.84	1.3
0.13	0.91	2.1
0.84	0.76	1.6
-0.26	0.31	0.6

Table 10.10: Results two straight lines 2m with 90 degree turn in between odometry only

x error (m)	y error (m)	platform orientation (rad)
0.63	0.33	0.3
0.73	0.57	0.4
0.54	0.08	0.3
-0.34	-0.04	-0.2
-0.43	0.22	0.3
0.38	0.16	0.1
-0.67	-0.19	-0.3
-0.36	0.16	0.1
-0.76	0.37	-0.2
0.16	-0.06	0.0

Table 10.11: Results two straight lines 2m with 90 degree turn in between IMU + odometry

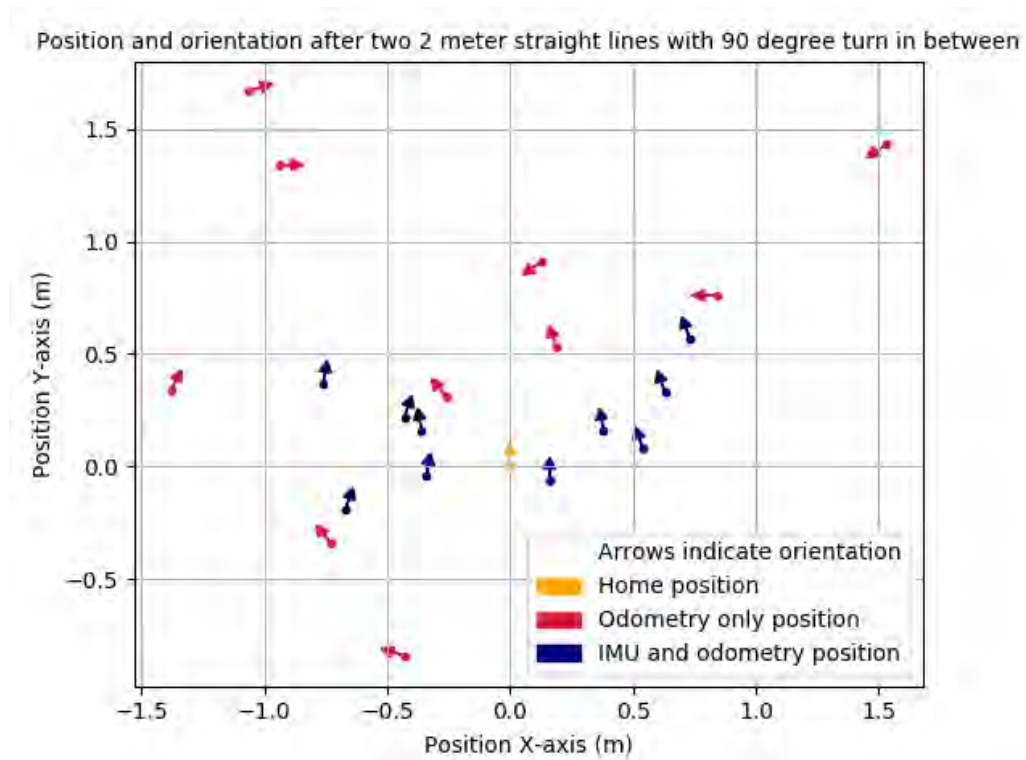


Figure 10.6: Visualization of Tables 10.10 and 10.11 for second test setup at two times length 2m with 90 degree turn in between

x error (m)	y error (m)	platform orientation (rad)
1.06	0.66	0.8
1.85	1.31	0.3
-1.16	0.08	-0.5
-1.26	-1.02	1.5
0.98	0.46	-1.3
1.37	0.76	-0.6
-1.67	-1.12	-1.2
-1.39	-0.47	-0.3
0.56	0.34	0.6
-0.86	-0.16	-1.0

Table 10.12: Results two straight lines 3m with 90 degree turn in between odometry only

x error (m)	y error (m)	platform orientation (rad)
-0.67	-0.34	-0.5
0.68	-0.54	0.3
-0.04	0.34	-0.2
-0.79	0.12	-0.3
0.35	0.10	0.1
0.02	-0.30	0.0
0.53	0.30	0.3
-1.12	0.21	-0.4
-0.30	0.34	-0.1
-0.12	0.09	0.0

Table 10.13: Results two straight lines 3m with 90 degree turn in between IMU + odometry

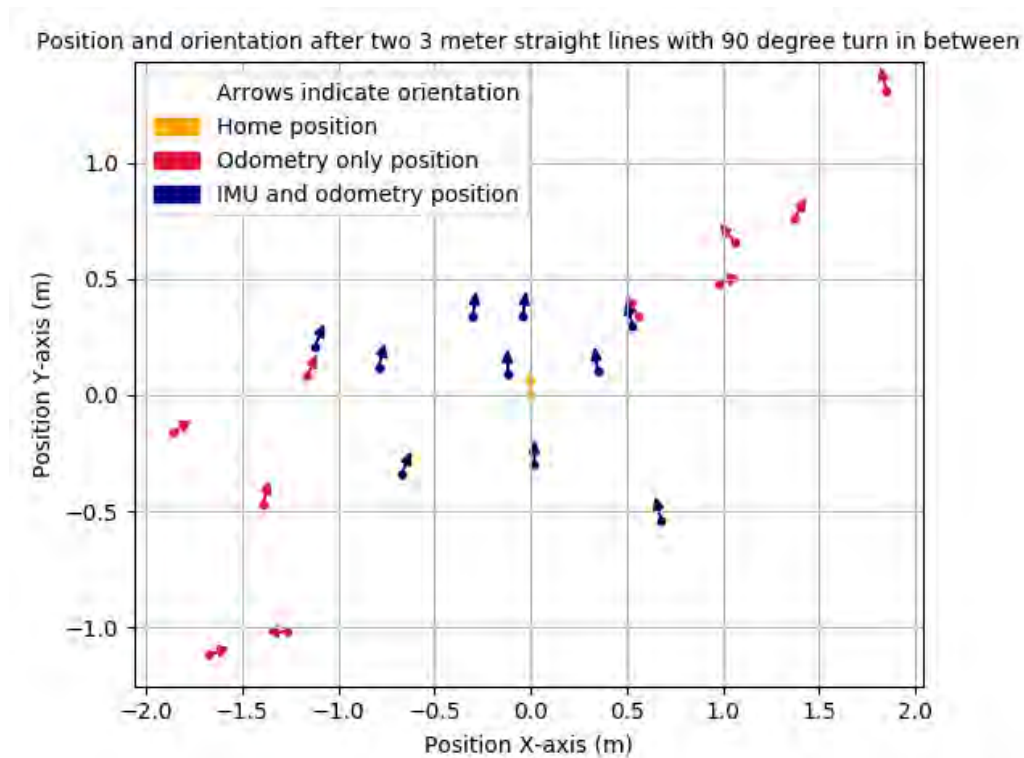


Figure 10.7: Visualization of Tables 10.12 and 10.13 for second test setup at two times length 3m with 90 degree turn in between

### 10.3.3 General remarks and four wheeled setup

The reason the odometry only method results have such a high variation in orientation is due to the platform no longer being able to sense the overall orientation of the sphere. The sphere itself easily rotates due to its single point of contact with the ground. Having the sphere itself rotate with the platform within it is probably also the reason the precision is lost in the cases of the odometry only method.

The IMU and odometry combination seems to deviate mostly in the X positions in the first test setup, with the Y positions coming up just short above the Y-axis. Which expanded to both the X and Y axis once the turn was introduced. The deviation is probably due to the drift which is found in the IMU itself as was mentioned in Table 9.2 . In normal cases this drift can be kept to a minimum by using the built in compass. This compass uses the earth's magnetic field as a global reference resulting in an absolute orientation value. However, in this case the presence of stepper motors renders this compass useless. These stepper motors produce strong magnetic fields in order to rotate the wheels which completely throw the compass off the trail. In order to improve this a Faraday cage could be built around the stepper motors to confine their magnetic field.

To get to the omission of the four wheeled setup and why no results for it are given. The reason for this exclusion is that this setup resulted in too much slippage of the wheels on the inside of the ball. This resulted to such a loss of precision and direction that even driving the robot towards the let go point was very difficult by hand. This is obvious since the odometry cannot tell when the wheels are slipping. The reason of the slippage can be compared to a four legged table, which can rock back and forth on a diagonal pair of legs. A three legged table however cannot, with all three legs being firmly on the ground at all time. This same principle applies to this robot. When the robot moves from one half of the sphere to another half the two halves move a bit compared to one another due to the small movement in their fastening. Due to this small movement the two halves no longer form a perfect sphere on the inside which leads to the robot rocking on a diagonal pair of legs. This leads to uncontrollable rotation when a third wheel still retains contact to one of the halves whilst its diagonal paired wheel hangs in the air. To solve this issue either a better fastening has to be made which does not allow for movement or suspension has to be added. The other option was to remove one of the wheels which results in contact with both halves at all times and thus reduces slippage to a minimum.

## Chapter 11

# Conclusions

In this chapter the conclusion based on the results are given. More future work can be proposed based on the conclusions given here, this is therefor discussed in chapter 13.

As is mentioned in subsection 10.3.3 of the previous chapter, getting a four wheeled setup working is impossible in the given configuration as seen in section 3.1 for a spherical robot. Due to the small movement between the two halves of the sphere the internal platform starts to rock back and forth lifting one wheel up from the inside of that sphere. This results in uncontrollable slippage, which is results to turning of the platform. This turning is not reported by the odometry due to its inability to detect this slippage. The only option for a four wheeled setup would be to add suspension to the mounts for the motors. However due to the increased complexity and therefor increased risk of mechanical failure a three wheeled setup is advised for future spherical robots with an internal omniwheel platform.

In contrast to more common robot which have direct wheel to ground contact the introduction of a sphere in between leads to orientation errors when moving. This cannot be controlled by the odometry alone since it reports no difference in orientation due to it being unaware of the absolute orientation of the robot as a whole. This loss of orientation awareness was apparent in the results given in section 10.3, even in motion straight forward and backward. Due to the slow rotation of this robot during turns only the orientation of the platform as a whole barely moved, not the sphere itself. However when rotating at higher speeds some rotation errors could occur due to more abrupt movements, which could result in the sphere itself also rotating or moving. The results thus have shown that during movement and rotation an IMU is necessary in order to improve the robots dead reckoning performance. With the addition of this IMU as the second level of the navigation architecture orientation awareness could be maintained for a longer period whilst the outer sphere rotated. However, the drift found in all IMU's was noticable in all of the results due to the usage of a lower cost IMU. This resulted in final positions which were nearly exactly at the distance traveled from the let go point, but rotated.

## Chapter 12

# Future navigation architecture

The navigation architecture as proposed with its two levels of odometry and IMU information can be expanded in multiple ways. A few more levels have already been mentioned which will be discussed in this chapter. These other levels have also been researched in varying length, thus the amount of detail differs per level.

### 12.1 Bluetooth positioning

While this part is out of the scope of this thesis for now, some work has been put into this to figure out how this could be implemented in a way that suits the needs of this robot.

To make sure a compensation is found for the drift in the IMU the need of an absolute reference system arises. To achieve this usually methods like the Global positioning system (GPS) are used. The downside to GPS is that in indoor environments most of the normal, consumer-grade, GPS receivers will not work due to the signal not being strong enough. This would mean it would have to be enhanced by capturing it and broadcasting it again with the use of extra antennas [95].

To increase the chances of getting a correct indoor position one could look to the method as described by rebroadcasting the GPS signal with antennas. On the other hand other systems have been created which could be implemented more easily by using exists WiFi signals and/or Bluetooth beacons [96]. The idea behind this method is that you can measure the received signal strength of the beacon or router, also called the Received Signal Strength Indicator (RSSI). Then with this RSSI value the distance can be calculated, knowing that the RSSI value will of course be lower once further away. Then the last thing that rests is to use these distances and the beacons known position to determine the robots position just like in the case of the GPS.

### 12.1.1 Retrieving RSSI

The easiest way to retrieve these values is with the use of Bluetooth Low Energy (BLE) beacons. With the introduction of the Bluetooth 4.0 standard the use of these beacons became much easier since the beacons no longer have to be paired with the users device. This means that it now works more like the normal GPS system in which the satellite is the only sender and does not listen to anything else. Once set up, the robot only has to listen to these beacons which are sending out their packets once every half to one second (on average, might be adjustable depending of beacon type). Then when a packet is received the bluetooth controller determines its signal strength and then gives back the RSSI value for further calculation.

The problem with retrieving an RSSI value is that in indoor environments it is rather noisy and thus needs to be filtered. To accompany for this noisy [6] proposes to use a weighted sliding window method to reduce the measured fluctuations in the RSSI values. This method works as follows for the array  $X$  with rssi readings  $x_i$  (up to a maximum of 10 readings):

---

**Algorithm 16** RSSI Smoothing [6]

---

```
1: procedure SMOOTHING( $X$ )
2:    $R = \{x_i \in X \mid (\bar{X} - 2 * \sqrt{Var(X)}) < x_i < (\bar{X} + 2 * \sqrt{Var(X)})\}$ 
3:    $rssi = \frac{\sum_{r_i \in R} r_i}{|R|}$ 
4: end procedure
```

---

In this way a lot of the noise is filtered out, by throwing away all of the outliers and taking the average of the more likely correct measurements. Keeping the array's size to a maximum of 10 does however ensures the reading remains updated frequent enough to produce accurate results.

### 12.1.2 Determine position

One of the problems in determining the position is that the correlation between RSSI and distance is not linear. The opposite even, whilst the RSSI values range from around  $-40\text{dB}$  when the beacons are  $1\text{m}$  away to  $-60\text{dB}$  at around  $4 - 5\text{m}$  to  $-100\text{dB}$  at  $30+$  meters. The next problem is that the default way of determining the distance as described in [6] assumes it works on a logarithmic scale that is continuous. However they are using a limited equation that is not specifically optimized for our setup. The Friis transmission equation [97] is a more advanced method. This equation takes into account what the gains of the sender and receiver antenna, the broadcasting strength and the exact frequency at which the signal is broadcast. However both of these methods do not incorporate a way to prevent wrong distance calculations in case an object is between the sender and the receiver. Thus resulting in a faulty outcome that is higher then it actually is, since the signal strength is lower at the receiver. There are ways to overcome this as described in [98] where they take the multipath propagation (the signals that bounce off of walls in the room and reach the receiver) into account.



The downside to all of these is that you would need to know exactly what the values are for each of the parameters. An easier way to correctly optimize the way the position is calculated would be to use neural networks, which have a much better precision [99]. With these the various levels of RSSI could be correctly linked to a position after enough training has been done. The problem with neural networks however is that they normally require a fixed size input and the signals from all of the beacons might not be available everywhere at all times. For this issue a solution has been devised by [18] where they proposed a Modular Multi Layer Perceptron (MMLP) architecture as seen in Figure 12.1

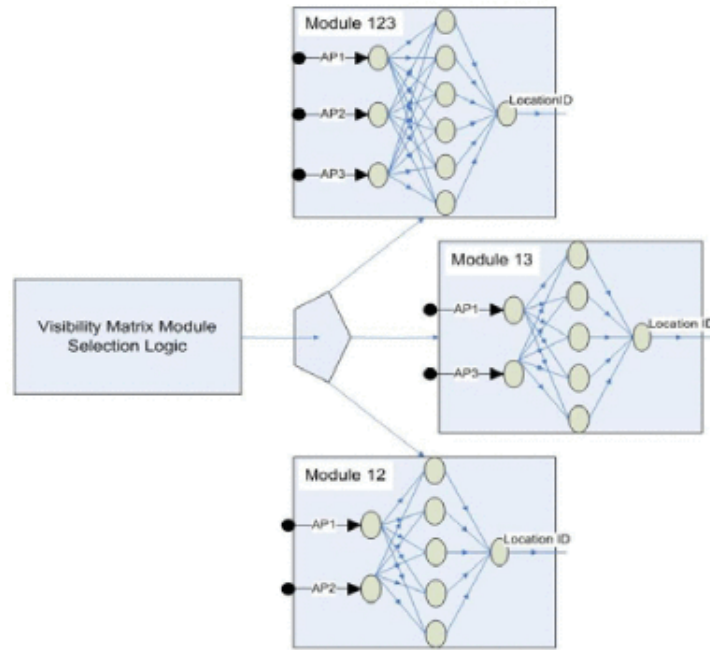


Figure 12.1: Modular Multi Layer Perceptron [18]. Left is the selector which selects which of the trained networks is used. Right are the networks with the input layers and the Acces Points (AP 1-3), the hidden layer and the Location ID which comes out of the output layer

This MMLP enables you to calculate the position even in the case of a loss of signal from one or perhaps multiple beacons. The selection logic will however always choose the network with the most available inputs to make sure all of the available data is used. In the case of our BB-8 robot the networks themselves have been adjusted according to [99] to produce actual  $x$  and  $y$  values in meters instead of location ID's, with which they labeled their positions. Besides this the input and hidden layer have more nodes in our case of course to incorporate for the extra beacons we have.

Finally after getting the  $x$  and  $y$  values out of the MMLP the retrieved  $x$  and  $y$  values are fed into a smoothing algorithm. This smoothing algorithm was initially proposed by [7] to smooth the RSSI values, but in this case it is also useful to smooth out jumps in the position because of the multiple networks that are used. This algorithm can be seen in Algorithm 17.

---

**Algorithm 17** Position smoothing [7]

---

```
1: procedure SMOOTHING( $\alpha, \beta, T_s, x_{prev(i)}$ )
2:    $x_{est(i)} = x_{pred(i)} + \alpha (x_{prev(i)} - x_{pred(i)})$ 
3:    $v_{est(i)} = v_{pred(i)} + \frac{\beta}{T_s} (x_{prev(i)} - x_{pred(i)})$ 
4:    $x_{pred(i+1)} = x_{est(i)} + v_{est(i)} * T_s$ 
5:    $v_{pred(i+1)} = v_{est(i)}$ 
6: end procedure
```

---

In these algorithm  $T_s$  is the timestep between measurement  $i$  and  $i - 1$ ,  $\alpha$  and  $\beta$  are the gain constants, which are usually fixed for a specific situation.  $x_{est(i)}$  is the smoothed x value,  $x_{pred(i)}$  is the predicted value and  $x_{prev(i)}$  is the value from the neural networks.  $v_{est(i)}$  and  $v_{pred(i)}$  are the estimated and predicted speeds with which the  $x_{est(i)}$  is updated and with which it thus moves closer to the calculated value  $x_{prev(i)}$ .

## 12.2 Radar collision detection

The indoor positioning system has now been defined with the aid of bluetooth for quick global positioning in a building. The next step would be to start with sensing the environment around the robot in order to start with collision avoidance. In order to do so the easiest way would be to start using radar sensors.

The advantage over the proposed BPS is that in the field of radars various prepackaged sensors are available. This ensures that signal processing is less of a concern and this section can go more into detail about how to use the gained information in conjunction with the previously proposed and/or implemented levels. Some of these radar sensors also have the capability of providing the user with speed and direction information. This direction information however is usually restricted to inbound or outbound.

As mentioned in Section 5.3, where a recommendation was made for the used mapping techniques, the most obvious approach for mapping would be to use occupancy grid. Due to the radars ability to tell at what distance an objects is not much more information can be gained except an object being at a certain position. Furthermore as mentioned in section 4.5 the best localization technique to use in combination with larger maps would be the Particle Filter. A recent study has combined both of these recommendations where they implemented a Real-Time Radar SLAM method in tracking a driving car [19]. Their proposed SLAM method can be seen in Figure 12.2.

If the method as proposed in [19] is used for the BB8 robot there would be an extension to their method as seen in Figure 12.2. In order to improve the Odometer data an EKF would be placed in between the Sensor buffering and Update particle poses as mentioned in Section 4.5. This would probably increase the effectiveness of the method since the odometer data as used in their study came straight from a very expensive car with various very precise systems. Their odometry precision can probably not be achieved with the unfiltered odometry data due to the instability as caused by the spherical shape.

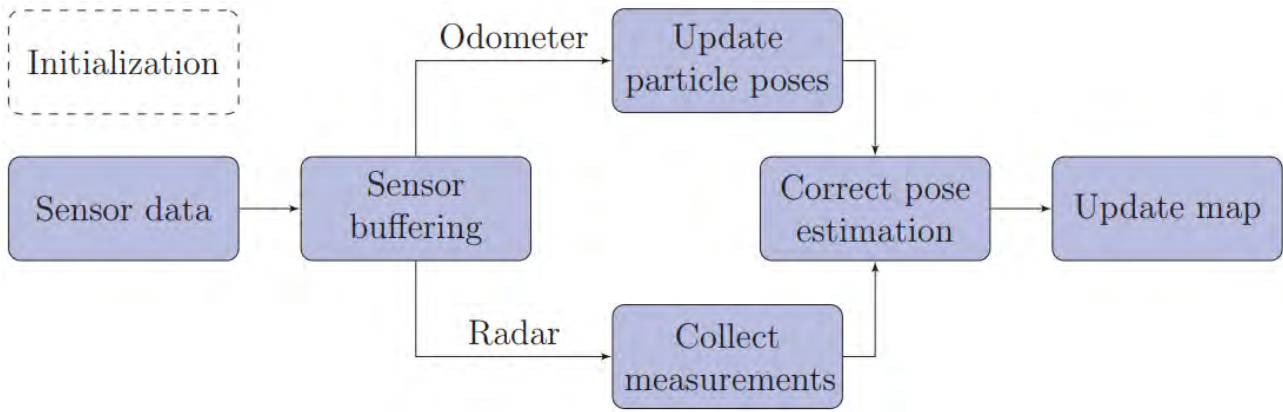


Figure 12.2: Radar based SLAM implementation [19]

### 12.3 RGB-D Camera

In order for the robot to eventually give guided tours to humans it has to be able to recognize those humans. The easiest way to do so is with the use of cameras, with the addition of a depth sensor with which point clouds can be made as well. Some work into using an RGBD camera has been done already and as such will be discussed here and recommendations for its implementation will be made. Note that this was early work and is far from optimal, much can be improved in its implementation. This is also seen later on in the images which do not yet correctly represent the surroundings.

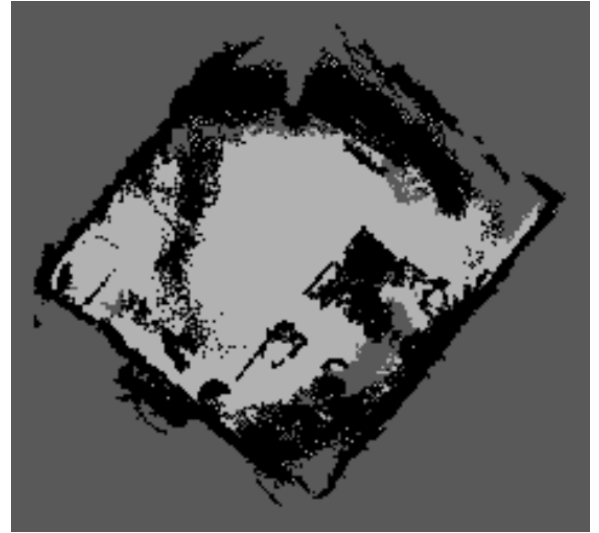
The usage of RGB-D camera's makes it a lot easier to distinguish individual places. With these camera's objects like paintings, poster, etc can quickly be identified and stored as a landmark with their unique appearance. This in turn leads to making the localization problem easier to solve. In the case of the robot for which this navigation architecture is proposed it also has a few other benefits. One of such benefits is the ability to detect humans which it is giving a guided tour to. It could then be able to directly look at those persons whilst talking creating a more personal experience.

During earlier work on the robot some work has been put into realizing a working setup with an RGB-D camera. This camera was eventually removed in order to focus on the basics of the movement before moving on to more complex sensors such as cameras. The camera in question was a XBOX Kinect V2 capable of delivering a full HD output at 30 FPS. The live RGB data was then combined with a lower resolution infrared depth camera to create a point cloud. The advantage of using these point clouds is that they can be converted to occupancy grids [100]. These occupancy grids can then be used to quickly calculate a path since it simply has less dimensions for which to calculate it. To see the point cloud and the occupancy grid which were used see Figure 12.3. In that figure both of the images have a top down view.

A good solution to using the RGB-D camera as a sensor for SLAM has been given in [21]. This method proposes an advanced set of techniques used to enable autonomous driving. The entire method can be found in Figure 12.4. The advantage of this method is that it enables the use of Long Term Memory (LTM). This is very useful in the case of this robot since this enable the robot to quickly find its position based on landmarks



(a) Point cloud



(b) Occupancy grid

Figure 12.3: Occupancy grid visualization of obtained point cloud for lab [20]

stored from previous guided tours, this is as proposed in section 5.3. These landmarks are imported into the Working Memory (WM) which is used to actually detect the robot position. With such a known position loop closures can then also be applied in order to improve the localization. If these loop closures are found the robot could then localize itself more precisely using the recommended Particle Localization technique in order to pinpoint its position within either the point cloud or occupancy grid as proposed in section 4.5. After the localization the Topological and Metric Path Planners as seen in Figure 12.4 simply use Dijkstra's algorithm to compute a path. However using the occupancy grid to more precisely calculate the robots position it might be better to use Approximate Cell Decomposition as proposed in section 7.3.

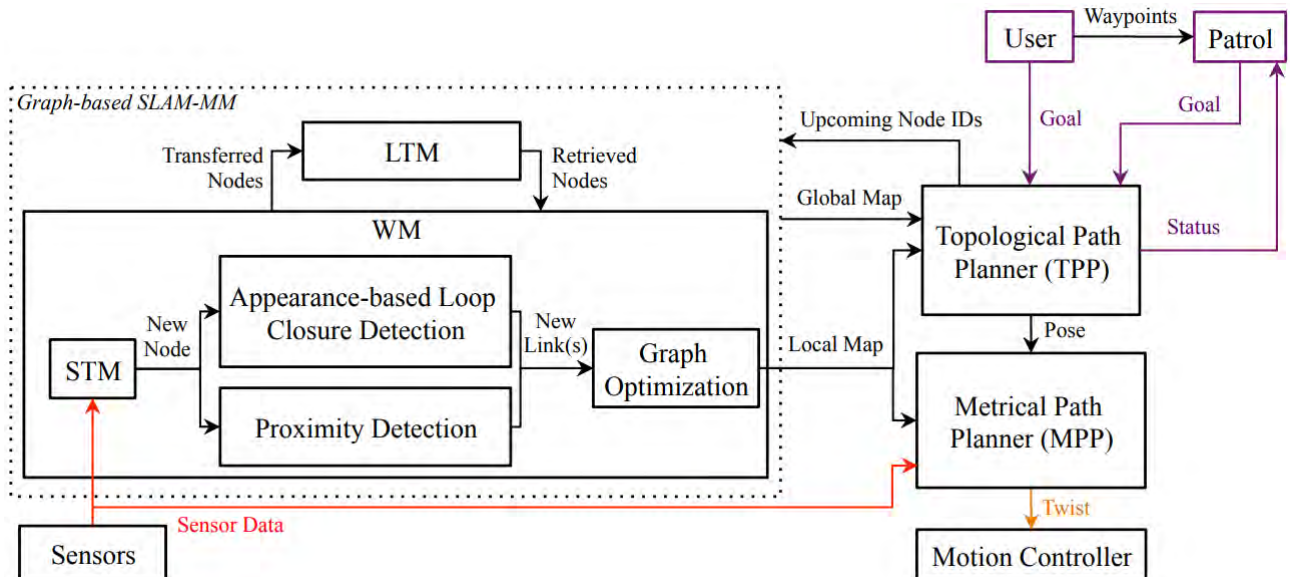


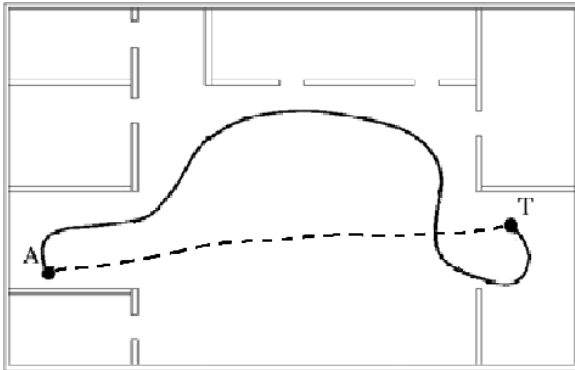
Figure 12.4: RGB-D base SLAM implementation [21]

## 12.4 Future evaluation

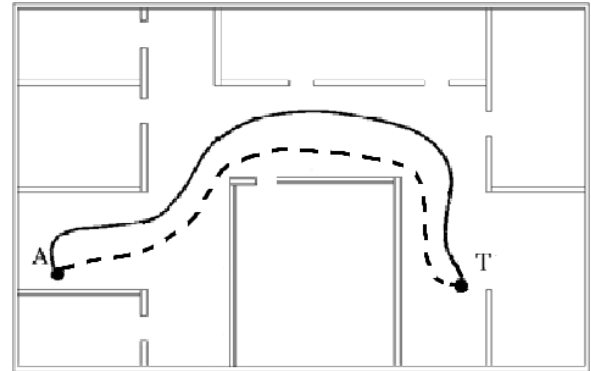
For now the robot is quite limited in capabilities due to its highly unstable nature and no external references with the aid of sensors. These future navigation architecture levels should improve this and be capable of more difficult tasks. Thus more research can be done towards its effectiveness when these levels are added.

Future test setups could include the following:

1. Incorporating more turns and turns whilst driving.
2. Moving around obstacles.



(a) Example of future test setup 1, moving with turns



(b) Example of future test setup 2, moving around obstacles

Figure 12.5: Future test setup examples.

More light could also be shed on the aspect of speed during movement and rotation. Whereas in the test setups used in the results as given in section 10.3 were performed at slow speed, the influence of speed was not researched. Furthermore it might be interesting to see how the errors in rotation exactly occur. This can be achieved by either following the path of the robot with a method to determine the true ground path. Another option might be to install a camera on the inside of the sphere. This camera can be used in order to see what happens when the internal platform drives over the joint connecting the two halves of the sphere. Another interesting this that might be seen with this camera is what exactly happens during driving and rotation as this is unknown at this moment.

## Chapter 13

# Discussion

Due to the limited resources and budget available to build the robot, much could be improved in terms of precision. The fabrication of the entire frame and motor mounts together with the assembly of all parts were done by one or more students. This leads to an imperfect internal drive platform which is a part of the errors found in the results. Also due to the limitation in budget cheaper sensors and motors were used. These cheaper parts have a lower resolution and higher noise which result in worse performance of the navigation architecture as a whole. Therefore with a higher budget and higher quality parts, frame, etc. a better result could be found than given here.

Furthermore due to the manual guidance of the robot to the let go point the path towards that let go point is suboptimal. But as mentioned due to the spherical shape an orientation error can arise easily. When guiding the robot by hand to its let go point dead ahead it will also move slightly to the sides. This sideways movement will also lead to orientation errors and therefore reduce the outcome. However, in the case of a navigation architecture capable of perfect position and orientation awareness this should not be a problem.

All of the test runs were also set up by hand. This means that the robot was not exactly at position 0,0 when it comes down to centimeters. Knowing that the length of each run is quite small and the size of the robot is big compared to this run, some of the precision errors can be traced back to the setup. As proposed in chapter 12 better test run setup can be done in order to more precisely measure the precision and orientation errors.

# Appendix A

## BB-8 technical details

In this appendix more details of the robot are given, elaborating on the the actual components and their characteristics. In the first section the shell is explained in more detail, followed by the motors and servo's. Finally the computing hardware and electrical components are shown together with a complete electrical diagram how everything is connected.

### A.1 Shell

Each of the shells' hemispheres is made up of 20mm thick PU foam which is milled with a CNC machine. The milling was done up to 8cm of height per slice of the hemisphere due to the limitations of the CNC machine, later on these are glued together to form a complete hemisphere. Around this foam two layers of 225 g/m<sup>2</sup> fiberglass and epoxy resin are placed, enveloping both hemispheres completely. The closing mechanism is provided by two 2cm long rabbets that interlock. These are first milled into the PU foam to make sure the fiberglass is molded to their correct shape. The rabbets are then covered with hook-and-loop fasteners all around the hemisphere. Some parts are intentionally left open to prevent the connection from becoming too strong to open by hand.

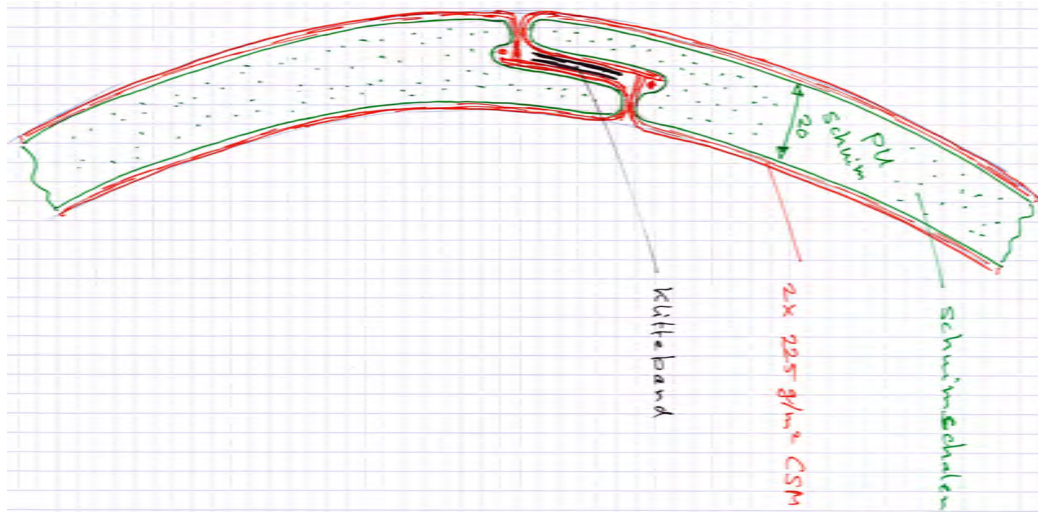


Figure A.1: BB-8 shell mock-up sketch

In Figure A.1 a mock-up sketch of the shell is given which was used during its design and construction. This sketch is zoomed in on the edge between the two hemispheres, with the rest continuing to the left and right. The dotted green part in the center represents the 20mm thick PU foam, the red are the layers of fiberglass and epoxy and the black part is the hook-and-loop fasteners, which are placed on the two 2 cm long rabbets.

## A.2 Motors and servo's

The motors are Trinamic QSH6018-65-28-210 stepper motors, each of which is controlled by its own driver. The stepper motors are capable of delivering 2.1Nm of holding torque, run at 36V and use up to 2.8A of power, making 200 steps per full rotation. The drivers that are used are Trinamic TMC2130 stepper motor drivers, built specifically for the used stepper motors. They have a voltage range of 9 to 51V and can, with proper heat sinks, deliver up to 4A continuously. Communication with these drivers is possible via serial USB. The USB connection is made over USB 1.1 (FullSpeed 12Mbit/s) which is more than enough for the simple data that is sent back and forth. Each motor has its own USB cable to control them directly without interference of data being sent to other devices that are also connected like with I2C or SPI.

For the head mount two types of servo's were used, the more powerful being the Hitec HS-805MG's with the smaller for just the rotation being the Hitec HS-645MG. These servo's are controlled by PWM signals originating from a simple arduino.



### A.3 Hardware and electrical components

The entire robot will eventually be controlled by a powerful Jetson TX1 development board, which has a quad-core ARM Cortex-A57 together with a 256 cuda-core Maxwell GPU. The CPU and GPU together run under 10W of power at full load, whilst still providing over 1TFLOPS of performance. To see all of the other available connections for the Jetson TX1 view Figure A.2. This board is located in the head for the fact that it has USB 3.0 built that probably will be required by camera's for its increased speed, since these camera's can only be placed in the head (the body is completely closed). It also operates in cooperation with a Raspberry Pi 3 Model B located in the body of the robot which are connected to each other via an ad-hoc Wi-Fi network. The Pi then communicates with the four motor controllers via serial USB and the arduino for the head mount via I2C [101]. This Pi is more than capable of handling simple instructions and sending commands to the motor due to its quad-core 1.2GHz processor.

Furthermore there is a 720 Wh 4 cell LiFePo<sub>4</sub> battery (3.2V, 60Ah per cell) inside the body for the motors and other internal hardware. These four separate cells are hooked up to a Simple Battery Management (SMB) board. This SBM Board combines the four cells to a 12.8V circuit capable of delivering 60A of current. With the use of this SBM Board the cells are charged and it also continuously balances the cells to increase their lifespan. In the head the same types of batteries are used however only with a capacity of 120 Wh, though enough to keep the Jetson and some sensors operating for a prolonged amount of time. To convert some power as the servo's run at 6V XL4015 step down converters are used, they are set parallel to make sure over current does not occur, since they cannot provide enough power when set up with one step down converter per servo. When they are put in series they can provide enough current since the smaller servo requires a lot less. To give the stepper motors some more speed, since they are able to attain higher rot/s when given a higher voltage, LTC1871 step up converters are used to boost the voltage up to 36V. A complete diagram of all the connections in the sphere can be found in Figure A.3.

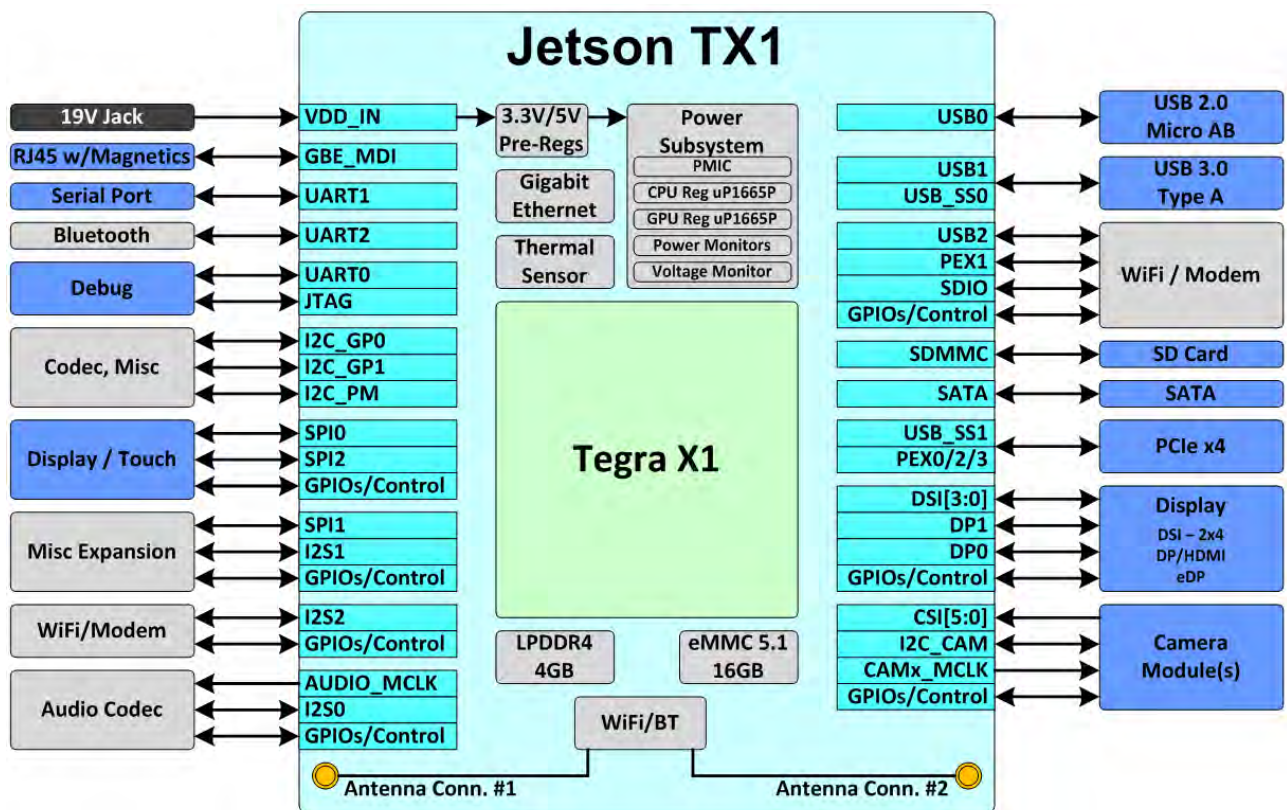


Figure A.2: Jetson TX1 Connections [22]

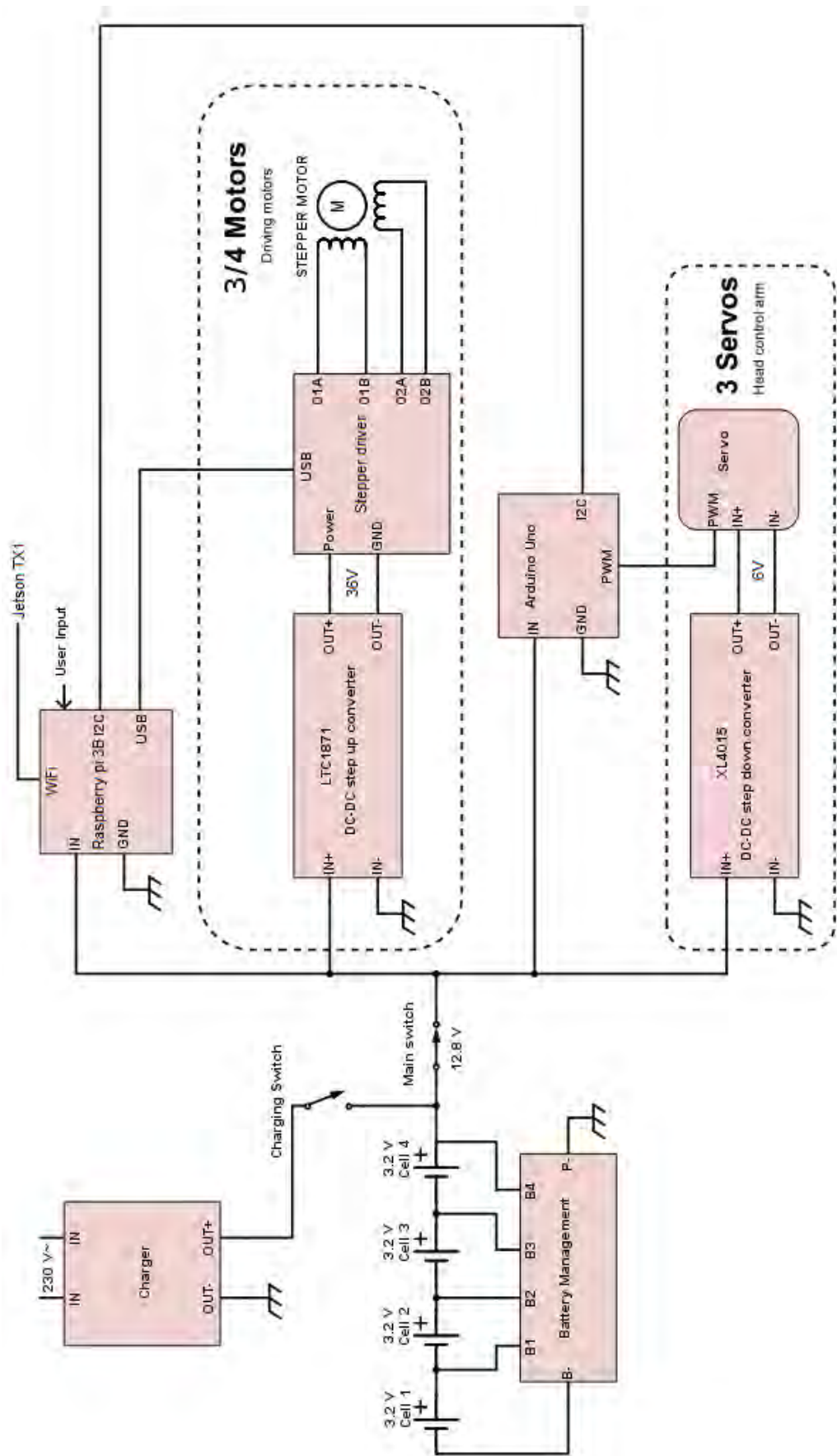


Figure A.3: BB-8 electrical diagram

## Appendix B

# Implementation into ROS

This chapter of the appendix will show the workings of the packages that work together with the aid of the ROS framework in order to make the robot move. To correctly understand everything that is mentioned a good understanding of the ROS framework is essential. First of all each of the packages will be reviewed. This will be followed by showing how these packages work together for each of the two proposed navigation architecture levels.

### B.1 Packages

In order to show how the robot works the usage of each of the packages is explained. This will encompass all of the packages used for the navigation architecture, including packages readily available from the community. In Table B.1 all of the packages are mentioned together with their usage. Each of these packages works in a separate process and communicates with each other through their respective topics via the ROS framework. After the quick overview of Table B.1 a more elaborate list is given in which all of the topics which it listens or broadcasts to are explained.

Package name	Use
<code>phidgets_imu_node</code>	Converts the IMU data from the installed Phidgets IMU drivers to ROS formats, does not filter or alter any data
<code>imu_filter_node</code>	Filters the IMU data to combine it to yaw, pitch and roll angles ready for usage in other packages
<code>ekf_localization</code>	Applies a EKF filter to odometry and IMU data in order to correctly fuse them and remove part of the noise
<code>bb8_control_node</code>	Takes input from the <code>/dev/js/input</code> file (the PS3 controller) and converts this to velocity commands and starts the homing sequence
<code>bb8_homing_node</code>	Takes odometry input which portrays the current position and tries to back-track it with velocity commands in order to end up at the home position
<code>bb8_movement_node</code>	Takes velocity commands and controls the wheel speeds reports the odometry back after movement

Table B.1: List of packages used in combination with ROS

- **Phidgets IMU node.** This package converts the data from the IMU to a format that can be transmitted over the local network through the ROS framework. The used Phidgets IMU has an included driver which can be installed on any device, but is therefore out of the box not compatible with ROS. This driver is used in this package in order to obtain the raw IMU data. The raw data is then transmitted via ROS to other packages via the `/imu/data_raw` topic.
- **IMU filter node.** The raw data from the IMU cannot be readily used since it does not include a heading. In order to obtain this heading the raw data is filtered with the Madgwick filter as mentioned in 9.2.1. This filter combines the gyroscope and acceleration data in order to determine roll pitch and yaw. As input this topic uses the `/imu/data_raw` topic from the `phidgets_imu_node`. When the data from this input topic has been converted the data is then broadcast on the `/imu/data` topic.
- **EKF Localization.** In the chapter 4 various localization techniques have been discussed. One of these is an EKF, which this node implements. It takes odometry data from the `/bb_8/odom` topic and IMU data from the `/imu/data` and fuses them together. By design it also filters out some noise that is introduced by the IMU sensor itself. The fused odometry data is then broadcast in the `/odometry/filtered` topic in the same structure as is the case on the `/bb_8/odom` topic.
- **BB8 control node.** This package connects the used PS3 controller to the ROS framework. By reading from the `/dev/js/input` file on a linux machine to which the PS3 controller is automatically connected after a bluetooth connection has been established. It then partially decodes the commands as sent over bluetooth and pre-processed by the Linux Joystick drivers. The values obtained afterwards are then converted to a ROS compliant structure and sent via the `/bb_8/cmd_vel` topic as velocity commands to move the robot. Once the PS button on the PS3 controller is pushed it calls the `/bb_8/homing` service once to hand over control to the homing node.
- **BB8 homing node.** The goal of the proposed navigation architecture is to move back to its home position. In order to complete that task this package takes odometry information as input from either the `/bb_8/odom` or `/odometry/filtered` topics. Depending on which navigation architecture level is being tested one of the two topics is chosen at the start. The odometry information is then stored in order to be used later when the homing procedure is started. This homing procedure is advertised as a service as `/bb_8/homing`. Once this service is called control of the velocity commands is handed over to this package which then starts to move the robot back to its home position. Only after calling that service for a second time during the homing procedure or after the homing procedure is finished the control is handed back to the control node to have the robot move through controller commands. During the homing procedure velocity commands are sent over the `/bb_8/cmd_vel` topic.
- **BB8 movement node.** This package actually controls the physical robot. It is the link between all of the logic implemented through the ROS framework and the physical wheels. This package listens to the velocity commands from the `/bb_8/cmd_vel` topic. These velocity commands are then converted to speeds used by the stepper motor drivers. These stepper motor drivers are attached to this package over an USB connection. Every cycle the stepper motor drivers are asked for the current step position in order to calculate the difference in steps. This step difference is then converted to distance and lastly the odometry for the entire robot is calculated from these values. The odometry is then broadcast on

the `/bb_8/odom` topic.

## B.2 ROS computation graphs

In the last section each of the built and used packages were explained. These packages listen and broadcast to a lot of the same topics in order to make them all work together in order for the navigation architecture to achieve its task. In the first level of the navigation architecture only the BB8 control, BB8 homing and BB8 movement nodes are used. Their connections are displayed in Figure B.1. This figure does not however display the services that can be called, such as the `/bb_8/homing` service, since these are called only once. The topics displayed there however continuously transfer data in the direction of the arrows (outgoing for broadcasting and incoming for listening to that topic).

The second level of the navigation architecture introduces the IMU and therefor the packages needed to use it. The added packages are the Phidget IMU, IMU filter and EKF localization nodes. The homing service is once again not displayed. The ROS computation graph for the inclusion of the IMU can be found in Figure B.2.

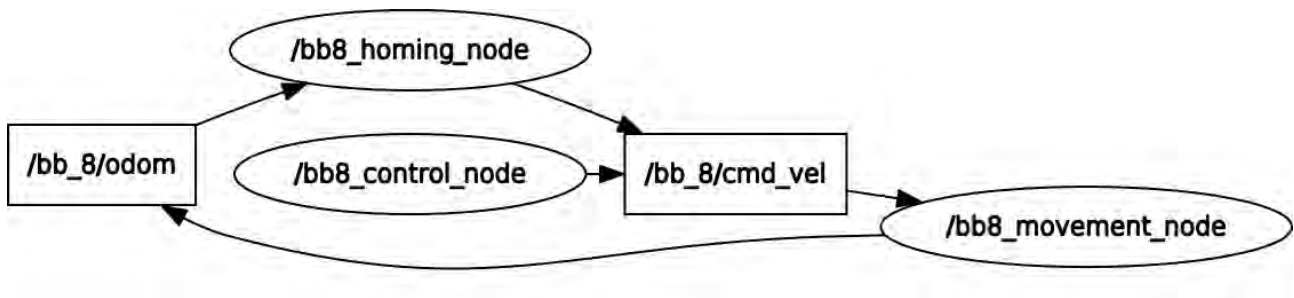


Figure B.1: ROS computation graph for the first navigation architecture level

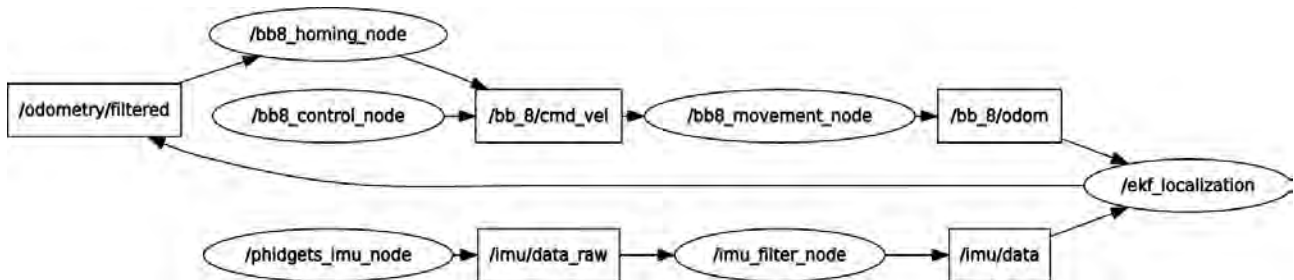


Figure B.2: ROS computation graph for the first navigation architecture level

# Bibliography

- [1] R. G. Boboc, M. Horațiu, and D. Talabă, "An educational humanoid laboratory tour guide robot," *Procedia - Social and Behavioral Sciences*, vol. 141, pp. 424 – 430, 2014.
- [2] W. Abdel-Hamid, "Accuracy enhancement of integrated mems-imu/gps systems for land vehicular navigation applications," Master's thesis, University of Calgary, 2005.
- [3] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Springer Publishing Company, Incorporated, 2nd ed., 2016.
- [4] Y. Maddahi, A. Maddahi, and N. Sepehri, "Calibration of omnidirectional wheeled mobile robots: method and experiments," *Robotica*, vol. 31, no. 6, pp. 969–980, 2013.
- [5] D. Lee, S. Lee, S. Park, and S. Ko, "Test and error parameter estimation for mems — based low cost imu calibration," *International Journal of Precision Engineering and Manufacturing*, vol. 12, pp. 597–603, Aug 2011.
- [6] H. Zou, X. Lu, H. Jiang, and L. Xie, "A fast and precise indoor localization algorithm based on an online sequential extreme learning machine," *Sensors*, vol. 15, no. 1, pp. 1804–1824, 2015.
- [7] E.-E.-L. Lau and W.-Y. Chung, "Enhanced rssi-based real-time user location tracking system for indoor and outdoor environments," in *Proceedings of the 2007 International Conference on Convergence Information Technology, ICCIT '07*, (Washington, DC, USA), pp. 1213–1218, IEEE Computer Society, 2007.
- [8] W.-H. Chen, C.-P. Chen, J.-S. Tsai, J. Yang, and P.-C. Lin, "Design and implementation of a ball-driven omnidirectional spherical robot," *Mechanism and Machine Theory*, vol. 68, pp. 35 – 48, 2013.
- [9] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.
- [10] D. Joubert, "Adaptive occupancy grid mapping with measurement and pose uncertainty," Master's thesis, Stellenbosch University, 12 2012.
- [11] B.-W. Kuo, H.-H. Chang, Y.-C. Chen, and S.-Y. Huang, "A light-and-fast slam algorithm for robots in indoor environments using line segment map," *Journal of Robotics*, vol. 2011, pp. 257852:1–257852:12, 2011.



- [12] M. Labbe and F. Michaud, "Online Global Loop Closure Detection for Large-Scale Multi-Session Graph-Based SLAM," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2661–2666, Sept 2014.
- [13] W. Burgard, C. Stachniss, M. Bennewitz, and K. Kai Arras, "Lecture notes in introduction to mobile robotics," July 2011.
- [14] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.
- [15] M. Klingensmith, "Overview of motion planning," Sep 2013.
- [16] V. Mazzari, "Ros robot operating system," Mar 2016.
- [17] J. Javier Moreno, E. Clotet, R. Lupiaez, M. Tresanchez, D. Martinez, T. Pallej, J. Casanovas, and J. Palacn, "Design, implementation and validation of the three-wheel holonomic motion system of the assistant personal robot (apr)," *Sensors*, vol. 16, p. 1658, 10 2016.
- [18] U. Ahmad, A. Gavrilov, U. Nasir, M. Iqbal, S. J. Cho, and S. Lee, "In-building localization using neural networks," in *2006 IEEE International Conference on Engineering of Intelligent Systems*, pp. 1–6, 2006.
- [19] M. Schoen, M. Horn, M. Hahn, and J. Dickmann, "Real-time radar slam," pp. 1–10, March 2017.
- [20] K. Benslimane, "De bb-8 rondleidingsrobot," bachelor's thesis, The Hague University of Applied Science, 2017.
- [21] M. Labbé and F. Michaud, "Long-term online multi-session graph-based splam with memory management," *Autonomous Robots*, vol. 42, pp. 1133–1150, Aug 2018.
- [22] NVidia, "Jetson tx1 block diagram," Nov 2015.
- [23] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *International Conference on Robotics and Automation*, 2010.
- [24] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "The interactive museum tour-guide robot," in *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, AAAI '98/IAAI '98*, (Menlo Park, CA, USA), pp. 11–18, American Association for Artificial Intelligence, 1998.
- [25] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz, "Minerva: a second-generation museum tour-guide robot," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 3, pp. 1999–2005 vol.3, 1999.
- [26] V. Alvarez-Santos, A. Canedo-Rodriguez, R. Iglesias, X. Pardo, C. Regueiro, and M. Fernandez-Delgado, "Route learning and reproduction in a tour-guide robot," *Robotics and Autonomous Systems*, vol. 63, Part 2, pp. 206 – 213, 2015. Cognition-oriented Advanced Robotic Systems.



- [27] A. D. Diallo, S. Gobee, and v. Durairajah, "Autonomous tour guide robot using embedded system control," *Procedia Computer Science*, vol. 76, pp. 126 – 133, 2015.
- [28] Y. Sun, M. Liu, and M. Q.-H. Meng, "Improving rgb-d slam in dynamic environments: A motion removal approach," *Robotics and Autonomous Systems*, vol. 89, pp. 110 – 122, 2017.
- [29] A. Banino, C. Barry, B. Uria, C. Blundell, T. Lillicrap, P. Mirowski, A. Pritzel, M. J. Chadwick, T. Degris, J. Modayil, G. Wayne, H. Soyer, F. Viola, B. Zhang, R. Goroshin, N. Rabinowitz, R. Pascanu, C. Beattie, S. Petersen, A. Sadik, S. Gaffney, H. King, K. Kavukcuoglu, D. Hassabis, R. Hadsell, and D. Kumaran, "Vector-based navigation using grid-like representations in artificial agents," *Nature*, vol. 557, pp. 429–433, 5 2018.
- [30] A. Halme, T. Schonberg, and Y. Wang, "Motion control of a spherical mobile robot," in *Advanced Motion Control, 1996. AMC '96-MIE. Proceedings., 1996 4th International Workshop on*, vol. 1, pp. 259–264 vol.1, Mar 1996.
- [31] I. Horswill, "Polly: A vision-based artificial agent," in *AAAI*, 1993.
- [32] I. R. Nourbakhsh, J. Bobenage, S. Grange, R. Lutz, R. Meyer, and A. Soto, "An affective mobile robot educator with a full-time job," *Artificial Intelligence*, vol. 114, no. 1, pp. 95 – 124, 1999.
- [33] T. Willeke, C. Kunz, and I. R. Nourbakhsh, "The history of the mobot museum robot series: An evolutionary study," in *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference*, pp. 514–518, AAAI Press, 2001.
- [34] R. D. Schraft, B. Graf, A. Traub, and D. John, "A mobile robot platform for assistance and entertainment," *Industrial Robot: An International Journal*, vol. 28, no. 1, pp. 29–35, 2001.
- [35] R. Bischoff and V. Graefe, "HERMES - an intelligent humanoid robot designed and tested for dependability," in *Experimental Robotics VIII [ISER 2002, Sant'Angelo d'Ischia, Italy, 8-11 July 2002]*, pp. 64–74, 2002.
- [36] G. Kim, W. Chung, K.-R. Kim, M. Kim, S. Han, and R. H. Shinn, "The autonomous tour-guide robot jinny," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 4, pp. 3450–3455 vol.4, Sept 2004.
- [37] M. Shiomi, Y. Kanda, H. Ishiguro, and N. Hagita, "Interactive humanoid robots for a science museum," in *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-robot Interaction, HRI '06*, (New York, NY, USA), pp. 305–312, ACM, 2006.
- [38] K. N. Museum, F. F. Limited, and F. L. Ltd., "Service robot enon from fujitsu acts as visitors guide at museum of best-selling mystery novelist," Sept. 2007.
- [39] D. Rodriguez-Losada, G. Matia, R. Galan, M. Hernando, J. M. Montero, and J. M. Lucas, *Urbano, an Interactive Mobile Tour-Guide Robot*, ch. 14, pp. 229 – 252. Ho Seok Ahn (Ed.), InTech, 2008.

- [40] D. Vogiatzis, C. D. Spyropoulos, S. Konstantopoulos, V. Karkaletsis, Z. Kasap, C. Matheson, and O. Deroo, "An affective robot guide to museums," in *Proceedings of the 4th International Workshop on Human-Computer Conversation, Bellagio, Italy*, 2008.
- [41] A. Chella and I. Macaluso, "The perception loop in cicerobot, a museum guide robot," *Neurocomputing*, vol. 72, no. 46, pp. 760 – 766, 2009.
- [42] A. Koubaa, *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Springer Publishing Company, Incorporated, 1st ed., 2016.
- [43] AGWA, "Aggie - the worlds first art gallery engagement robot," 2016.
- [44] R. Siegwart, K. O. Arras, S. Bouabdallah, D. Burnier, G. Froidevaux, X. Greppin, B. Jensen, A. Lorotte, L. Mayor, M. Meisser, R. Philippsen, R. Piguet, G. Ramel, G. Terrien, and N. Tomatis, "Robox at expo.02: A large-scale installation of personal robots," *Robotics and Autonomous Systems*, vol. 42, no. 34, pp. 203 – 222, 2003. Socially Interactive Robots.
- [45] S. McKeegan, "Toyotas new tour guide robot," Aug. 2007.
- [46] A. Syarif and A. S. Prihatmanto, "Design and implementation of computational platform for social-humanoid robot lumen as an exhibition guide in electrical engineering days 2015," *CoRR*, vol. abs/1607.04763, 2016.
- [47] R. Thrapp, C. Westbrook, and D. Subramanian, "Robust localization algorithms for an autonomous campus tour guide," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 2, pp. 2065–2071 vol.2, 2001.
- [48] C. Chiu, "The bryn mawr tour guide robot," Master's thesis, Bryn Mawr College, 2004.
- [49] K.-H. Chiang, S.-H. Tseng, Y.-H. Wu, G.-H. Li, C.-P. Lam, and L.-C. Fu, "Multisensor-based outdoor tour guide robot ntu-i," in *2008 SICE Annual Conference*, pp. 1425–1430, Aug 2008.
- [50] R. Stricker, S. Müller, E. Einhorn, C. Schröter, M. Volkhardt, K. Debes, and H.-M. Gross, *Konrad and Suse, Two Robots Guiding Visitors in a University Building*, pp. 49–58. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.
- [51] R. Simmons, D. Goldberg, A. Goode, M. Montemerlo, N. Roy, B. Sellner, C. Urmson, A. Schultz, M. Abramson, W. Adams, A. Atrash, M. Bugajska, M. Coblenz, M. MacMahon, D. Perzanowski, I. Horswill, R. Zubek, D. Kortenkamp, B. Wolfe, T. Milam, and B. Maxwell, "Grace: An autonomous robot for the aaai robot challenge," *AI Mag.*, vol. 24, pp. 51–72, June 2003.
- [52] A. Haasch, S. Hohenner, S. Hwel, M. Kleinhagenbrock, S. Lang, I. Toptsis, G. A. Fink, J. Fritsch, B. Wrede, and G. Sagerer, "BIRON - The Bielefeld Robot Companion," in *Proc. Int. Workshop on Advances in Service Robotics* (E. Prassler, G. Lawitzky, P. Fiorini, and M. Hgele, eds.), pp. 27–32, Fraunhofer IRB Verlag, 2004.

- [53] f. Faber, M. Bennewitz, C. Eppner, A. Görög, C. Gonsior, D. Joho, M. Schreiber, and S. Behnke, "The humanoid museum tour guide Robotinho," in *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, (Toyama, Japan), pp. 891–896, Sept. 2009.
- [54] H. M. Gross, H. Boehme, C. Schroeter, S. Mueller, A. Koenig, E. Einhorn, C. Martin, M. Merten, and A. Bley, "Toomas: Interactive shopping guide robots in everyday use - final implementation and experiences from long-term field trials," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2005–2012, Oct 2009.
- [55] J. Gonzalez, C. Galindo, J. L. Blanco, J. A. Fernandez-Madrigal, V. Arevalo, and F. A. Moreno, "Sancho, a fair host robot. a description," in *2009 IEEE International Conference on Mechatronics*, pp. 1–6, April 2009.
- [56] K. Yelamarthi, S. Sherbrook, J. Beckwith, M. Williams, and R. Lefief, "An rfid based autonomous indoor tour guide robot," in *2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 562–565, Aug 2012.
- [57] D. Karreman, G. Ludden, and V. Evers, *Visiting Cultural Heritage with a Tour Guide Robot: A User Evaluation Study in-the-Wild*, pp. 317–326. Cham: Springer International Publishing, 2015.
- [58] A. Canedo-Rodriguez, V. Alvarez-Santos, C. Regueiro, R. Iglesias, S. Barro, and J. Presedo, "Particle filter robot localisation through robust fusion of laser, wifi, compass, and a network of external cameras," *Information Fusion*, vol. 27, pp. 170–188, 2016.
- [59] A. Bicchi, A. Balluchi, D. Prattichizzo, and A. Gorelli, "Introducing the "sphericle" an experimental testbed for research and teaching in nonholonomy," in *Proceedings of International Conference on Robotics and Automation*, vol. 3, pp. 2620–2625 vol.3, Apr 1997.
- [60] A. Halme, T. Schonberg, and Y. Wang, "Motion control of a spherical mobile robot," vol. 1, pp. 259 – 264 vol.1, 04 1996.
- [61] B. P. DeJong, E. Karadogan, K. Yelamarthi, and J. Hasbany, "Design and analysis of a four-pendulum omnidirectional spherical robot," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 1, pp. 3–15, 2017.
- [62] Y. Li, M. Yang, H. Sun, Z. Liu, and Y. Zhang, "A novel amphibious spherical robot equipped with flywheel, pendulum, and propeller," *Journal of Intelligent & Robotic Systems*, pp. 1–17, 2017.
- [63] Y. L. Karavaev and A. A. Kilin, "Nonholonomic dynamics and control of a spherical robot with an internal omniwheel platform: Theory and experiments," *Proceedings of the Steklov Institute of Mathematics*, vol. 295, pp. 158–167, Nov 2016.
- [64] R. Chase and A. Pandya, "A review of active mechanical driving principles of spherical robots," *Robotics*, vol. 1, no. 1, pp. 3–23, 2012.
- [65] M. Matarić, *The Robotics Primer*. Intelligent robotics and autonomous agents, MIT Press, 2007.

- [66] A. R. Cassandra, L. P. Kaelbling, and J. A. Kurien, "Acting under uncertainty: discrete bayesian models for mobile-robot navigation," in *Intelligent Robots and Systems '96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on*, vol. 2, pp. 963–972 vol.2, Nov 1996.
- [67] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-gaussian bayesian state estimation," *IEE Proceedings F - Radar and Signal Processing*, vol. 140, pp. 107–113, April 1993.
- [68] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME—Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.
- [69] A. Canedo-Rodriguez, V. Alvarez Santos, C. Regueiro, R. Iglesias, S. Barro, and J. Presedo, "Particle filter robot localisation through robust fusion of laser, wifi, compass, and a network of external cameras," *Information Fusion*, vol. 27, pp. 170 – 188, 2016.
- [70] H. Moravec and A. E. Elfes, "High resolution maps from wide angle sonar," in *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, pp. 116 – 121, March 1985.
- [71] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.
- [72] D. Hhnel, W. Burgard, and S. Thrun, "Learning compact 3d models of indoor and outdoor environments with a mobile robot," *Robotics and Autonomous Systems*, vol. 44, no. 1, pp. 15 – 27, 2003. Best Papers of the Eurobot '01 Workshop.
- [73] I. Berget, B.-H. Mevik, and T. Næs, "New modifications and applications of fuzzy k-means methodology," *Computational Statistics & Data Analysis*, vol. 52, no. 5, pp. 2403 – 2418, 2008.
- [74] B. Kuipers and Y.-T. Byun, "A robust qualitative method for spatial learning in unknown environments," in *Proceedings of the National Conference on Artificial Intelligence*, (Menlo Park, CA, USA), American Association for Artificial Intelligence, 1988.
- [75] H. Choset and K. Nagatani, "Topological simultaneous localization and mapping (slam): toward exact localization without explicit localization," *IEEE Transactions on Robotics and Automation*, vol. 17, pp. 125–137, Apr 2001.
- [76] S. Thrun, M. Montemerlo, D. Koller, and B. Wegbreit, "Fastslam: A factored solution to the simultaneous localization and mapping problem," in *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pp. 593–598, AAAI, 2002.
- [77] R. Triebel, P. Pfaff, and W. Burgard, "Multi-level surface maps for outdoor terrain mapping and loop closing," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 2276–2282, IEEE, 2006.
- [78] R. Triebel, P. Pfaff, and W. Burgard, "An efficient extension to elevation maps for outdoor terrain mapping and loop closing," *The International Journal of Robotics Research*, vol. 26, no. 2, pp. 217–230, 2007.

- [79] J.-F. Lalonde, N. Vandapel, and M. Hebert, "Data structures for efficient dynamic processing in 3-d," *The International Journal of Robotics Research*, vol. 26, no. 8, pp. 777–796, 2007.
- [80] P. S. Heckbert and M. Garland, "Optimal triangulation and quadric-based surface simplification," *Comput. Geom. Theory Appl.*, vol. 14, pp. 49–65, Nov. 1999.
- [81] D. B. Gennery, "Traversability analysis and path planning for a planetary rover," *Autonomous Robots*, vol. 6, no. 2, pp. 131–146, 1999.
- [82] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," in *Autonomous robot vehicles* (I. J. Cox and G. T. Wilfong, eds.), pp. 167–193, New York, NY, USA: Springer-Verlag New York, Inc., 1990.
- [83] J. A. Castellanos, J. M. M. Montiel, J. Neira, and J. D. Tardós, "The spmap: A probabilistic framework for simultaneous localization and map building," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 5, pp. 948–952, 1999.
- [84] J. Folkesson and H. I. Christensen, "Closing the loop with graphical slam," *IEEE Transactions on Robotics*, vol. 23, pp. 731–741, Aug 2007.
- [85] J. Callmer, K. Granström, J. Nieto, and F. Ramos, *Tree of Words for Visual Loop Closure Detection in Urban SLAM*, p. 102. Australian Robotics and Automation Association, 2008.
- [86] L. Clemente, A. Davison, I. Reid, J. Neira, and J. Tardós, "Mapping large loops with a single hand-held camera," in *Proceedings of Robotics: Science and Systems*, (Atlanta, GA, USA), June 2007.
- [87] P. Newman, D. Cole, and K. Ho, "Outdoor slam using visual appearance and laser ranging," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 1180–1187, May 2006.
- [88] M. Labbe and F. Michaud, "Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 734–745, 2013.
- [89] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*. Santa Clara, CA, USA: Springer-Verlag TELOS, 3rd ed. ed., 2008.
- [90] H.-J. von der Hardt, P. Arnould, D. Wolf, and M. Dufaut, "A method of mobile robot localisation by fusion of odometric and magnetometric data," *The International Journal of Advanced Manufacturing Technology*, vol. 9, no. 1, pp. 65–69, 1994.
- [91] A. Surrecio, U. Nunes, and R. Araujo, "Fusion of odometry with magnetic sensors using kalman filters and augmented system models for mobile robot navigation," in *Proceedings of the IEEE International Symposium on Industrial Electronics, 2005. ISIE 2005.*, vol. 4, pp. 1551–1556, June 2005.
- [92] S. Ginzburg and S. Nokleby, "Indoor localization of an omni-directional wheeled mobile robot," *Transactions of the Canadian Society for Mechanical Engineering*, vol. 37, no. 4, pp. 1043–1056, 2013.

- [93] S. O. H. Madgwick, A. J. L. Harrison, and R. Vaidyanathan, "Estimation of imu and marg orientation using a gradient descent algorithm," in *2011 IEEE International Conference on Rehabilitation Robotics*, pp. 1–7, June 2011.
- [94] T. Moore and D. Stouch, "A generalized extended kalman filter implementation for the robot operating system," in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*, Springer, July 2014.
- [95] R. Xu, W. Chen, Y. Xu, and S. Ji, "A new indoor positioning system architecture using gps signals," *Sensors*, vol. 15, no. 5, pp. 10074–10087, 2015.
- [96] Y. Sung, "Rssi-based distance estimation framework using a kalman filter for sustainable indoor computing environments," *Sustainability*, vol. 8, no. 11, 2016.
- [97] H. T. Friis, "A note on a simple transmission formula," *Proceedings of the IRE*, vol. 34, pp. 254–256, May 1946.
- [98] C. Sapumohotti, M. Y Alias, and S. W Tan, "Effects of multipath propagation and measurement noise in ieee 802.11g wlan beacon for indoor localization," in *Proceedings of PIERS 2012 in Kuala Lumpur*, PIERS 2012, pp. 447 – 451, PIERS Proceedings, 2012.
- [99] A. Martonova and F. Mazan, "A study of devising neural network based indoor localization using beacons: First results," *Computing and Information Systems Journal*, vol. 19, pp. 15–20, 01 2015.
- [100] L. Guo, M. Yang, B. Wang, and C. Wang, "Occupancy grid based urban localization using weighted point cloud," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 60–65, Nov 2016.
- [101] N. Semiconductors, *I2C-bus specification and user manual*, 2014.