



MASTER THESIS

Autonomous Exploration and Navigation with the Pepper robot

Author:
Robert Groot

Supervisor:
prof. dr. Remco Veltkamp

ICA-4148185

Department of Information and Computing Sciences

Oktober, 2018

Abstract

Humanoid robots, such as Softbank's Pepper, are mainly used as eye-catcher during events. Businesses would like to see such robots being used more functionally. They could be used as receptionist or guide. To accomplish these tasks, the robots need to navigate within an environment robustly.

This thesis aims to determine whether Pepper, in combination of existing algorithms, is suitable to create maps, localize and navigate successfully. The research question is as follows: Is Pepper suitable for autonomous exploration, Simultaneous Localization And Mapping (SLAM) and navigation using Robot Operating System (ROS) without the usage of external sensors?

This research question is answered by three experiments. The SLAM experiment has shown that Gmapping can use the 3D sensor to create accurate maps. The second experiment has shown that Pepper can have difficulty localizing itself, causing undesired navigational behaviour. This is caused by the narrow field of view and the relative low quality of the 3D sensor of Pepper. The last experiment has shown that, mainly due to the failure of localization, Pepper cannot autonomously and completely explore an unknown environment.

The results of the experiments indicate that, using the algorithms as defined in this thesis, Pepper is not suitable for autonomous exploration, SLAM and navigation. Further steps could be undertaken to improve these results. Better sensor hardware or adaptations in the algorithms could make the difference in localization and make Pepper suitable for these tasks. Improving the maps manually or automatically, or using landmark detection are also interesting methods that could improve the behaviour of Pepper.

Acknowledgements

I would like to thank Joost Bosman for the introduction of this subject and allowing me to work on it. I am also grateful for the help at ING from Guido van Bommel, Annika Schijf and Ralf Wolter, who made sure I had access to a Pepper robot and helped me within ING. Last but not the least I would like to thank my supervisor at Utrecht University Remco Veltkamp, who introduced me to this subject and guided me along the way.

Contents

Abstract	iii
Acknowledgements	v
1 Introduction	1
2 Platform	3
2.1 Pepper the Humanoid Robot	3
2.1.1 Dimensions	3
2.1.2 Hardware	3
2.1.3 Software	6
2.2 Robot Operating System	6
2.2.1 Concepts of ROS	7
3 Related Work	9
3.1 Pepper	9
3.2 Simultaneous Localization And Mapping	10
3.3 Navigational Algorithms	13
3.4 Exploration Techniques	15
3.5 Research Questions	15
4 Approach	17
5 Experiments	19
5.1 Implementation	19
5.1.1 ROS	19
5.1.2 Pepper setup	20
5.1.3 SLAM	21
5.1.4 Exploration and Navigation	27
5.2 Expected Environment	31
5.3 SLAM performance	31
5.3.1 Design & Objective	32
5.3.2 Results	34
5.4 Navigation performance	42
5.4.1 Design & Objective	42
5.4.2 Results	42
5.5 Exploration performance	45
5.5.1 Design & Objective	45
5.5.2 Results	46
6 Conclusion	49
6.1 Results	49
6.2 Future work	50

A ROS Setup	53
A.1 ROS	53
A.2 Cross compilation	53
A.3 Running ROS	54
B Results	57
Bibliography	71

Chapter 1

Introduction

The interest of using humanoid robots as information and service devices for humans is growing with the advancements in the relevant technologies. For instance, for a banking institution it would be beneficiary if a humanoid robot can interact with customers and give information and answers to their questions quickly. Within a customer office, a robot could welcome customers, answer initial questions and guide towards another location. Looking at the business offices a robot would be handy to welcome guests. After dealing with identification it could guide a guest to the elevators or even to the location of the host. Another use case would be dealing with small internal packages, mail and the like. Letting these tasks be done by a robot is just another form of automation that could save employees time. But while the automation of these tasks seems easy, most tasks are still better handled by humans as the robots require to understand the intentions of the visiting humans. This can be hard for a computer and is an active subject of research. The tasks robots can handle more efficiently are mostly processing structured information with clear intent.

Most robots designed for these human tasks are also shaped like a human: humanoid robots. There are quite a few different options available and they range between simple robots that cost a couple of hundreds of euros to advanced robots costing millions of euros. A frequently used robot in research is the PR2 from Willow Garage. While it has impressive specs and sensors, including two eight-core CPUs and LIDAR, it also costs approximately 240.000 euros. On the more affordable side of the spectrum, Softbank has introduced two social robots: Pepper and Nao. They are both focused on human interaction, with Nao being the smaller robot with a height of 58 centimeter. Pepper is bigger, with its 120 centimeter height, and costs close to 9.000 euro and is frequently used as showcase on tech events.

With the affordability of robots like Pepper, businesses are more and more interested in the possible use cases of such robots and there are quite a few companies that already have one or multiple Peppers. They are used as a showcase and are subject to various fields of research. While Pepper is a robot that is frequently used, it has drawbacks. Specifically, one of the limitations of Pepper is the lack of support of customization and adaptations in the built-in software and packages. Pepper gained some navigational features recently, including a Simultaneous Localization And Mapping implementation and an exploration mode. However, there is little to no room for custom improvements and changes of those features, as it uses an closed operating system for its robots.

Namely, Softbank provides a multi-platform application called Choregraphe with support for the creation of animations, programmed behaviors, testing and monitoring. It can be a useful tool for programming the human machine interaction behaviors and can be used without writing a single line of code. But it has its limitations, as it is intended to be used fully within Pepper's closed environment. This results in little room for optimizations or customizations and no real support for the widely used Robot Operating System (ROS) framework.

In contrast with Choregraphe, ROS works with arbitrary sensor hardware and has multiple state-of-the-art algorithms available as open source packages and is widely used in research. The biggest advantage of ROS is that those packages can be combined and tweaked for a specific robot. For instance, implementing and testing algorithms can be made a lot easier by simulating a robot in stead of using a real robot. This can be achieved without difficulty using Gazebo, a supported robot simulation environment.

Examples of the available open source packages are implementations of Simultaneous Localization And Mapping (SLAM) algorithms. Practically speaking, SLAM means that you can let a robot observe an unknown space and let it automatically create a map with the data it gathers through its sensors. When implemented and working correctly, a SLAM algorithm results in a robot that can navigate in a previously unknown space without predefined models or maps and can do so without colliding with obstacles and walls. Adding an exploration implementation creates a robot that can autonomously explore and map an environment. For the use case of a robot host, this would mean that the robot can be deployed at any location without the need of fully modeling or mapping that environment, as the robot can do that by itself.

The main question that rises, is whether the frequently used Pepper robot is suitable for the state-of-the-art SLAM algorithms implemented using ROS, which have proven useful on other robots. The goal of this thesis is to present a working solution of SLAM using ROS on Pepper and answer the following question: Is Pepper, and specifically its hardware, considered good enough for a decently performing ROS based SLAM implementation?

Chapter 2

Platform

The first section of this overview introduces the software and hardware used in this thesis. The hardware and software specifications of Softbank's Pepper are discussed in the first section. The second section covers the introduction of the Robot Operating System framework. Then the relevant literature is discussed in the third section and lastly the research questions are presented in the fourth section.

2.1 Pepper the Humanoid Robot

Pepper is a humanoid robot manufactured by Softbank Robotics and was introduced in 2014. Designed for human interaction, it comes equipped with voice recognition, superior joint technology with graceful gestures and emotion recognition. This section gives an overview of its hardware. The Pepper used throughout this thesis is version v1.8a¹.

2.1.1 Dimensions

Pepper has a humanoid shape. The total weight of Pepper is 29kg and it is mostly focused in the lower part of the body, optimizing torso and arms movements without falling over. The dimensions are displayed in Figure 2.1.

The joints of Pepper can be found in Figure 2.2a (excluding joints in the fingers) and the details of the wheelbase and wheels in Figure 2.2b. Pepper is a holonomic robot, meaning that the controllable degree of freedom is the same as the total degrees of freedom. This is accomplished by using three omnidirectional wheels and gives Pepper the capability to move and rotate in any direction in the two dimensional plane.

2.1.2 Hardware

This section gives an overview of Pepper's hardware and specifically its sensors. Pepper has an inertial unit, sonars and lasers in its base and multiple cameras in its head. The details of those sensors are described in the next sections. Besides the sensors and cameras, Pepper has hardware available as seen in Table 2.1.

¹http://doc.aldebaran.com/2-5/family/pepper_technical/pepper_versions.html

FIGURE 2.1: Dimensions of Pepper in millimeters. [25]

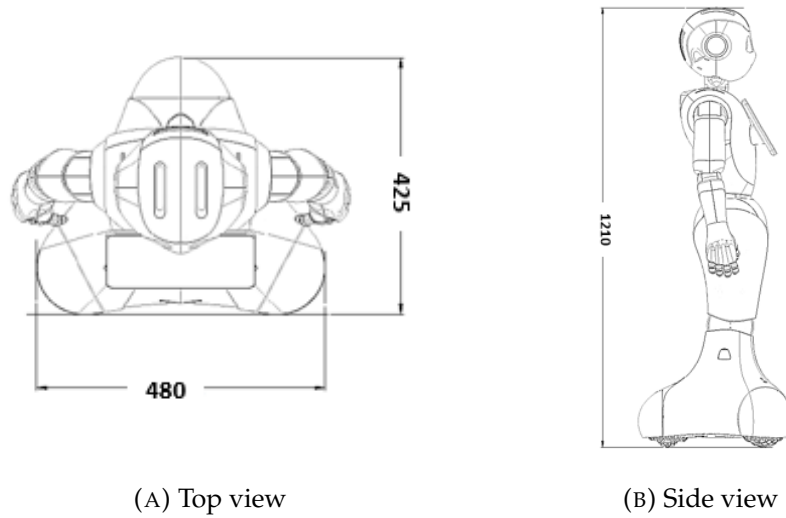


FIGURE 2.2: Joints (left) and wheel base (right) of Pepper. Distances in millimeters. [25]

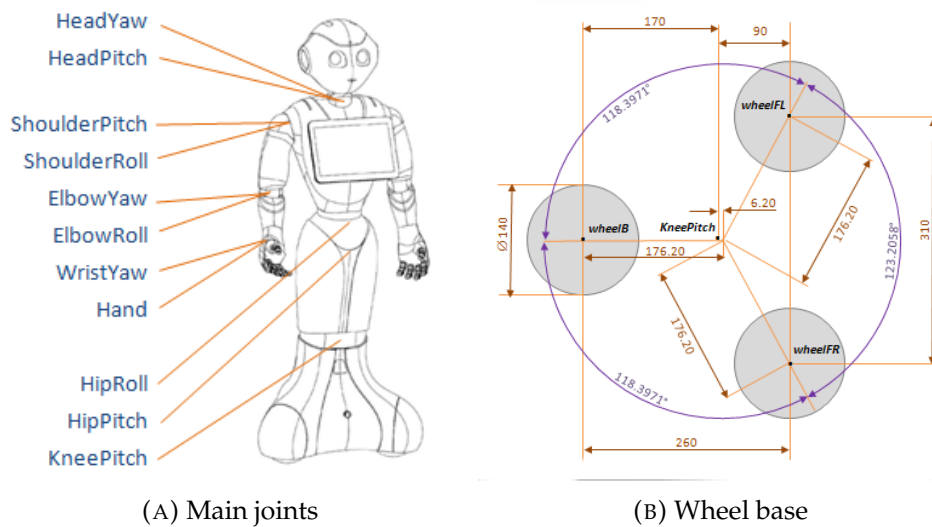


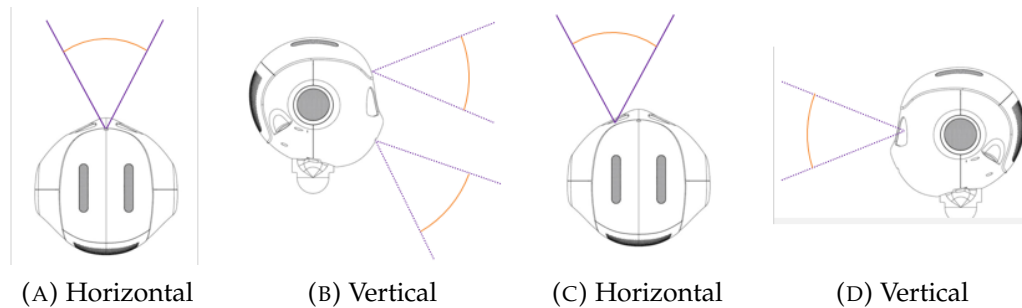
TABLE 2.1: Pepper hardware

CPU	Quad core Atom E3845 @ 1.91 GHz
RAM	4 GB DDR3
Ethernet	RJ45 - 10/100/1000 base T
WiFi	IEEE 802.11 a/b/g/n (WEP, WPA/WPA2)

Cameras

Pepper has two identical cameras mounted in the forehead, providing a resolution of 2560×1920 at 1 frame per second (fps) or 640×480 at 30 fps. The field of view can be seen in Figures 2.3a and 2.3b. As the cameras do not overlap with the field of view, they cannot be used to create a stereo image.

FIGURE 2.3: Field of view of the 2D (2.3a and 2.3b) and 3D (2.3c and 2.3d) sensor. [25]



3D Sensor

One ASUS Xtion 3D sensor is located in the forehead, providing image resolution of 320×240 at 20 fps. The field of view can be seen in Figures 2.3c and 2.3d and it has a focus range of 40cm to 8m.

Lasers

Pepper is equipped with 6 laser line generators. They produce three laser sensors focused on the ground directly in front of Pepper and three laser sensors focused on the surrounding of Pepper (front, right and left). Lasers aimed directly on the ground in front of the robot are located as seen in Figure 2.5a. The surrounding laser sensors can be seen in Figure 2.4. The distance measurement of the lasers are projected onto the ground as seen in Figure 2.5b.

FIGURE 2.4: Location (2.4a and 2.4b) and field of view (2.4c) of the surrounding front, left and right laser sensors, respectively labeled 1, 2 and 3. [25]

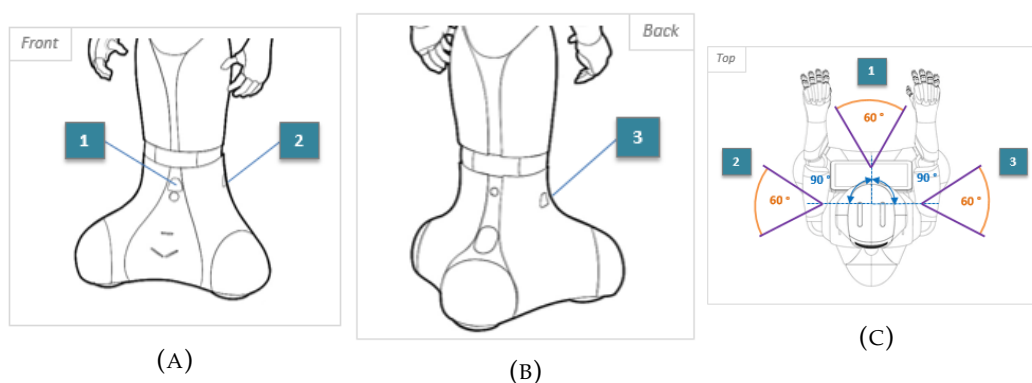
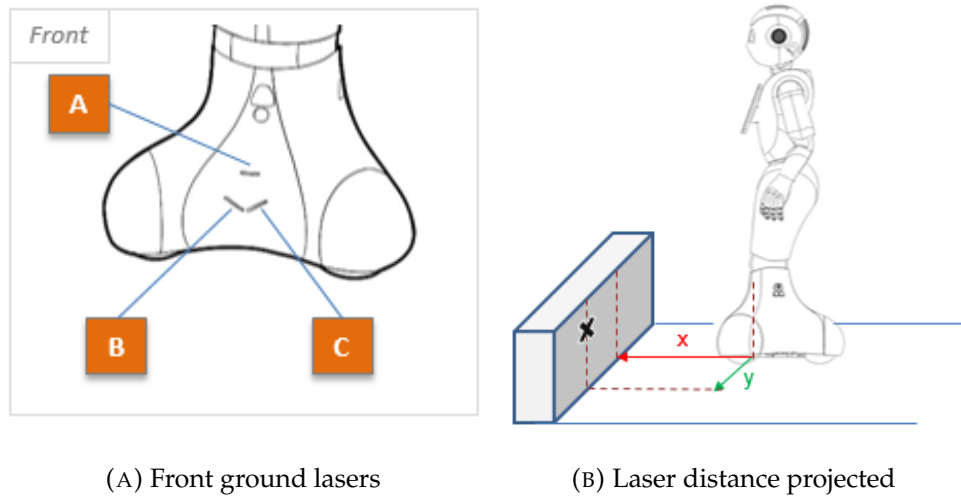


FIGURE 2.5: Front ground lasers (2.5a): shovel, vertical left and vertical right (respectively labeled A, B and C) and the laser distance measurement projected (2.5b) [25]



2.1.3 Software

Pepper runs on the NAOqi OS. It is an embedded GNU/Linux distribution based on Gentoo and includes the and NAOqi framework². NAOqi continuously runs on the robot and acts as a broker. It makes sure the available modules do not crash the whole robot when the programs which control them do crash. NAOqi exposes modules such as motion, vision, people perception and more, making them available for API calls.

The first option to program Pepper is by using Choregraphe³. It connects to NAOqi to upload, test and deploy animations, behaviors and other programs. Animations and simple behaviors can be created without writing a single line of code. It also supports Python to create more complex programs.

The second and more advanced option is using one of the SDKs, with the most supported languages being Python and C++. The Python SDK allows the usage and calls of all C++ API from a remote machine and allows to create Python modules that can be run remotely or on the robot. The C++ SDK can be used to work with the NAOqi framework on a custom platform, use it with a simulator or create software that communicates with the robot. On supported platforms, it can also be used to cross-compile libraries for the robot.

2.2 Robot Operating System

The Robot Operating System (ROS)⁴ is collection of tools, libraries and conventions that aim to simplify the task of creating complex and robust robot behaviour on a wide variety of robotic platforms. ROS was designed to accommodate collaborative

²<http://doc.aldebaran.com/2-5/dev/libqi/index.html>

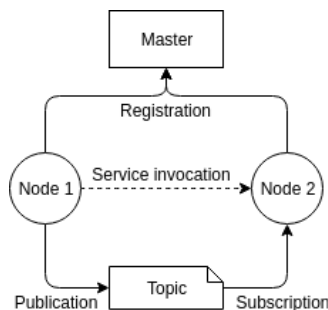
³<http://doc.aldebaran.com/2-5/software/choregraphe/index.html>

⁴<http://www.ros.org/>

software development, giving groups with different expertises the option to collaborate and build upon each other's work. The goals of the framework are that it is thin (so that code written for ROS can be used with other frameworks), that it uses ROS-agnostic libraries and that it is language independent, easily testable and scalable.

ROS runs as a peer-to-peer network of processes called the ROS Computation Graph and is potentially distributed. The processes are loosely coupled using a ROS communication infrastructure. Several styles of communication are implemented, including synchronous RPC-style communication, asynchronous streaming of data and storage of data. Some of the concepts of the ROS network are *Nodes*, *Master*, *Messages*, *Services*, *Topics* and *Bags*. These concepts will be briefly discussed in the next paragraphs, which cover the core concepts of ROS. An example of a simple Computation Graph can be seen in Figure 2.6.

FIGURE 2.6: Simple ROS Computation Graph



2.2.1 Concepts of ROS

Nodes are processes that perform the actual computations. They are designed to be modular. A system usually has many nodes, each controlling its own part of a robot. For example, one node may control the motors of the robot, one node provides navigation, one node provides object recognition and so on.

The **Master** acts as a mediator and provides name registration and lookup of the complete Computation Graph, much like a DNS server. There is always strictly one master, without it nodes cannot find each other and interact.

Nodes communicate by passing **Messages**. Messages are simply data structures, with support for standard primitive types, arrays of those types and nesting of structures and arrays.

The messages are routed through a publish / subscribe transport system. A node sends out a message by publishing it to a **Topic**. The topic has a name so that it can be identified by other nodes. A node can subscribe to a topic when it is interested in the data it has. Each topic can be seen as a strongly typed message bus and can have multiple subscriber and publisher nodes, as long as the type of the published and subscribed messages is correct. The nodes are not aware of each other's existence.

For request / reply interactions the topics are not appropriate. **Services** are defined by a pair of message structures, one for the request and one for the reply. A node can offer a service under a name and other nodes can use the service by sending the request message.

Bags are a data format for saving and playing back ROS messages. The bags can be used to record sensor data and reuse it at a later time, which make them useful for development, testing and experimenting of algorithms and other programs.

Chapter 3

Related Work

The literature that has been studied during the preparation phase of this project is presented in this chapter.

Several experiences, implementations and challenges observed by working with Pepper are described in Section 3.1. Section 3.2 is focused on Simultaneous Localization and Mapping (SLAM) algorithms, as the navigation of robots rely on a known map of the environment. In order to apply a mapping algorithm, a robot needs to observe an unknown environment and navigate to unknown parts of that environment. The navigational algorithms that could be used are covered in Section 3.3. Besides driving the robot manually, Section 3.4 explores techniques for autonomous exploration. These algorithms provide methods to reach a main goal of humanoid robots: autonomous behaviour.

3.1 Pepper

Pepper has already been subject to a number of research topics. This section explores the relevant subjects with regard to previous experiences with Pepper, Robot Operating System, Simultaneous Localization And Mapping and the combination of these subjects.

Softbank has created a coherent and well-thought development environment for the Nao/Pepper robots, called Choregraphe. It has the possibility to read signals from the sensor, process these information and control the robot. However, experience indicates that for the implementation of a stable and more advanced application, the dedicated SDKs should be used [9].

Robot Operating System (ROS) is a structured communications layer above host operating systems. It is designed to support the philosophy of modular, tool-based software development. ROS reuses code from numerous other open-source projects, such as hardware drivers, navigation system, and simulators. In each case, ROS is used only to expose various configuration options and to route data into and out of the respective software. Several hundred (or more) ROS packages exist across several publicly viewable repositories[21].

Methods to run ROS locally on Pepper already exist. Perera et al. [20] attempt to improve Peppers capabilities by increasing the awareness of the environment and adding personalized interactions. The improvement in awareness is implemented within a ROS environment and they describe a method to run ROS Indigo locally on the NAOqi operating system. Namely, the ROS source code is cross compiled on a

virtual machine. After copying the compiled code to Pepper, the ROS nodes can be launched on Pepper. The used ROS package, `naoqi_driver`, uses the NAOqi SDK to convert the information from Pepper into ROS messages.

Guerron[12] shows a positive review about the characteristics shown after implementing a social forces model on Pepper. The social forces model was implemented as a ROS local planner and used in combination with NAOqi's People Detection module to keep track of a partner. The Gmapping and AMCL packages were used for the initial creation of a map of the environment and localization, and a set of parameters for Pepper is given. Additionally, Guerron notes that the built-in lasers of Pepper are not good enough for the purpose of a SLAM algorithm. The depth image obtained from the depth sensor is converted to a laser scan to obtain better results. A drawback of this approach is that the head movements of the head of Pepper should be limited in order to prevent sudden movements. Another note is that the created map was digitally edited to improve the results of the main experiment.

Review & Conclusion

Given the experiences of others and after a brief introduction of Choregraphe, it is clear that the software environment of NAOqi is not fit for advanced algorithms and the like. Instead, ROS is designed for these kind of implementations and has many advantages. These benefits, including but not limited to inter-platform operability, modularity, resource handling and simulation support lead to the choice of using ROS.

Within the ROS research area, Pepper has already been subject to several experiments. While some of them were within the field of navigation and simultaneous localization and mapping, there are no conclusive results showing the performance of Pepper with regard to available state of the art autonomous SLAM, exploration and navigation algorithms. The benefits of ROS and the lack of SLAM research on Pepper leads to the first and main research question.

RQ: Is Pepper suitable for autonomous exploration, SLAM and navigation using ROS without the usage of external sensors?

For sensor hardware, the environment is important. Factors such as glass windows, moving people and direct sunlight all influence sensor readings and therefore can impact performance of exploration, SLAM and navigation. This raises the first sub-question.

SQ1: What is the expected environment for the usage of SLAM, exploration and navigation?

3.2 Simultaneous Localization And Mapping

Simultaneous Localization And Mapping (SLAM) is the problem of constructing and updating a map of an unknown environment while keeping track of the robot's location within it. As building maps is one of the fundamental tasks of mobile robots, a lot of researchers have been focused on this problem. Some of the relevant approaches are discussed in this section.

Mapping Techniques

Mapping algorithms can be roughly classified according to the map representation and the underlying estimation technique. One popular map representation is the occupancy grid. While these grid-based approaches are relatively computationally expensive, they are able to represent arbitrary and detailed features. Feature based representations can be attractive because of their compactness, but they rely on feature extractors which assume some knowledge of the structures in the environment [7]. The most popular estimation techniques are based on the Extended Kalman filters (EKFs), graph-based optimization techniques or particle based methods [26].

Grisetti¹, Stachniss, and Burgard [11] present an improved approach to learning grid maps using Rao-Blackwellized Particle Filters. An on the fly computed improved proposal distribution is computed, in stead of a previously used fixed proposal distribution. This allows for the direct usage of most of the information obtained from the sensors while generating the particles. While the computation of the proposal distribution is similar of that of others, it does not rely on predefined landmarks. This approach also draws the particles more effectively and the highly accurate proposal distribution gives a robust indicator to decide whether a resampling has to be carried out. Grisetti, Stachniss, and Burgard [10] optimized the proposal distribution to even more accurately draw the next generation of particles. Experiments show highly accurate metric maps and the number of required particles is one order of magnitude lower than previous approaches. An implementation is available as open source ROS package named `Gmapping`¹.

While `Gmapping` relies on sufficiently accurate odometry, `Hector Mapping`² utilizes the high update rate of modern LIDAR systems. This approach, proposed by Kohlbrecher et al. [14], is based on optimization of the alignment of beam endpoints with the map learned so far. The basic idea is to use a Gauss-Newton approach, similar to the idea of a Kalman Filter. Using this approach there is no need for a data association search between beam endpoints or an exhaustive search. As scans align with the existing map, all preceding scans are implicitly matched with the current one. The system is sufficiently accurate to close loops typically encountered in small scale scenarios without use of an approach for explicit loop closure, keeping computational requirements low and preventing changes to the estimated map during runtime.

Another frequently used ROS SLAM implementation is `Cartographer`³ [13]. The individual submaps created by the local grid-based SLAM approach do not use a particle filter to achieve better performance on modest hardware. To cope with the accumulation of error, a pose optimization is run regularly. When no new scans will be inserted in a submap, it takes part in the loop closure. The loop closure constraints are created using pixel-accurate scan matches.

Besides the (laser) point cloud based approaches mentioned so far, visual mapping algorithms exist. However, using a visual sensor requires algorithms with good and consistent performance under variable light conditions, apparition of featureless regions and other unforeseen situations. Some algorithms fail to detect light colored office-style wall surfaces, leaving gaps in point clouds of a sparse 3D scene, where

¹<http://wiki.ros.org/gmapping>

²http://wiki.ros.org/hector_mapping

³<http://wiki.ros.org/cartographer>

the surface of walls should be. These algorithms could be unsuitable for indoor SLAM because of the collision risk [3, 8].

ORB-SLAM2 [17, 18] is an open source system for monocular, stereo and RGB-D cameras and includes loop closure, relocalization and map reuse. The system pre-processes the input to extract monocular or stereo features as key-points, so that it is independent of the sensor being stereo or RGB-D. Mur-Artal and Tardós propose a bundle adjustment based back-end which results in a lightweight method that works on standard CPUs. An implementation is openly available as ROS package `orb_slam2`⁴.

Another approach, proposed by Labbé and Michaud [15], combines loop closure detection and graph optimization through a memory management process to achieve a good SLAM performance. A combination of a range finder laser scan and a RGB-D depth cloud is used for the creation of a map and the RGB images are used for the appearance based loop closure. While the memory management makes sure that the processing time stays reasonable, errors can occur when maps are written to the long term memory before a loop closure. However, revisiting old maps, and possibly detecting more loop closures increases the global map quality. The implementation in the form of a ROS package is called `rtabmap_ros`⁵.

Evaluation Techniques

For the evaluation of these SLAM techniques, a comparison metric between maps is needed. When a ground truth is available, it can be directly compared to the computed map to obtain a metric of error. Santos, Portugal, and Rocha [24] align the binarized map with the respective ground truth. Then the distance from each occupied cell of the ground truth to the nearest cell of the computed map is determined using *knnsearch*. The sum of these distances is divided by the number of occupied cell to obtain a normalized error metric, suitable for evaluation.

However, a ground truth is not always available and thus Filatov et al. [5] present three metrics to evaluate the quality of a SLAM algorithm without the use of a ground truth. The first metric is the proportion of the occupied cells. This can be used to determine whether a map has blurred or extra walls because of a failure of the algorithm. The second metric is the amount of corners in a map. This measures the precision of a map, as a low quality map should have more corners than the accurate one in case of overlapping or inconsistent curvature of straight walls. Lastly, the amount of enclosed areas is measured to detect overlapping rooms or artifacts.

Lastly, Pålsson and Smedberg [19] show that when there is no ground truth map is available, real world measurements can also provide an error metric for SLAM algorithms. Measurements from a hand-held laser rangefinder are compared with measured distances in the map in multiple places to obtain the error metric.

Review & Conclusion

SLAM algorithms are widely available and comparisons have been made. The laser point cloud based algorithms depend on different properties of certain hardware.

⁴https://github.com/raulmur/ORB_SLAM2

⁵http://wiki.ros.org/rtabmap_ros

Gmapping relies on sufficiently accurate odometry, Hector Mapping relies on the high refresh rate of LIDAR systems, Cartographer relies on the created submaps and loop closures. However, the comparisons between the algorithms and observations that have been made, are using mostly different hardware than the hardware Pepper has available. This lack of information about the performance using other (Pepper like) hardware supports the need for a comparison with a special focus on Pepper's hardware.

While Pepper does have some laser sensors, they are not comparable to modern LIDAR systems and are probably not usable for SLAM purposes. It does have a 3D sensor, which can create a point cloud of detected depths. Besides the 3D sensor, Pepper also has normal RGB cameras which can also be used for visual SLAM algorithms.

The lack of knowledge about whether the sensor data (or a combination of sensors) is sufficient for the mentioned SLAM algorithms gives us the second sub-question:

SQ2: How does Pepper perform using SLAM algorithms, with regard to an expected environment?

The evaluation techniques mentioned above present a way to compare results of the algorithms, even when there is no ground truth available. When a ground truth is present they can be compared more directly and precise however. So these techniques provide ways to answer SQ1 using comparable metrics.

3.3 Navigational Algorithms

The goal of the SLAM and exploration techniques is to obtain a map for a robot to navigate within. For the navigational aspect there are multiple algorithms available. This section briefly covers relevant techniques that are available. First the global planners and then the local planners are discussed.

Global Planners

Navigational tasks in ROS are implemented in the Navigation Stack⁶. It takes information from odometry, sensor beams and a goal pose and outputs velocity commands to a robot base. By default, a couple of options in terms of planners are available. `global_planner` is built as a flexible global path finder, is based on work by Brock and Khatib [2] and includes support for Dijkstra's, A*, grid paths and more. It works in combination with one of the local planners, which are described in the next section called **Local Planners**.

`spqrrel_navigation`⁷, with its approach presented in [16], is a navigation system requiring very few computational resources, but still provides performance comparable with commonly used tools in the ROS environment. Using a fast global path planner that adapts the computed path to dynamic changes in the environment, it only needs to consider planning at a global level. Eventually, Dijkstra's algorithm is used to compute the minimal paths to a goal. It also works both with ROS and NAOqi middle-wares.

⁶<http://wiki.ros.org/navigation>

⁷https://github.com/LCAS/spqrrel_navigation/

Local Planners

Most global planners need a local planner to provide the eventual speeds needed to follow the global path. Dynamic obstacles are taken into account as well in the local planners, with the exception of some global planners that take them into account as well.

The Dynamic Window Approach (DWA)⁸ [6] produces a circular trajectory that is optimal for a robot's local position. It maximizes an objective function that depends on the progress to the target, clearance from obstacles and forward velocity to produce the optimal trajectory. It depends on the local costmap for obstacle information, making tuning the local costmap parameters crucial.

The `dwb_local_planner`⁹ extends the DWA and aims to make it as customizable as possible. Just as DWA, the trajectories are sampled, but instead of evaluating velocities in isolation, it scores an array of sample poses that are anticipated along that trajectory.

Rösmann, Hoffmann, and Bertram [23] propose to use Timed-Elastic-Band to implement an online optimal local trajectory planner for navigation. A sparse scalarized multi-objective optimization problem is solved to obtain the optimal trajectory. Since these local planners get stuck in locally optimal trajectories as they cannot transit across obstacles, an extension is implemented. In parallel, a subset of possible trajectories is optimized. Within that candidate set, the planner is able to switch, making it possible to transit across obstacles. It is implemented in the `teb_local_planner`¹⁰ ROS package.

The idea of elastic bands [22] is also used in the implementation of `eband_local_planner`¹¹. The idea is to deform the path computed by the global planner with two forces: an internal contraction force to simulate tension in the elastic band and an external repulsion force to avoid obstacles.

Conclusion

The global and local planners provide the actual navigation within the ROS Navigation Stack. Depending on the requirements, such as CPU performance and support for dynamic obstacles, a particular implementation of the planners should be used. Pepper has a mid-end CPU and the variation in CPU usage can be one of the interesting differences of the algorithms. Furthermore, the expected environment can make a difference as well. The navigation is also influenced by whether dynamic obstacles, such as moving people, are taken into account. The question that follows, is:

SQ3: What navigation technique performs best on Pepper, given a created map and expected environment?

⁸http://wiki.ros.org/dwa_local_planner

⁹https://github.com/locusrobotics/robot_navigation/tree/master/dwb_local_planner

¹⁰http://wiki.ros.org/teb_local_planner

¹¹http://wiki.ros.org/eband_local_planner

3.4 Exploration Techniques

Mapping algorithms need to observe unknown areas of an environment in order to create a map out of them. While this can be done manually by teleoperation, autonomous exploration is preferred, especially when the goal is autonomous robot behaviour. A couple exploration packages from ROS are discussed in this section.

A widely used concept for exploration is that of frontiers. Frontiers are defined as regions on the border between space known to be open and unexplored space. An approach, presented by Yamauchi [28], uses frontier detection to set goals for exploration. They use computer vision techniques similar to edge detection and region extraction to find boundaries between open space and unknown space. Open cells adjacent to unknown cells are grouped into frontier regions and any frontier region above a certain size is considered a frontier. Then the path planner is used to navigate towards a frontier, probably observing new sensor data. This process is repeated until there are no more frontiers. This type of exploration is used in multiple ROS packages such as a direct implementation called `frontier_exploration`¹², and some variations including but not limited to, `explorer`¹³ [4], `explore`¹⁴ [1], `hector_exploration`¹⁵ [27]

Review & Conclusion

For the complete autonomous mapping of an environment, a robot needs to explore the unknown parts while using a SLAM algorithm. There are some exploration packages available within ROS and almost all used packages rely on frontier exploration. However, the small distinctions between the implementations make a clear choice difficult.

Regarding Pepper, during the literature review no results were found on frontier exploration results using Pepper. This raises the following sub-question:

SQ4: How does a frontier exploration of a whole unknown environment with Pepper perform, while creating a map using a ROS SLAM algorithm?

3.5 Research Questions

Given the main research question

RQ: Is Pepper suitable for autonomous exploration, SLAM and navigation using ROS without the usage of external sensors?

the following sub-questions have been formulated to answer the main research question:

SQ1: What is the expected environment for the usage of SLAM, exploration and navigation?

¹²http://wiki.ros.org/frontier_exploration

¹³<http://wiki.ros.org/explorer>

¹⁴<http://wiki.ros.org/explore>

¹⁵http://wiki.ros.org/hector_exploration_planner

SQ2: How does Pepper perform using SLAM algorithms, with regard to an expected environment?

SQ3: How does Pepper perform navigating in a SLAM created map of an expected environment?

SQ4: How does a frontier exploration of a whole unknown environment with Pepper perform, while creating a map using a ROS SLAM algorithm?

Chapter 4

Approach

This chapter describes the used approach in this thesis. It includes how the research questions will be answered and covers the experiments on a general level.

As defined in Section 3.5, RQ formulates the question whether Pepper is suitable for particular exploration, SLAM and navigational tasks. For this question to be answered, the sub-questions that have been formulated need to be answered.

SQ1 will be answered by determining specific use cases and locations of those use cases.

For SQ2, an experiment is set up. The goal of the experiment is to determine what SLAM algorithm works best in combination with the hardware Pepper has available. First of all, data is gathered by driving Pepper around in the environments defined by the answers of SQ1. A ROS bag is used to record data from the sensors, motors and cameras. By using a ROS bag, the same data can be used repeatedly and makes sure the different algorithms perform on the same input data. Multiple ROS bags are recorded, making sure that single outliers do not influence the score of algorithms significantly.

The different SLAM algorithms and configurations are setup for Pepper. By running the algorithm on the ROS bag data, a resulting map can be obtained. Initially, a working configuration is found by tweaking different parameters empirically. The initial working configuration is then slightly tweaked on multiple parameters that impact the map result significantly, creating the different input configurations for each of the algorithms. Using the metrics mentioned in Section 3.2, the results can be compared and the difference in score and thus performance can be shown. If a ground truth is available, it will be used as reference metric. If no ground truth is available, the alternative metrics will be used. Looking at the metrics and especially the metric of the best scoring algorithm, SQ2 can be answered. A map is considered good when no critical features (such as walls or other obstacles) are significantly misrepresented.

The second experiment answers SQ3. The best scoring SLAM algorithm and setup determined by the answer of SQ2 is used in this experiment. The exploration algorithm is implemented and multiple configurations are prepared. Pepper is put in a environment compatible with the answer to SQ1. The exploration algorithm will be used to let Pepper explore the unknown environments. By exploring, the new observed data is used to create a map using the previously determined SLAM algorithm. When the exploration algorithm is done, the result will be a map. The same metrics used in the first experiment can be used to determine the quality of the results. By comparing the metrics obtained, the algorithms and configurations

can be ranked and the highest scoring algorithm gives the answer to SQ3. When a high percentage of the map is explored and the resulting map is also considered sufficient, the exploration is also considered good.

Lastly, the third experiment is set up to answer SQ4. The different configurations will be set up for Pepper. This time, Pepper is set up in a single environment compatible with the answers of SQ1. A predefined map will be used. It can be created by a SLAM algorithm if the result of SQ2 is satisfactory. If not, a map can be created manually to continue the third experiment. A couple of navigational goals are determined and are the same for each algorithm. Depending on the answer to SQ1, dynamic and / or static obstacles are introduced. Characteristics such as efficiency, speed and collisions are used to compare different algorithms. That comparison answers SQ4. When there are no significant collisions, and the efficiency and speed are reasonable, an algorithm is considered good.

When all the sub-questions are answered, the main research question can be answered. Pepper is considered suitable for autonomous exploration, SLAM and navigation when the results of SQ2 are satisfactory and result in usable maps, the results of SQ3 are good such that most of the unknown space is explored and the results of SQ4 are good such that navigation is efficient and does not result in significant collisions.

Chapter 5

Experiments

The experiments that answer the research question are listed in this chapter. Firstly an overview of the implementations is given and then the designs, objectives and results of the experiments are covered.

5.1 Implementation

In the first section, specifics about ROS and the cross-compilation of ROS are covered. Secondly the configuration of Pepper is discussed. At last the algorithms and their configurations are described. The full installation guide for reconstructing the environment of these experiments can be found in Appendix A.

5.1.1 ROS

Hardware

The main ROS node (running `roscore`) and the algorithms used are run on a laptop. This choice is made to avoid loss in performance due to the weaker hardware of Pepper (see Section 2.1.2). The hardware of the laptop used in this thesis can be seen in Table 5.1. This results in Pepper being a mainly publishing node and the laptop being a node that processes all the data and runs the algorithms (and sending move commands if needed).

TABLE 5.1: Internal hardware

CPU	Intel® Core™ i7-6700HQ CPU @ 2.60GHz
GPU	nVidia GeForce GTX 960M
RAM	16GB
Ethernet	USB C adapter to RJ45 - 10/100/1000

Software

For the conversion between the NAOqi OS and ROS the `naoqi_driver` package is needed. As said before, this package converts the information from NAOqi to ROS messages. According to the documentation¹ the newest working version at this time

¹https://github.com/ros-naoqi/naoqi_driver

runs on Ubuntu Xenial and works with ROS Kinetic. Thus, all experiments in this thesis are run using a fresh install of Ubuntu 16.04 LTS (Xenial) and ROS Kinetic.

ROS is installed using the `ros-kinetic-desktop-full` package for Ubuntu and includes ROS and tools for visualization, simulation and navigation².

Cross-Compilation

Due to the fact that a user does not have root or sudo rights on Pepper, ROS cannot be installed directly on the robot. This is solved by cross-compiling ROS and the needed packages on another machine. A Docker container is used for this task.

Before compiling, a small change to the `naoqi_driver` package is made. This package converts the NAOqi information to ROS messages. In the current version, the camera distortion of the depth camera is not available in the messages. The algorithms that convert the depth images to laser scans need this camera distortion. To enable this, the lines that add the distortion to the messages can just be uncommented in the source of the `naoqi_driver`.

The SLAM algorithms are run on the remote laptop, meaning that the packages for those algorithms are not needed on Pepper locally. The compiled ROS packages are eventually transferred to Pepper and can be run normally. Specific and complete instructions can be found in Appendix A.

5.1.2 Pepper setup

The setup used to run ROS with Pepper is discussed in this section and is the same in all three experiments.

When working with ROS on Pepper, all other programs need to be stopped to prevent interference. The autonomous behaviour specifically, but other custom programs as well, can be stopped using Choregraphe. In Choregraphe, one can also lock motor function. The head of Pepper is locked looking straight ahead using this function. It should be noted that the motors can overheat when they are locked for a longer period of time. After overheating, the robot needs some time to cool before it can be moved and locked again.

Simple access points have failed to provide the bandwidth that is needed to stream all of the camera data to the laptop. Pepper is connected with the laptop through a cable to prevent this issue from happening. A non wireless connection can mean limited movement, but by actively making sure the cable does not get stuck while avoiding interference with the sensor data, this problem is nonexistent.

The emergency stop functions, that make sure Pepper does not collide with obstacles, can be kept enabled. When such a stop occurs, the robot stops moving and can only be moved in directions it deems safe.

Lastly, when needed, manual movement of Pepper is done by the `teleop_twist_keyboard`³ package. This package listens for key presses and sends move commands as ROS messages.

²<http://wiki.ros.org/kinetic/Installation/Ubuntu>

³http://wiki.ros.org/teleop_twist_keyboard?distro=kinetic

Depth sensor

All algorithms used in this thesis are 2D mapping algorithms. This means that there is only information about the environment available in the two dimensional plane, mostly located at ground level. As the resulting map is 2D, the input data also need to be 2D. In practice, the input data consists of laser beam end points, projected to the ground.

The only sensor of Pepper that is suitable and deliver enough of such input data is the depth sensor. The initial data from the depth sensor consists of grey-scale depth images. To convert the depth images to laser scans the combination of the `depthimage_to_pointcloud` and `pointcloud_to_laserscan` package is used, configured as seen in Table 5.2. The algorithm works by transforming the depth image pixels to a point cloud using the available transforms published by the `naoqi_driver`. That point cloud is converted to a laser scan by projecting the points to the ground using the provided configuration from the `pointcloud_to_laserscan` package.

TABLE 5.2: Depth image to point cloud to laser scan configurations.

Parameter		Setup 1	Setup 2
<code>min_height</code>	(m)	0.4	0.4
<code>max_height</code>	(m)	1.5	1.5
<code>angle_increment</code>	(rad)	$\pi/720$	$\pi/720$
<code>range_min</code>	(m)	0.50	0.50
<code>range_max</code>	(m)	5.0	12.0

The `min_height` parameter determines the height under which points from the point cloud are ignored. This is set to 40cm, because the depth camera of Pepper creates noise on flat surfaces. The noise causes the point-cloud-to-laser-scan node to publish laser points on flat floors. Eventually, that generates nonexistent obstacles on flat floors, which is not desirable. The drawback of this solution is that Pepper does not see small obstacles lower than this height. The `max_height` parameter determines the height above which points from the point cloud are ignored. This is set to 1.40m to ignore all points that are higher than Pepper and also excludes ceilings from potentially being detected as obstacles. The `angle_increment` parameter is the resolution of the laser scan in radians per ray. The `range_min` parameter is the range under which points are ignored. Pepper's depth camera does not perform well with objects really close, so this is set to 50cm. The `range_max` parameter is tested with two options. The first one is set to 12 meters. This is a little bit more than the average usable range of the 3D sensor. The second option is set to 5m. This is done because of the reliability of the depth camera at long range. In long ranges the amount of noise on the floor and other areas increases. Setting the range max to 5m attempts to solve this, while preserving the ability to see obstacles that are nearby.

5.1.3 SLAM

As for the SLAM algorithms there are three different methods used and compared. The three algorithms used in this thesis are `slam_gmapping`⁴, `hector_slam`⁵ and

⁴http://wiki.ros.org/slam_gmapping?distro=kinetic

⁵http://wiki.ros.org/hector_mapping?distro=kinetic

Cartographer⁶. Their configurations are covered in the next sections.

The created maps are all grid based maps. This means that every cell in the map corresponds with an area in the real world. For every cell, the value of that cell defines whether that cell is occupied, free or unknown. For all algorithms covered below, the map size is set to 2048 and the resolution to 0.05, meaning that every grid cell length is 5 cm in the real world and the map has 2048 cells along the two axes. For the experiments in this thesis, this means that the maximum area that can be mapped is 102.4m x 102.4m. The scenes are all smaller than that area, so the robot should not exceed the map limits.

In ROS, the maps created are saved as grey-scale images. This is a convenient format to save the values for each grid cell and it also gives a visual representation of the map. In the navigation system, those images are loaded and used for the localization and navigation (see Section 5.1.4).

Lastly, for all of the SLAM algorithms, the laser scan topic originating from the depth camera is used.

slam_gmapping

Table 5.3 shows the parameters of Gmapping that have been altered, as well as their default values. Empirical tests in the early phase that have resulted in a somewhat useful map give us the starting set of parameters and are also shown. These parameters were obtained by starting with the default values and were changed to comply with the setup of Pepper and it's hardware. The changed parameters are discussed in the next paragraphs.

TABLE 5.3: Gmapping custom parameters.

Parameter	Default	Used
map_update_interval	5.0	1.0
maxUrange	80.0	12.0
maxRange	-	15.0
minimumScore	0.0	variable
srr	0.1	variable
srt	0.2	variable
str	0.1	variable
stt	0.2	variable
linearUpdate	1.0	0.5
particles	30	50
xmin	-100.0	-50.0
ymin	100.0	50.0
xmax	-100.0	-50.0
ymax	100.0	50.0
transform_publish_period	0.05	0.01

The lower value of `map_update_interval` means the occupancy grid is updated more often. This is at the expense of greater computational load, but it was done because the algorithm is run on a reasonably powerful machine.

⁶<https://google-cartographer-ros.readthedocs.io/en/latest/>

The `maxUrange` and `maxRange` parameters are respectively the maximum usable range and the maximum range of the laser input data. These settings are set to the maximum range of the 3D sensor. Further adjustments to this ranges are handled by the depth image to point cloud to laser scan configuration discussed in the previous section. The initial empirical tests have shown that changing the `minimumScore` and error in translation and rotation parameters (`srr`, `str`, `srt`, `stt`) mattered the most. `minimumScore` is the minimum score for considering the outcome of the scan matching good. Increasing this parameter can avoid jumping pose estimates in large open spaces when using laser scanners with limited range. Pepper's depth sensor does have a limited range (and field of view), so `minimScore` is used as one of the experimental parameters.

Furthermore, as Guerron[12] mentions in his thesis, the odometry of Pepper is not fully accurate. Especially the rotational sensor data is not correct after turning. The parameters `srr`, `srt`, `str` and `stt` can be helpful in solving such problems with odometry and thus are chosen as variables in the experiments. `srr` is the odometry error in translation as a function of translation, `srt` the odometry error in translation as a function of rotation, `str` the odometry error in rotation as a function of translation and `stt` the odometry error in rotation as a function of rotation. Setting these values to 0 means the algorithm interprets the odometry sensor values as perfect.

Table 5.4 shows the eventual parameter options, used in the configurations of the Gmapping SLAM experiment.

TABLE 5.4: Gmapping variable parameters.

Parameter			
<code>srr</code>	0.0	0.1	
<code>str</code>	0.0	0.1	
<code>srt</code>	0.0	0.1	
<code>stt</code>	0.0	0.2	0.02
<code>minimumScore</code>	100	200	

hector_slam

Hector Mapping (`hector_slam`) does not have a lot of parameters. They can be found in Table 5.5.

The `map_update_distance_thresh` and `map_update_angle_thresh` are the thresholds for performing map updates. When the platform moves or rotates more than the thresholds, a map update occurs. The distance threshold is set to a relatively short distance, and the angle threshold is set as variable.

The period between map updates is set by `map_pub_period` and is set to 1 second to increase the updates of the map. When using a reasonably powerful machine, lowering this value does not hinder the performance of the actual SLAM.

Hector Mapping works using multi-resolution grid levels. For every level, the resolution is twice as high, and together they are used to compute the best pose estimation. The number of levels, `map_multi_res_levels`, is set as a variable.

The minimum and maximum distances of the laser are left as is, as the point-cloud-to-laser-scan node handles these settings as well.

TABLE 5.5: Hector Mapping custom parameters.

Parameter	Default	Used
map_update_distance_thresh	0.4	0.4
map_update_angle_thresh	0.9	variable
map_pub_period	2.0	1.0
map_multi_res_levels	3	variable
update_factor_free	0.4	0.4
update_factor_occupied	0.9	0.9
laser_min_dist	0.4	0.4
laser_max_dist	30.0	30.0
pub_map_odom_transform	true	true
scan_subscriber_queue_size	5	5
pub_map_scanmatch_transform	true	true

pub_map_odom_transform determines whether the system should output the transform between the odometry and the map and pub_map_scanmatch_transform determines whether the scan matcher transform should be published to the transform data (/tf). Both of these are desired and thus set to true. Lastly, pub_map_scanmatch_transform can be used to play back log files faster than real time. As this is not done, this is set to the default value.

The values of the variable parameters can be seen in Table 5.6. These values are chosen to show the difference between a high update rate from rotation, to compensate for the error in odometry, and the difference in number of resolution layers.

TABLE 5.6: Hector Mapping variable parameters.

Parameter		
map_update_angle_thresh	0.1	0.9
map_multi_res_levels	2	6

Cartographer

The Cartographer configuration system is quite different than the other SLAM algorithms. It uses lua configuration files. The main configuration can be seen in Table 5.7. Included in all configurations are the default map builder and trajectory builder configuration files. The defaults and variable parameters for this experiment can be seen in Tables 5.8 and 5.9. The variable parameters are discussed in the next paragraphs and their values are defined in Table 5.10.

Because the odometry readings from Pepper can be inaccurate, the provide_odom_frame and use_odometry are made variable. These determine whether Cartographer provides and uses the odometry data respectively.

The num_accumulated_range_data parameter determines how many scans should be bundled together when performing scan matching. The relatively narrow scans of Pepper could be bundled together into a bigger scan, possibly increasing the performance of the scan matcher. To test this, the parameter has been made variable.

TABLE 5.7: Cartographer custom parameters.

Parameter	Value
map_builder	MAP_BUILDER
trajectory_builder	TRAJECTORY_BUILDER
map_frame	map
tracking_frame	base_link
published_frame	odom
odom_frame	odom
provide_odom_frame	variable
publish_frame_projected_to_2d	false
use_odometry	variable
use_nav_sat	false
use_landmarks	false
num_laser_scans	1
num_multi_echo_laser_scans	0
num_subdivisions_per_laser_scan	1
num_point_clouds	0
lookup_transform_timeout_sec	0.2
submap_publish_period_sec	0.3
pose_publish_period_sec	5e-3
trajectory_publish_period_sec	30e-3
rangefinder_sampling_ratio	1.
odometry_sampling_ratio	1.
fixed_frame_pose_sampling_ratio	1.
imu_sampling_ratio	1.
landmarks_sampling_ratio	1.

TABLE 5.8: Cartographer TRAJECTORY_BUILDER_2D custom parameters.

Parameter	Value
num_accumulated_range_data	variable
num_range_data	variable
use_imu_data	false
submaps.num_range_data	variable
motion_filter.max_distance_meters	variable
motion_filter.max_angle_radians	variable

Cartographer works by using and merging submaps. The size of the submaps can be configured with the `submaps.num_range_data` parameter. More specifically, the parameter defines the number of range data before a new submap is added.

The `motion_filter.max_distance_meters` and `motion_filter.max_angle_radians` options determine after which angle or distance difference new range data gets inserted. Especially lowering the angle could overcome the error in odometry that Pepper has.

TABLE 5.9: Cartographer POSE_GRAPH custom parameters.

Parameter	Value
<code>optimization_problem.local_slam_pose_translation_weight</code>	1e2
<code>optimization_problem.local_slam_pose_rotation_weight</code>	1e1
<code>optimization_problem.odometry_translation_weight</code>	variable
<code>optimization_problem.odometry_rotation_weight</code>	variable
<code>constraint_builder.min_score</code>	variable

The `local_slam_pose_translation_weight`, `local_slam_pose_rotation_weight`, `odometry_translation_weight` and `odometry_rotation_weight` are parameters that control the weights in the calculation of the optimization of the pose of the robot. They can be used to favor the local slam or favor the odometry. Lastly, the `constraint_builder.min_score` controls the minimum score that is used in the scan matcher. This is similar to the minimum score parameter in Gmapping.

TABLE 5.10: Cartographer variable parameters.

Parameter	1	2	3	4
<code>provide_odom_frame</code>	true	false	false	false
<code>use_odometry</code>	true	true	false	true
<code>num_accumulated_range_data</code>	disabled	30	disabled	disabled
<code>num_range_data</code>	disabled	disabled	disabled	disabled
<code>max_distance_meters</code>	0.1	0.1	0.1	0.4
<code>max_angle_radians</code>	0.1	0.1	0.1	0.05
<code>odometry_translation_weight</code>	1e2	1e2	1e2	1e2
<code>odometry_rotation_weight</code>	1e10	1e10	1e10	1e10
<code>min_score</code>	0.65	disabled	disabled	disabled

TABLE 5.10: Cartographer variable parameters, continued

5	6	7	8	9
false	false	false	false	false
true	false	true	true	true
1	disabled	disabled	disabled	disabled
disabled	120	120	120	120
0.4	0.1	0.1	0.1	0.1
0.05	0.1	0.1	0.1	0.1
1e1	1e2	1e2	1e2	1e2
1e2	1e10	1e2	1e2	1e2
disabled	disabled	disabled	0.65	0.45

5.1.4 Exploration and Navigation

For the exploration and navigation, the navigation stack is configured. The package that handles the navigation and movement is `move_base`⁷. The configuration of this can be seen in Table 5.11a. The parameters of the costmap used in the navigation layer, which are defined and used in both experiments, are also shown in this table. In the navigation experiment, the variable parameters will be experimented on. These are discussed in the corresponding sections below. The results will be used in the exploration experiment.

TABLE 5.11: `move_base` parameters, continued below

(A) Main parameters	
Parameter	Value
<code>base_global_planner</code>	<code>navfn/NavfnROS</code>
<code>base_local_planner</code>	<code>base_local_planner/TrajectoryPlannerROS</code>
<code>local_costmap/width</code>	2.0
<code>local_costmap/height</code>	2.0

Navigation

Beside `move_base` performing the navigation, Pepper also needs to be localized within a map. `amcl`⁸ is used to localize Pepper within the map. Adaptive Monte Carlo Localization (`amcl`) is used for this task. It loads a grid based map, created by SLAM for example, and uses the cell data to perform the localization. Scans from the robot are matched with the cells from the map. Using a particle filter it converges to an estimation of the robot's position within the map. Throughout the navigation the robot gets continuously localized using this method. the configuration of `amcl` can be seen in Table 5.13.

The variable parameters are shown in Table 5.12. As Pepper does not have a wide field of view and cannot look sideways at the same time as straight ahead, the `acc_lim_y`, `max_vel_y`, `min_vel_y` and `y_vels` are chosen as variables to disable or minimize movement along the sideways axis. Another method to disable this movement is by changing `holonomic_robot`. Lastly, in order to reduce the chance of a collision, `inflation_radius` is also made variable.

Exploration

For the exploration part, the best scoring SLAM algorithm and navigation configuration are used. These are determined in Section 5.3.2 and 5.4.2 respectively. Moreover, `frontier_exploration`⁹ is used and the following costmap parameters are set as seen in Table 5.14. The variable parameters can be seen in Table 5.15.

⁷http://wiki.ros.org/move_base?distro=kinetic

⁸<http://wiki.ros.org/amcl?distro=kinetic>

⁹http://wiki.ros.org/frontier_exploration?distro=kinetic

TABLE 5.11: move_base parameters, continued

(B) Shared costmap parameters

Parameter	Value
robot_radius	0.30
footprint_padding	0.01
robot_base_frame	base_link
update_frequency	4.0
publish_frequency	3.0
transform_tolerance	1.0
resolution	0.05
obstacle_range	16.0
raytrace_range	16.0
static_layer	
map_topic	/map
subscribe_to_updates	true
localisation_layer	
observation_sources	depth scan laser
topic	/Pepper0/camera/depth/scan
marking	true
clearing	true
obstacles_layer	
observation_sources	Base laser
topic	/Pepper0/laser
marking	true
clearing	true
inflation_layer	
inflation_radius	variable

(C) Local costmap parameters

Parameter	Value
global_frame	map
rolling_window	true
holonomic_robot	true
plugins	
ObstacleLayer	(see Table 5.11b) localisation_layer
InflationLayer	inflation_layer

TABLE 5.11: move_base parameters, continued

(D) Global costmap parameters

Parameter	Value
global_frame	map
rolling_window	false
track_unknown_space	true
plugins	(see Table 5.11b)
StaticLayer	static_layer
ObstacleLayer	localisation_layer
InflationLayer	inflation_layer

(E) Local planner parameters

Parameter	Value
acc_lim_x	0.2
acc_lim_y	variable
acc_lim_theta	0.2
max_vel_x	0.7
min_vel_x	0.2
max_vel_y	variable
min_vel_y	variable
max_vel_theta	0.7
min_vel_theta	-0.7
min_in_place_vel_theta	0.3
escape_vel	-0.2
holonomic_robot	variable
y_vels	variable
yaw_goal_tolerance	1.0
xy_goal_tolerance	0.2
latch_xy_goal_tolerance	false
sim_time	1.8
sim_granularity	0.025
angular_sim_granularity	0.025
vx_samples	10
vtheta_samples	20
controller_frequency	20
meter_scoring	true
pdist_scale	0.6
gdist_scale	0.85
occdist_scale	0.1
heading_lookahead	0.325
heading_scoring	false
heading_scoring_timestep	0.8
dwa	true
publish_cost_grid_pc	true
global_frame_id	odom
oscillation_reset_dist	0.1
prune_plan	true

TABLE 5.12: Navigation: move_base variable parameters.

Parameter		
inflation_radius	0.05	0.2
acc_lim_y	0.0	0.2
max_vel_y	disabled	0.7
min_vel_y	disabled	0.2
holonomic_robot	true	false
y_vels	0.0	[0.2,0.7]

TABLE 5.13: acml configuration

Parameter	Value	
/scan	/Pepper0/camera/depth/scan	
min_particles	100	
max_particles	8000	
update_min_d	0.2	
update_min_a	0.523	
resample_interval	2	
transform_tolerance	1.0	
recovery_alpha_slow	0.0	
recovery_alpha_fast	0.0	
gui_publish_rate	-1.0	=disabled
laser_min_range	-1.0	=disabled
laser_max_range	-1.0	=disabled
laser_max_beams	61	
laser_z_hit	0.95	
laser_z_rand	0.05	
laser_sigma_hit	0.2	
laser_z_short	0.01	
laser_z_max	0.01	
laser_model_type	likelihood_field	
laser_likelihood_max_dist	2.0	
odom_model_type	omni-corrected	
odom_alpha1	0.01	
odom_alpha2	0.01	
odom_alpha3	0.01	
odom_alpha4	0.015	
odom_alpha5	0.01	
base_frame_id	base_footprint	
odom_frame_id	odom	
global_frame_id	map	
initial_pose_x	0.0	
initial_pose_y	0.0	
initial_pose_a	0.0	

TABLE 5.14: Frontier exploration costmap parameters

Parameter	Value
global_frame	map
rolling_window	false
track_unknown_space	true
explore_boundary	
resize_to_boundary	false
frontier_travel_point	middle
explore_clear_space	false
plugins	(see Table 5.11b)
StaticLayer	static_layer
BoundedExploreLayer	explore_boundary
ObstacleLayer	variable
InflationLayer	inflation_layer

TABLE 5.15: Exploration: move_base variable parameters.

Parameter		
inflation_radius	0.05	0.2
ObstacleLayer	localisation_layer	obstacles_layer

5.2 Expected Environment

The main scenario that inspired this thesis is the scenario where Pepper provides guidance to guests through an office building. This scenario is also used to define the expected environment. This results in the answer to the first sub-question, **SQ1**:

The expected environment for the usage of SLAM, exploration and navigation is inside of an office building. This is an environment with rich areas, such as actual workplaces with desks, computers and chairs, as well as more simple hallways and open areas. The environment does not include people or other dynamic obstacles.

Three areas were chosen, with one being a subset of another. Firstly a **hallway** area was chosen to test basic SLAM functionality. Secondly, the area around the **elevators** in an office building was chosen. The elevators area has some open space, as well as multiple hallways and a more rich seating area. The hallway area is part of the elevators area. Lastly, a section of a **work floor** was chosen. This area has multiple desks, chairs and other objects. This area was chosen to test a more difficult and rich area. These areas are discussed in Section 5.3.1 and the plans are visible in Figures 5.1, 5.2 and 5.3.

5.3 SLAM performance

The following sections cover the experiment of the SLAM algorithms on Pepper. The design, objectives and results are discussed.

5.3.1 Design & Objective

The main objective of this experiment is to answer **SQ2**: How does Pepper perform using SLAM algorithms, with regard to an expected environment?

To answer **SQ2**, sensor data from the expected environment was collected. To account for changes in driving style, and thus differences in sensor data, multiple recordings of the **Elevators scene** were made. Eventually, the recorded scenes can be seen in Table 5.16.

TABLE 5.16: Recorded scenes.

Name	Length (sec)
Hallway single	102
Hallway double	180
Elevators normal	601
Elevators extended	1833
Elevators variation	1445
Work floor	1121

The first environment, the hallway, is meant to test basic SLAM performance in a simple corridor. **Hallway single** is a simple hallway scene with a dead end at one side. In this scene Pepper is driven one time up and down the hallway in a somewhat straight line. **Hallway double** is the same hallway, but Pepper drives down the hallway two times, possibly fixing initial errors in the created map. The ground truth and trajectories for the hallway scenes can be seen in Figure 5.1.

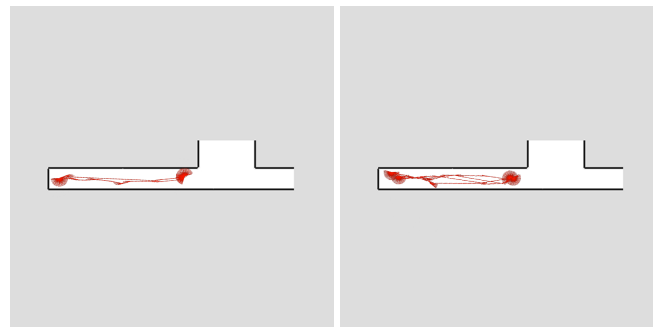
Elevators normal, **elevators extended** and **elevators variation** are scenes in the elevator area of an office building, it's ground truth and trajectories can be found in Figure 5.2. In the scene **elevators normal** Pepper drives around all parts of the area once, not focusing on anything in particular. Pepper drives in straight lines, covering the whole area efficiently. In the scene **elevators extended**, Pepper is being driven around the area such that all parts are covered by the sensor. This means that Pepper zigzags the whole area, but does not focus on specific parts. In the third elevator scene (**elevators variation**) Pepper again drives in the elevators area. This time, Pepper's sensors are specifically pointed to interesting areas, such as corners, rich objects and walls. This includes Pepper moving sideways in order to observe walls, objects and details more extensively.

In the last scene, **work floor**, Pepper is driven around on the actual work floor of an office building. There is no particular focus on specific object, but the trajectory was chosen such that all parts and object are observed by the sensor. Lastly, the ground truth and trajectories for the work floor can be seen in Figure 5.3.

Recording the data

For each of the scenes at least the following topics are recorded using the `roscop record` command:

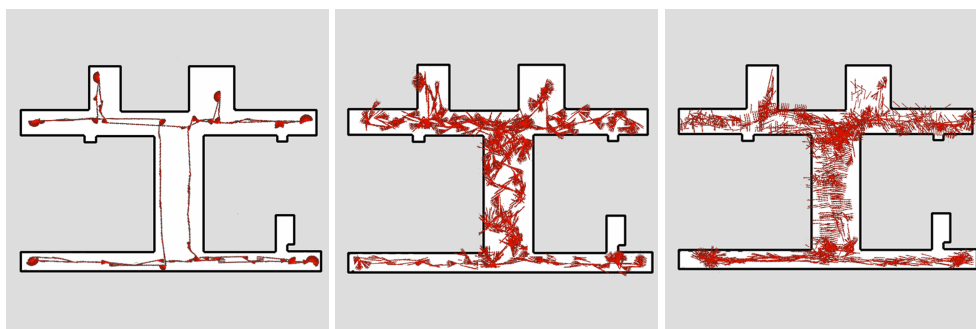
- `/Pepper0/camera/depth/camera_info`: Technical information about the depth camera. I.e. intrinsic information.
- `/Pepper0/camera/depth/image_raw`: Images of the depth camera.



(A) Hallway

(B) Hallway double

FIGURE 5.1: Ground truth hallway scenes



(A) Elevators normal

(B) Elevators extended

(C) Elevators variation

FIGURE 5.2: Ground truth elevator scenes

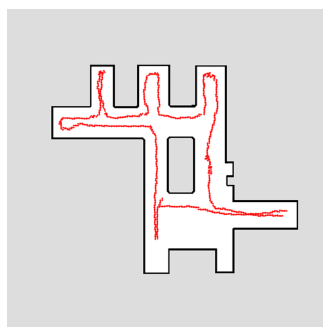


FIGURE 5.3: Ground truth work floor scene

- `/Pepper0/odom`: Odometry of Pepper.
- `/joint_states`: The states of the joints.
- `/tf`: The transform data of all of the frames.

Map creation

For the eventual map creation, the following steps are executed. Firstly, the ROS master node is started. Secondly, ROS on Pepper is started so it publishes its data. Then the depth image to point cloud to laser scan is started. After these are all fully started, the SLAM algorithm is run. The rosbag(s) of a scene is then played and afterwards the map is saved. This process can be easily automated for each of the scenes and configurations.

Map evaluation

The last step of the experiment is the evaluation of the created maps. As all scenes have a ground truth map the mean squared error is used as main performance metric. The other metrics are being calculated and are visible in Appendix B.

For the ground truth metric the images need to be aligned to the ground truth image. This is done by using the OpenCV functions `estimateRigidTransform` and `warpAffine`. After the alignment, the distance from each occupied cell of the ground truth to the nearest cell of the computed map is determined using OpenCV's `knnsearch`. The sum of these distances is divided by the number of occupied cells, creating the normalized error metric.

The other metrics are the proportion of occupied cells, amount of corners and amount of enclosed areas. The calculation of the proportion of occupied cells is trivial and can be counted by counting white pixels in the image. The amount of corners can be extracted by using OpenCV's `cornerHarris` function. The enclosed areas are counted using OpenCV's `contours`. They can be found using `findContours` and can be filtered on color, area and more.

5.3.2 Results

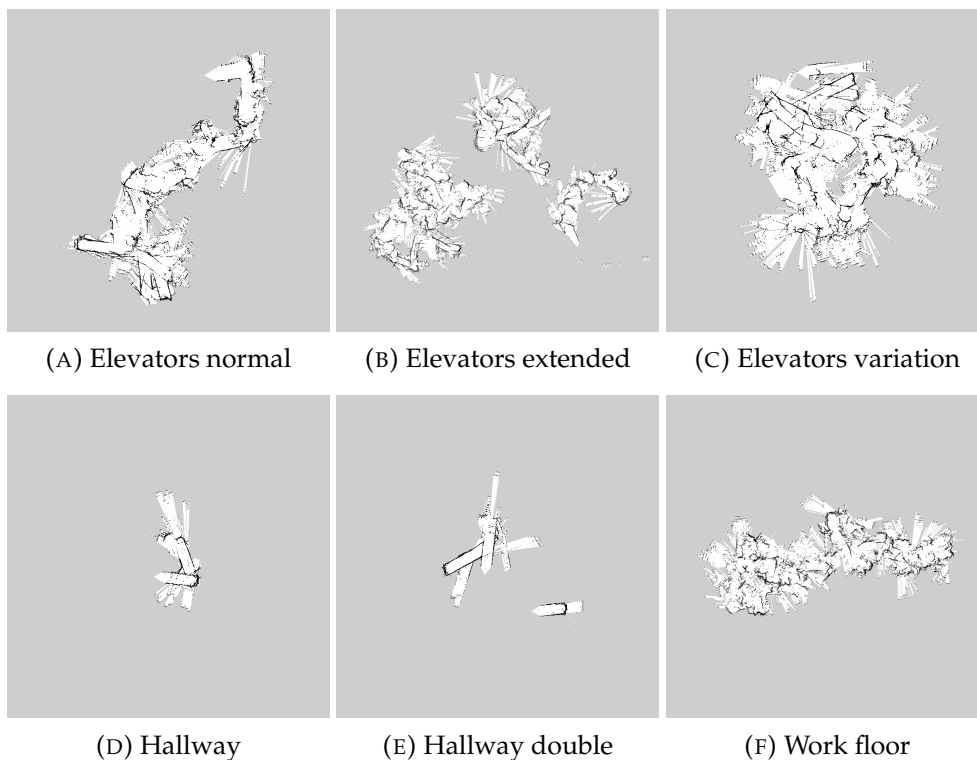
This section covers the results of the SLAM experiment. First, the results of the Hector Mapping and Cartographer are discussed, as they are very different than the Gmapping results. The Gmapping results are shown after. For each scene, the best scoring configurations and other interesting configurations are shown. Discussion of the results is presented in Section 6. All created maps can be found in the accompanying materials.

When the term 'usable' is used with regard to the created maps, it means that it is a map that covers at least most of the free space and obstacles, so that a robot can use it to navigate to all areas within said map.

Hector Mapping

First of all, using the configurations given in Section 5.1.3, Hector Mapping never produced a usable map. Some maps that were produced can be seen in Figure 5.4 and 5.5. The variation of the `map_update_angle_thresh` did not impact the results significantly and most of the times not at all. The variation of the `map_multi_res_levels` parameter is visible in the two figures. Given that the resulting maps are not usable at all, the metrics that should be calculated are not representative and thus not calculated and used.

FIGURE 5.4: Hector mapping maps (`map_multi_res_levels = 2`)



Cartographer

Cartographer, configured using the configurations shown in Table 5.10, also did not give usable maps. Again, the metrics that should be calculated are not representative compared with usable maps and are thus not calculated. The maps that looked most like the ground truth images can be seen in Figure 5.6 and 5.7. The configuration of those maps can be seen in Table 5.17.

Gmapping

Gmapping did produce usable maps. For each scene, the best scoring maps and their configurations and metrics can be seen in the tables of this section. The best scoring created maps can also be seen in the Figure 5.9. All results can be seen in Appendix B and accompanying maps in the complementary materials. If a map was produced that was not usable, the scores were not calculated and these maps are not

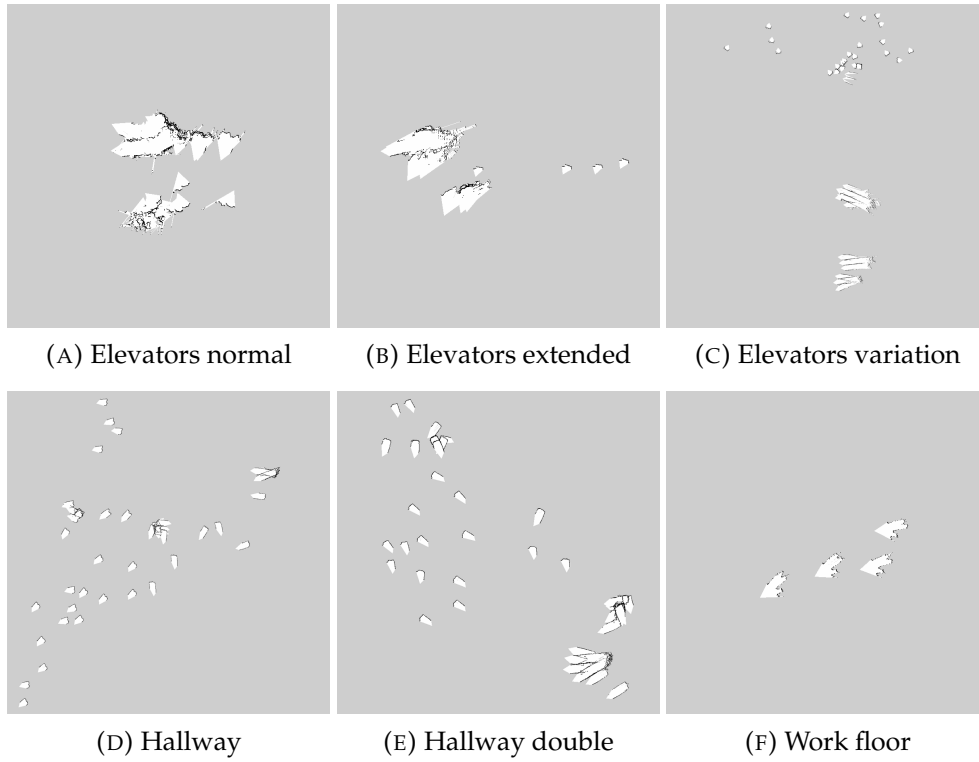
FIGURE 5.5: Hector mapping maps (`map_multi_res_levels = 6`)

FIGURE 5.6: Cartographer maps (maximum laser distance = 5)

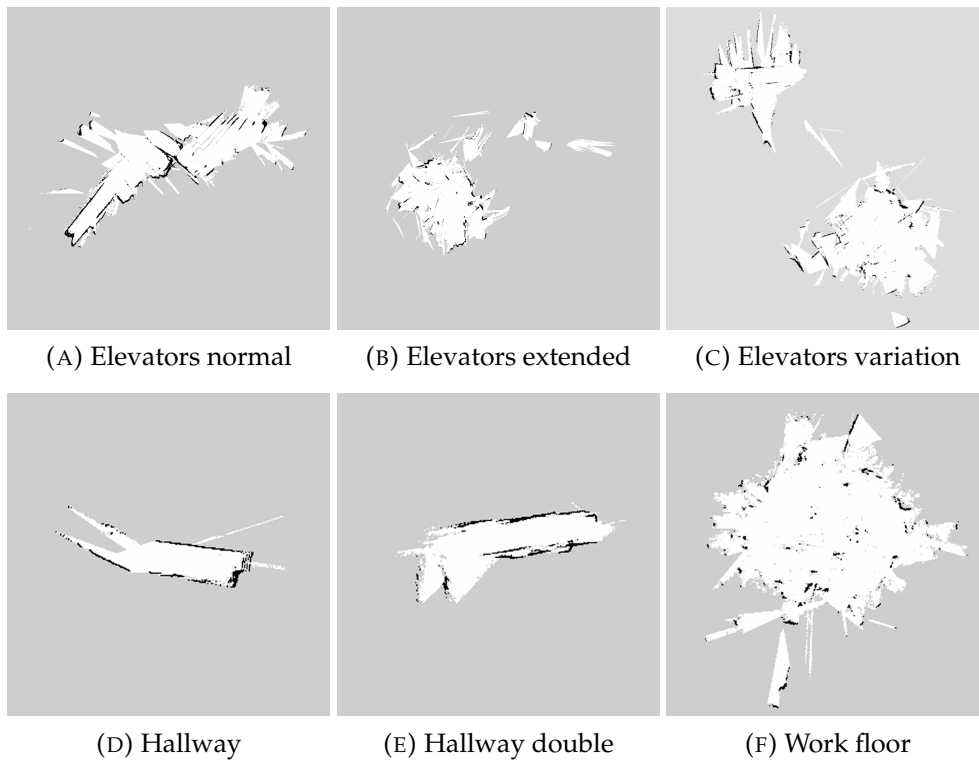


FIGURE 5.7: Cartographer maps (maximum laser distance = 12)

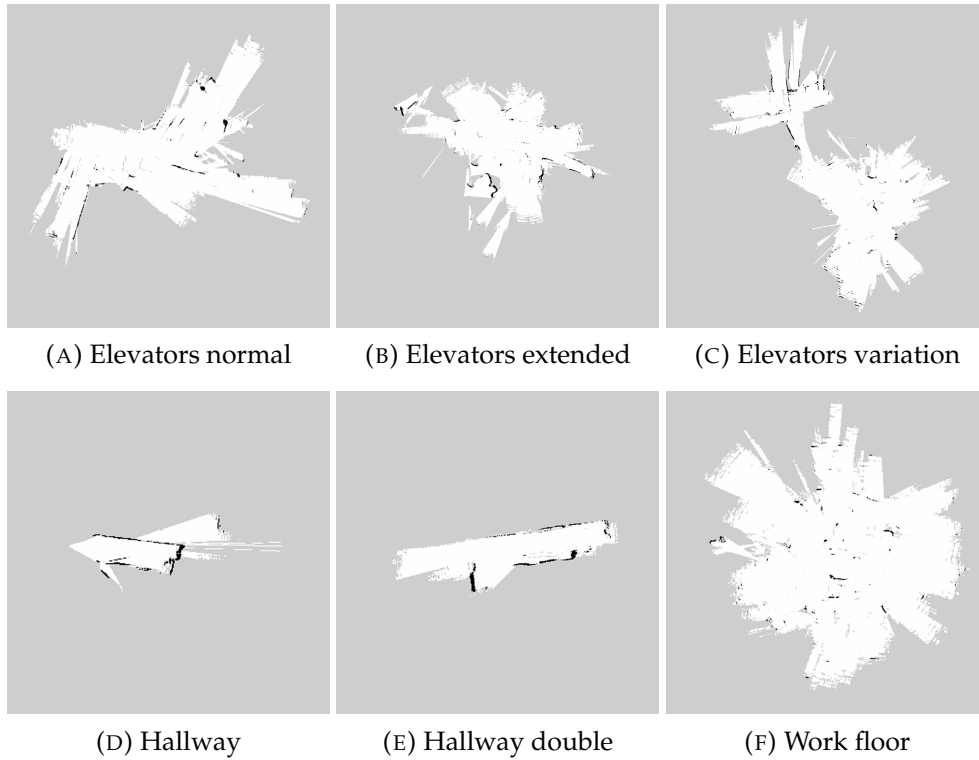


TABLE 5.17: Cartographer best performing parameters.

Parameter	Value
provide_odom_frame	false
use_odometry	true
num_accumulated_range_data	disabled
num_range_data	120
max_distance_meters	0.1
max_angle_radians	0.1
odometry_translation_weight	1e2
odometry_rotation_weight	1e2
min_score	disabled

included in the top scores. Some examples of the non-usable maps can be seen in Figure 5.8.

FIGURE 5.8: Gmapping: unusable maps

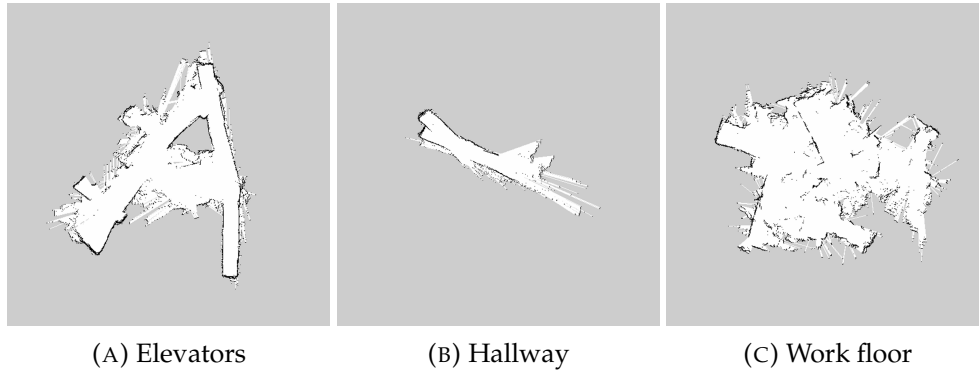
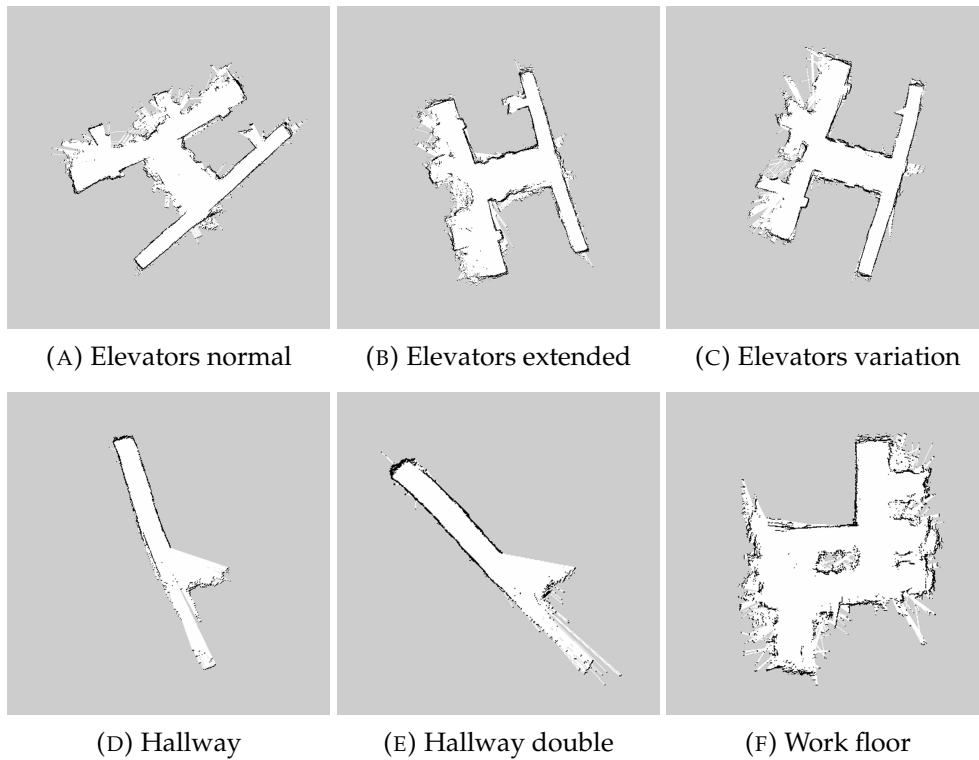


FIGURE 5.9: Gmapping: best scoring maps



Best performance

From the tested configurations, Gmapping clearly performs best on the recorded data sets. There is no clear best configuration, but the configuration chosen as best can be seen in Table 5.24 and is explained in the next paragraph.

For all of the parameters, the results show that there is no real optimal setup for all scenes. *srr*, *srt* and *stt* are set to 0.1, 0.1 and 0.02 respectively, as most of the best performing configurations use this value. *str* is set to 0, as this parameter is intended to compensate for the error in odometry and the results show that it is a

TABLE 5.18: Gmapping results for scene **work floor**

Nr	MSE	srr	str	srt	stt	minimum Score	Max laser distance
1	1.9851289927	0	0	0.1	0.2	200	5
2	2.1342394757	0.1	0.1	0.1	0	100	5
3	2.3400592811	0.1	0.1	0	0.02	100	5
4	2.4718676991	0.1	0.1	0	0.02	200	5
5	2.5780578078	0.1	0	0	0.02	100	5
6	2.616063256	0.1	0.1	0.1	0	200	5
7	2.6475122163	0.1	0	0	0.02	200	12
8	2.7356917548	0.1	0	0	0.2	100	12
9	2.747862063	0.1	0.1	0.1	0.02	100	5
10	2.9733542664	0	0.1	0	0.2	200	5
11	2.9811623728	0.1	0.1	0	0.02	100	12
12	2.9877237415	0	0	0	0.2	200	5
13	3.0724878799	0.1	0	0.1	0	200	5
14	3.1876565679	0.1	0	0	0	200	5
15	3.21395944	0.1	0.1	0	0.02	200	12

TABLE 5.19: Gmapping results for scene **hallway**

Nr	MSE	srr	str	srt	stt	minimum Score	Max laser distance
1	7.6644623547	0.1	0.1	0.1	0.02	200	5
2	7.8059689115	0	0	0	0	100	12
3	7.808512965	0.1	0	0.1	0.02	100	5
4	7.9185673136	0	0	0	0.2	100	12
5	7.9498310851	0.1	0	0	0	100	5
6	8.1921640383	0.1	0.1	0	0	100	5
7	8.8569985418	0	0.1	0.1	0.2	100	5
8	9.2448147082	0	0	0	0	100	5
9	9.5086761651	0	0.1	0.1	0.02	100	12
10	9.5578613795	0.1	0.1	0.1	0.2	100	12
11	9.6076166604	0.1	0.1	0	0.2	100	12
12	9.6893308644	0	0	0	0	200	5
13	9.795294606	0	0	0.1	0	100	5
14	9.8089427664	0.1	0	0.1	0	200	5
15	9.8125168722	0	0	0.1	0	200	12

TABLE 5.20: Gmapping results for scene **double hallway**

Nr	MSE	srr	str	srt	stt	minimum Score	Max laser distance
1	6.5178467135	0	0	0	0.02	100	12
2	6.5612825029	0.1	0.1	0	0	100	12
3	6.6517707285	0.1	0.1	0.1	0.02	100	5
4	6.8060895682	0	0.1	0.1	0	100	12
5	6.8075969642	0.1	0	0	0.02	100	12
6	6.8304017523	0.1	0	0.1	0.2	100	5
7	7.3791336171	0	0.1	0.1	0.02	100	12
8	7.5249289809	0	0	0	0	100	5
9	7.5604638467	0.1	0.1	0	0.2	100	5
10	7.7655184735	0.1	0	0.1	0	200	12
11	8.0825848987	0	0	0	0.02	100	5
12	8.3299362059	0.1	0.1	0	0.02	100	12
13	8.4093368438	0	0	0.1	0.02	100	12
14	8.434494163	0	0.1	0.1	0.02	100	5
15	8.6281870228	0.1	0	0	0	100	5

TABLE 5.21: Gmapping results for scene **elevators normal**

Nr	MSE	srr	str	srt	stt	minimum Score	Max laser distance
1	14.93401111	0	0	0	0	100	5

TABLE 5.22: Gmapping results for scene **elevators extended**

Nr	MSE	srr	str	srt	stt	minimum Score	Max laser distance
1	13.0969277382	0.1	0	0.1	0.02	200	12
2	13.8052665862	0.1	0.1	0.1	0	200	12
3	14.8201897302	0.1	0	0.1	0.2	200	12
4	16.2308389779	0.1	0	0	0	200	5
5	17.4559488332	0.1	0.1	0	0.2	200	5
6	18.1427536558	0.1	0.1	0	0	200	12
7	19.2116838646	0	0	0.1	0.02	200	5
8	19.5914200418	0.1	0	0	0.02	200	5
9	20.0696447364	0.1	0	0.1	0.02	200	5
10	20.4828003879	0	0	0.1	0	200	12
11	21.583124839	0	0.1	0.1	0.02	200	5
12	26.0867660599	0	0.1	0	0.02	200	12
13	26.6165201703	0	0	0.1	0.02	200	12
14	32.9637704572	0.1	0.1	0	0.02	200	5
15	34.9151411013	0	0.1	0.1	0.2	100	5

TABLE 5.23: Gmapping results for scene **elevators variation**

Nr	MSE	srr	str	srt	stt	minimum Score	Max laser distance
1	7.2019722004	0	0.1	0.1	0.02	100	5
2	8.1373130354	0.1	0	0.1	0.2	100	5
3	8.4643356229	0.1	0	0.1	0.2	200	5
4	9.4121481799	0	0.1	0	0.2	100	12
5	9.9935039926	0.1	0.1	0.1	0.2	100	5
6	10.3713939386	0.1	0	0.1	0.2	100	12
7	11.1217625028	0	0	0	0.2	200	12
8	11.6552504276	0.1	0.1	0.1	0	100	12
9	11.7915281038	0	0	0.1	0.2	200	12
10	11.9448849922	0.1	0.1	0.1	0.02	100	12
11	12.0295315446	0	0	0.1	0.2	100	5
12	12.1085567482	0	0	0.1	0.2	100	12
13	12.1553100255	0.1	0	0.1	0	100	12
14	12.3056993844	0	0	0.1	0	200	5
15	12.3857561254	0	0	0	0.02	100	5

TABLE 5.24: Chosen best performing configuration

Parameter	Value
srr	0.1
str	0
srt	0.1
stt	0.02
minimumScore	100
Max laser distance	12

viable value. `minimumScore` is set to 100. The only scenario that clearly favors 200 is the variation of the elevators scene. In this scene Pepper is driving the scene in short lines with a lot of rotation. This is not really expected when exploring, so the value 100 is chosen as the other scenes favor that value. `Max laser distance` is set to 12 meters, to maximize what Pepper sees. The artifacts on the floor are mostly fixed by the minimum height of the point cloud to laser scan algorithm.

5.4 Navigation performance

5.4.1 Design & Objective

The third experiment is designed to answer **SQ4**: What navigation technique performs best on Pepper, given a created map and desired environment? This experiment is done before the exploration experiment, as the exploration also uses the `move_base` node for its navigation. The results of the SLAM experiment are used as map input for this experiment and the resulting best parameters of this experiment are used in the exploration experiment.

Again, for the environment the area from the elevator scenes is used (Figure 5.2). Within this area navigational starting points and goals are used, using varying degrees of difficulty. The difficulty ranges from straight lines within the open areas, to goals that require taking multiple corners and movement within the narrow corridors.

The setup of the experiment is as defined as: First the ROS master node, ROS on Pepper and the depth-to-laser node are started. Afterwards, `amcl` is started with a decent scoring map from Section 5.3. For `amcl`, used to localize Pepper within the environment, Pepper needs to be driven around manually for a short time. After Pepper is localized the `move_base` node is started, providing the actual navigation. Pepper is driven to the starting location and the navigational goal is set using `rviz`. The navigation starts and results in either Pepper reaching the goal, getting stuck in the environment or not being able to calculate a route. Reaching the goal, collisions and the trajectory Pepper uses are used to determine performance.

The configurations of the fixed and variable navigation parameters used can be seen in Table 5.11.

5.4.2 Results

First of all, navigation in straight lines within a open area of the used map work fine for all tested configurations. Figure 5.10 shows the trajectory and success of such a navigational task.

When using the parameters that enable the holonomic properties of the robot (see Table 5.25), navigation that involves corners fails often. The robot gets stuck around corners, complex objects and in narrow passages and cannot successfully complete the navigation.

Setting the parameters to the values that disable the sideways velocities helps improve the success rate of the navigation. Increasing the inflation of the obstacles also decreases the risk of a collision. With the configuration as seen in Table 5.26, the

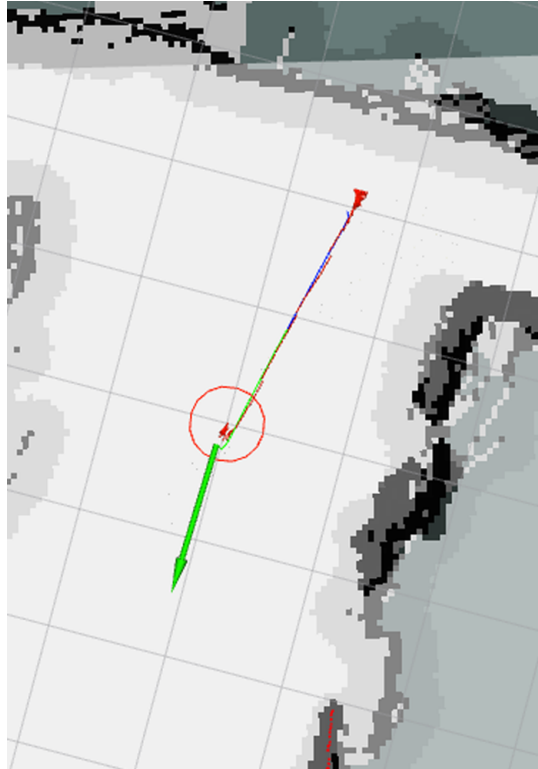


FIGURE 5.10: Navigation in a straight line

TABLE 5.25: Navigation: failing parameters

Parameter	Value
inflation_radius	0.05
acc_lim_y	0.2
max_vel_y	0.7
min_vel_y	0.2
holonomic_robot	true
y_vels	[0.2,0.7]

robot can complete multiple sorts of navigational tasks. The results of these tasks can be seen in Figure 5.11. On these figures, the poses of the robot are drawn on top of the map. In Figure 5.11a, Pepper is given the task to navigate from the bottom left to the top left portion of the map. In Figure 5.11b, Pepper is given the task to navigate from the top left to the bottom left portion of the map.

While the robot reaches the goals successfully, it is not optimal. In both cases, the narrow passage at the bottom of the map is still a difficult spot. The robot gets stuck in the narrow passage temporarily, but this time the recovery behaviour corrects it. The recovery behaviour consists of rotating 360 degrees so that the localization and obstacle sensor data gets updated. This is visible in the poses of the robot in the figures. After the recovery, the robot is able to improve it's localization and continues it's path towards the goal.

TABLE 5.26: Navigation: best parameters

Parameter	
inflation_radius	0.2
acc_lim_y	0.0
max_vel_y	disabled
min_vel_y	disabled
holonomic_robot	false
y_vels	0.0

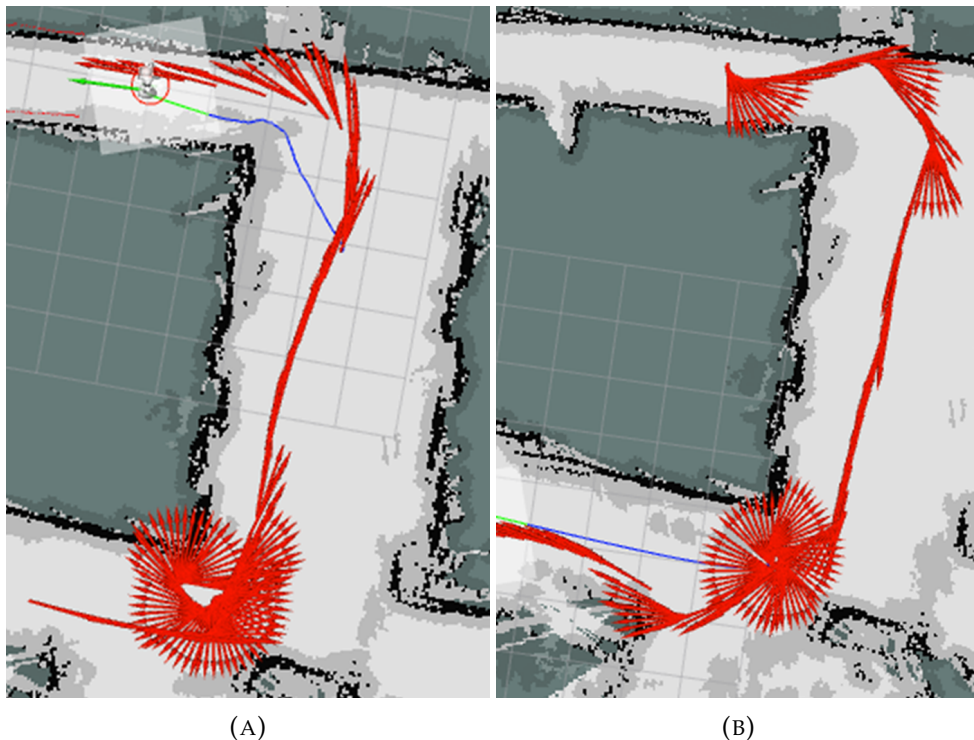


FIGURE 5.11: Navigation success

Nevertheless, the navigational tasks are not always completed successfully. Especially when the localization of the robot is not accurate, the robot can get stuck. An example of this case, Pepper was given the task to navigate from the top right to the bottom left of the map. Figure 5.12 shows that the robot is not localized properly. The localization thinks the robot is almost at the corner, while some of the sensor

data shows the wall continues for a little bit. It tries to maneuver around the corner too early and gets too close to the wall. Recovery behaviour does not always solve this problem and sometimes the emergency brake of NAOqi kicks in, disabling all movement except moving backwards.

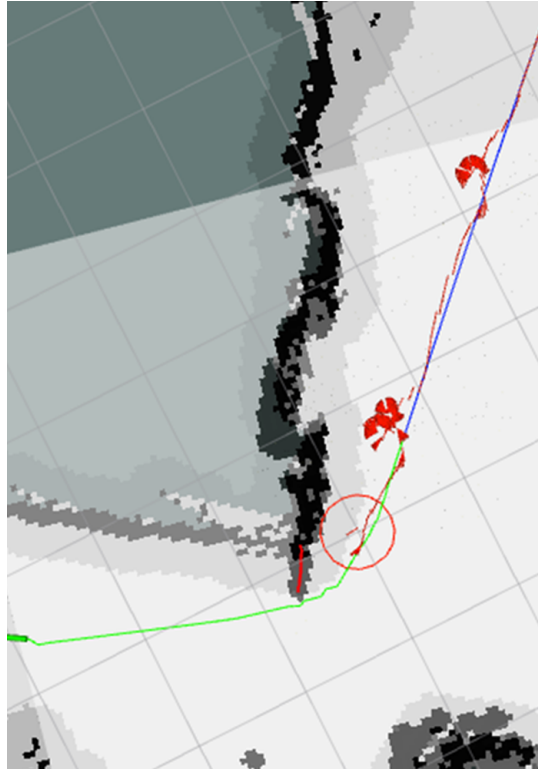


FIGURE 5.12: Navigation failure

Best performance

While definitely not perfect, the best performing configuration is seen in Table 5.26. The increased inflation radius decreases the chance of a collision with the obstacles as seen in the map. Furthermore, disabling all holonomic functions such as sideways movement creates trajectories less likely to collide and eliminates movement in the directions the robot does not have sensor data from.

5.5 Exploration performance

5.5.1 Design & Objective

The main objective to this experiment is to answer **SQ3**: How does a frontier exploration of a whole unknown environment with Pepper perform, while creating a map using a ROS SLAM algorithm?

For the unknown environment, the area from the elevator scenes is used (Figure 5.2), according to the answer of **SQ1**. Moreover, the best scoring parameters of the algorithms are used, as discussed in Section 5.3.2 and 5.4.2. Lastly Pepper is positioned

in the middle of the wide hallway, between the two smaller hallways. The area to explore is set so that it covers at least the whole area.

The setup of the experiment consists of the following steps. First the ROS master node, then ROS on Pepper, the depth to laser and SLAM algorithm are started. For this experiment, the best performing SLAM configuration is used (see Table 5.24). Furthermore, the `move_base` node is started, using the configuration as seen in Table 5.26. Lastly the frontier exploration node is started and the search area is set using `rviz`. The area set is made big enough so that the whole map fits within that area. The starting position for the frontier exploration is set 3 meters in front of Pepper. After setting the starting position, the exploration starts and stops when either Pepper cannot move anymore or when it cannot find any unexplored space. In the end, the resulting map can be saved and compared to the ground truth map.

The configurations used can be seen in Table 5.14.

5.5.2 Results

All configurations did not result in a complete map of the environment. The best result was created with an increased inflation radius, and with using the 3D depth sensor as the obstacle layer and is visible in Figure 5.13a. The map created with the increased inflation radius but with the lasers of Pepper as the obstacle layer is visible in Figure 5.13b. For every experiment run, the robot explores most of the open space fine, but eventually gets stuck in a position it cannot get itself out. This always happens in an area with more complex objects and in narrow corridors. An example of such a location can be see in Figure 5.14. In the area of the red circle, the frontier exploration has set a frontier and tries to navigate to that point. However, this area has objects in it that can be difficult to detect with the hardware of Pepper. This results in the robot trying to navigate to a frontier that not really should be a frontier.

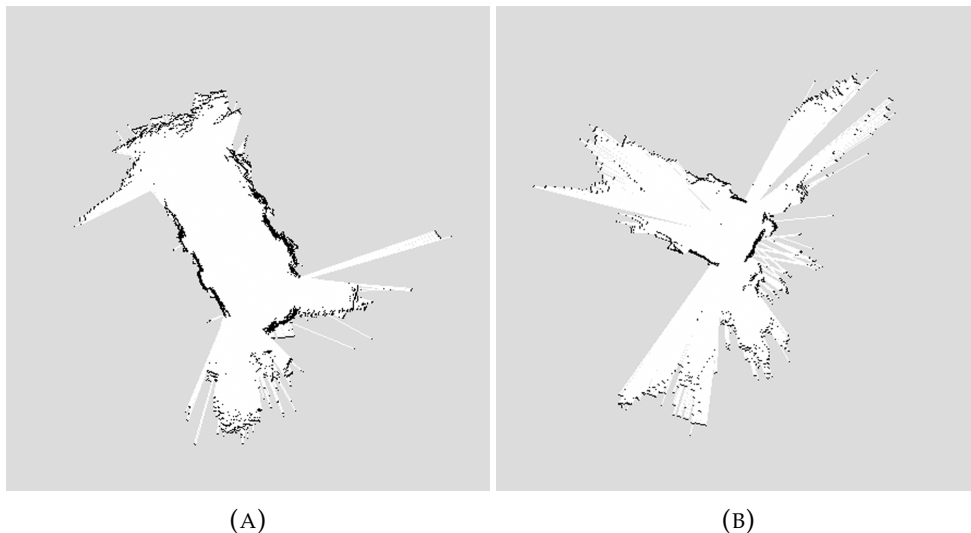


FIGURE 5.13: Exploration results

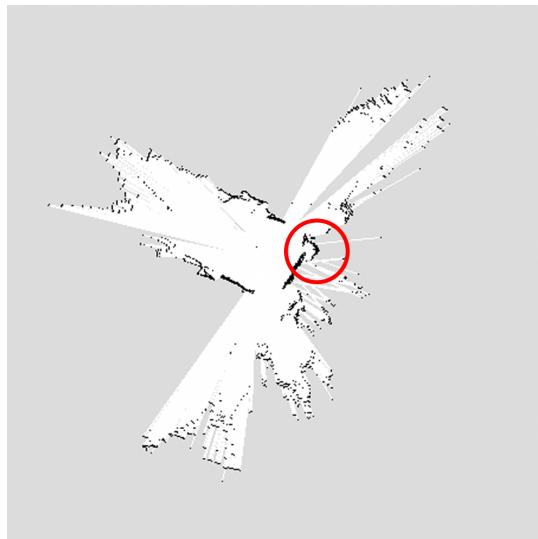


FIGURE 5.14: Exploration fail

Chapter 6

Conclusion

This chapter draws conclusions about the results as seen in Chapter 5 and provides discussion of potential improvements and future work.

6.1 Results

First of all, the expected environment, and thus the answer to **SQ1**, was determined as follows: The expected environment for the usage of SLAM, exploration and navigation is inside of an office building. More specifically, this is an environment with rich areas, such as actual workplaces with desks, computers and chairs, as well as more simple hallways and open areas. The environment does not include people or other dynamic obstacles. When talked about an expected environment in the next paragraphs, it means that it is an environment conform the answer of **SQ1**.

Regarding the Gmapping results: using the 3D depth sensor from Pepper as a laser scan results in decent to really accurate maps. The difference in results of the three elevator scenes shows that it does matter how Pepper moves within an area. When looking distinctively at interesting features such as corners and obstacles, the resulting maps can be much more detailed than when Pepper does not focus it's sensor. Revisiting parts of the areas can also help improve the quality, but it is not necessary. Tweaking parameters of the algorithm(s) was needed to get the best results possible, as the odometry information of Pepper is not accurate. Especially the parameters that influence the weights of the odometry data are useful in this case and can overcome the error in odometry by preferring the scan matcher.

In the end the resulting maps are accurate. The only exception are the objects of some type of material that the 3D sensor simply cannot detect. Examples in the environment of an office building can be fully glass doors or walls. **SQ2** was defined as: 'How does Pepper perform using SLAM algorithms, with regard to an expected environment?'. The conclusion and the answer to **SQ2** is: Pepper creates decent to accurate maps of unknown environments, with the exception being certain non-detectable materials.

While Gmapping creates accurate maps, Hector Mapping and Cartographer can not create full maps with the used recordings. Hector Mappings uses an Extended Kalman Filter (EKF) as apposed to the particle filter in Gmapping and tries to rely on high frequency and accurate laser scans. Mainly due to the quality of the depth sensor and error in odometry data, the recordings could not provide such accurate information. Cartographer also does not use a particle filter. Instead, it relies on correctly created sub-maps that get scan matched into one bigger map. However, with

the configurations and recordings used, these sub-maps were not accurate enough to create complete maps reliably. The conclusion here is that the particle filter used in Gmapping, handles the data from Pepper better than the alternative methods.

With regard to the navigation capabilities, the results from Pepper fluctuated. Simple tasks such as straight line navigation are no problem at all, but in certain areas Pepper experiences problems. The main problem here is the quality of the localization. In some rich areas and narrow corridors, Pepper localizes itself incorrectly, making the navigation fail as well. Limiting the movement of Pepper in the side-ways directions help, but do not solve all problems.

The relevant sub-question **SQ3** was: 'How does Pepper perform navigating in a SLAM created map of an expected environment?' and is answered as follows: Pepper fluctuates in performance when navigating in a SLAM created map. Pepper reaches most navigational goals, but there is a chance Pepper can get stuck when the localization fails to locate Pepper correctly.

Lastly, **SQ4** was defined as 'How does a frontier exploration of a whole unknown environment with Pepper perform, while creating a map using a ROS SLAM algorithm?'. The results of the exploration experiment show that the data from the 3D sensor is not enough to perform a full exploration in an office environment. Open areas with simple structures get explored, but Pepper gets stuck in the more difficult areas. This is partly due to the sometimes inaccurate localization and partly due to the limited field of view and range of the 3D sensor. To answer **SQ4**: Pepper can explore simple and open areas, but fails to explore more difficult areas.

With the experiments answering all of the sub-questions, the main research question can be answered. **RQ** is 'Is Pepper suitable for autonomous exploration, SLAM and navigation using ROS without the usage of external sensors?'. The answer is: No, Pepper is not suitable for autonomous exploration, SLAM and navigation in it's current state, using the existing algorithms used in this thesis.

Pepper can create usable maps using SLAM, but it fails to navigate without any collisions and fails to explore the whole unknown area. As defined in Section 5.3.2, a usable map covers at least most of the free space and obstacles, so that a robot can use it to navigate to all areas within said map. The failure to localize Pepper correctly is the most direct cause of the failures with the navigation and localization. An obvious part of this problem is the actual sensor itself, but it is a combination of the sensor and the usage of that sensor within the algorithms.

6.2 Future work

While the conclusion was made that Pepper is not suitable for autonomous exploration, SLAM and navigation using the mentioned algorithms, there is a lot of room for improvements.

First of all, while Gmapping creates usable maps, it is still dependent on the driving style of Pepper. The driving style determines what the sensors of Pepper see, as the field of view of the camera is quite narrow. A possibility to improve this, is to move the head of Pepper while driving. This can increase the total field of view drastically and could remove the need for certain driving styles. The challenge here would be

to create an algorithm that moves the head and the body of Pepper while moving, such that the resulting map is optimal.

Another problem Pepper has is the error in odometry. While some SLAM algorithms compensate for these kinds of errors, better odometry data results in better maps. One could use an alternative external sensor to provide accurate odometry for the SLAM algorithms, improving the created maps.

One of the big problems lies in the quality of the 3D sensor. While it is still good enough to perform Gmapping, the shortcomings cause failures in the navigation and exploration algorithm. While this thesis focuses only on the hardware Pepper already has, using a better sensor could prevent a lot of these failures. There are a lot of options regarding Li-DAR systems. Most of these sensors have a wide field of view and are significantly more accurate in distance readings than the 3D sensor. Besides the improvement in navigation and exploration, an improved laser scan also will improve the SLAM results. An interesting question is what sensor would be needed to let Pepper explore and navigate autonomously and successfully.

Lastly, one could create algorithms specifically designed for Pepper. The localization of Pepper is the most important factor that needs improvement as the 3D sensor alone is not good enough in the standard `amcl` algorithm. The sensor does not give accurate enough data for the scan matching to work properly. If one wants to create a localization algorithm for Pepper, there needs to be a focus on using the sensor data most effectively. Just like a possible improvement in a SLAM algorithm, the mobility of the upper body could be used to gather more data from the sensor. It could be an interesting challenge to use the mobility such that the localization works in more or even all cases.

Besides the localization, the navigation could also be improved with custom algorithms. The main problem with the navigation is the previously mentioned localization. Besides the options to improve localization, one could focus on adding more of Pepper's sensors to improve navigation results. The RGB camera could be used for this. Especially the possibility of using object detection within the algorithms is interesting. This object detection can be used to improve obstacle detection and with that the performance of the navigation.

In conclusion, there are a lot of options to improve capabilities of Pepper with regard to SLAM, navigation and exploration. However, the difficulty and time invested in the creation of those custom solutions need to be considered.

Appendix A

ROS Setup

This appendix lists the steps needed to setup the ROS environment for Pepper ¹. First the ROS installation details are given and then the cross compilation is covered. All text underlined is a link that refers to more information or a download location.

A.1 ROS

Prerequisites:

- [Ubuntu 16.04 LTS](#)
- [ROS Kinetic \(Desktop-Full\)](#)
- [NAOqi C++ SDK](#)
 - [SDK and Cross Toolchain download](#)
- [Docker CE](#)
- [Linux packages: gcc-multilib libc6-dev libc6-i386](#) for cross compiling 32 bit software in Linux 64 bit.

A.2 Cross compilation

Prerequisites are all from Section A.1 and the following:

- A working internet connection
- [The ros2_pepper repository](#)

Steps for cross compiling:

- Add the environment variables to all of the following `.sh` scripts that are run. If not done correctly, the line 17: `ALDE_CTC_CROSS: unbound variable` is thrown. Example:

```
export AL_DIR=/home/USERNAME/naoqi-sdk
export ALDE_CTC_CROSS=/home/USERNAME/ctc-linux64-atom-2.5.2.74
```

- Prepare the requirements:

¹<https://github.com/Robert16296/Pepper-Dev>

```
sudo ./prepare_requirements_ros1.sh
```

- Download and setup dependancies:

```
sudo ./build_ros1_dependencies.sh
```

- Pull all repositories needed:

```
sudo ./get_repos_ros1.sh
```

- **Optional:**

- Edit the `naoqi_driver` source code. The camera distortion on the depth camera is disabled by default. Uncomment the lines that define the distortion model in `camera_info_definitions.hpp` and add `.convert_to_container<std::vector<double>> >()`; to the distortion model to enable them.

- Build ROS for Pepper:

```
sudo ./build_ros1.sh
```

- **Optional:**

- Edit `.ros-root/ros1_inst/share/naoqi_driver/share/boot_config.json` file to edit the setup of the sensors of Pepper. Topics can be enabled, disabled or edited. My version can be used with the following command:

```
scp boot_config.json nao@pepper.local:.ros-root/ros1_inst/share/naoqi_driver/share/boot_config.json
```

- Transfer the ROS folder to Pepper (if `pepper.local` does not work, use the IP address of Pepper):

```
scp -r .ros-root nao@pepper.local:.ros-root
```

A.3 Running ROS

Not required but used:

- Add the following to `~/ .bashrc`. Change `USED_NETWORK_INTERFACE` to the used network interface (i.e. `wlan0`):

```
export ROS_INTERFACE=USED_NETWORK_INTERFACE
export LOCAL_IP='ifconfig $ROS_INTERFACE | grep "inet_addr" | cut -d ':' -f 2 | cut -d ' ' -f 1'
export ROS_MASTER_IP=$LOCAL_IP
export ROS_MASTER_URI=http://$ROS_MASTER_IP:11311
export ROS_HOSTNAME=$LOCAL_IP
export ROS_IP=$LOCAL_IP
```

- Add the following to `.ros-root/setup_ros1_pepper.bash` on Pepper. Change `USED_NETWORK_INTERFACE` to the used network interface (i.e. `wlan0`) and `ROS_MASTER_IP_FROM_PC` to the IP address of the master node:

```
export ROS_INTERFACE=USED_NETWORK_INTERFACE
export LOCAL_IP='ifconfig $ROS_INTERFACE | grep "inet_"
| cut -d ' ' -f 10'
export ROS_MASTER_IP=ROS_MASTER_IP_FROM_PC
export ROS_MASTER_URI=http://$ROS_MASTER_IP:11311
export ROS_HOSTNAME=$LOCAL_IP
export ROS_IP=$LOCAL_IP
export NAMESPACE=Pepper0

alias pepper_run_ros='roslaunch naoqi_driver
naoqi_driver.launch roscore_ip:=$ROS_MASTER_IP
network_interface:=$ROS_INTERFACE
namespace:=$NAMESPACE'
```

Starting ROS:

- Main pc:

```
roscore
```

- Pepper (ssh):

```
source .ros-root/setup_ros1_pepper.bash
pepper_run_ros
```

You should see the topics being published. This can be checked by running:

```
rostopic list
```


Appendix B

Results

This appendix shows all results with metrics were calculated. All are from Gmapping.

TABLE B.1: Gmapping: Workfloor results

Nr	MSE	srr	str	srt	stt	Score	minimum	Max laser distance	Non zero ratio	Enclosed areas
1	1.985128993	0	0	0.1	0.2	200	200	5	0.0006422996521	2
2	2.134239476	0.1	0.1	0.1	0	100	100	5	0.0006370544434	2
3	2.340059281	0.1	0.1	0	0.02	100	100	5	0.0006704330444	5
4	2.471867699	0.1	0.1	0	0.02	200	200	5	0.00069999969482	1
5	2.578057808	0.1	0	0	0.02	100	100	5	0.0006170272827	4
6	2.616063256	0.1	0.1	0.1	0	200	200	5	0.0005986690521	1
7	2.647512216	0.1	0	0	0.02	200	200	12	0.0008232593536	6
8	2.735691755	0.1	0	0	0.2	100	100	12	0.0006291866302	0
9	2.747862063	0.1	0.1	0.1	0.02	100	100	5	0.0007157325745	5
10	2.973354266	0	0.1	0	0.2	200	200	5	0.0008091926575	1
11	2.981162373	0.1	0.1	0	0.02	100	100	12	0.0004754066467	0
12	2.987723742	0	0	0	0.2	200	200	5	0.0004866123199	2

Table B.1 continued from previous page

Nr	MSE	srr	str	srt	sst	Score	minimum	Max laser distance	Non zero ratio	Enclosed areas
13	3.07248788	0.1	0	0.1	0	200	5	0.0006918907166	2	
14	3.187656568	0.1	0	0	0	200	5	0.0004749298096	0	
15	3.21395944	0.1	0.1	0	0.02	200	12	0.0004754066467	0	
16	3.313792285	0	0.1	0.1	0.2	100	12	0.0005960464478	1	
17	3.316087255	0.1	0.1	0.1	0.02	200	12	0.0005633831024	3	
18	3.338377154	0.1	0.1	0	0.2	200	12	0.0005033016205	0	
19	3.55289756	0.1	0	0	0	100	5	0.0006022453308	8	
20	3.596973712	0.1	0.1	0	0	100	12	0.0007541179657	4	
21	3.668633601	0	0	0	0.02	100	12	0.0007264614105	1	
22	3.682077105	0.1	0.1	0.1	0.02	200	5	0.000617980957	7	
23	3.709706549	0.1	0	0.1	0.02	100	5	0.0007274150848	1	
24	3.746822154	0	0	0.1	0.02	200	12	0.0007719993591	3	
25	3.78732016	0.1	0	0.1	0.02	200	5	0.0007374286652	8	
26	3.891545145	0	0	0.1	0.02	100	5	0.0006086826324	8	
27	3.952043751	0	0.1	0	0.2	100	12	0.0007100105286	6	
28	4.058760008	0	0.1	0.1	0	200	12	0.0007829666138	4	
29	4.099466424	0.1	0	0.1	0.02	200	12	0.0005252361298	2	
30	4.310226052	0	0.1	0	0	200	12	0.0007393360138	0	
31	4.39559077	0.1	0.1	0	0	200	12	0.000777721405	4	
32	4.425120822	0	0	0.1	0	100	5	0.000657081604	10	
33	4.52981167	0	0	0.1	0.2	200	12	0.000853061676	1	
34	4.620132549	0	0.1	0	0.2	200	12	0.0005867481232	2	
35	6.830055001	0	0.1	0.1	0.2	200	12	0.0007035732269	0	
36	10.93302187	0	0.1	0	0	200	5	0.0008497238159	15	

TABLE B.2: Gmapping: Hallway results

Nr	MSE	srr	str	srt	stt	Score	minimum	Max laser	Non zero	Enclosed
								distance	ratio	areas
1	7.664462355	0.1	0.1	0.1	0.02	200	5	0.0002472400665	0	
2	7.805968912	0	0	0	0	100	12	0.000244140625	0	
3	7.808512965	0.1	0	0.1	0.02	100	5	0.0002510547638	1	
4	7.918567314	0	0	0	0.2	100	12	0.000248670578	1	
5	7.949831085	0.1	0	0	0	100	5	0.0002472400665	0	
6	8.192164038	0.1	0.1	0	0	100	5	0.0002481937408	0	
7	8.856998542	0	0.1	0.1	0.2	100	5	0.000275850296	1	
8	9.244814708	0	0	0	0	100	5	0.0002708435059	0	
9	9.508676165	0	0.1	0.1	0.02	100	12	0.0002639293671	0	
10	9.557861379	0.1	0.1	0.1	0.2	100	12	0.0002672672272	0	
11	9.60761666	0.1	0.1	0	0.2	100	12	0.0002627372742	0	
12	9.689330864	0	0	0	0	200	5	0.0002493858337	2	
13	9.795294606	0	0	0.1	0	100	5	0.0002627372742	0	
14	9.808942766	0.1	0	0.1	0	200	5	0.0002629756927	0	
15	9.812516872	0	0	0.1	0	200	12	0.000253200531	1	
16	9.83150232	0	0	0	0.02	100	12	0.0002603530884	0	
17	9.849223758	0.1	0	0.1	0	100	12	0.0002660751343	2	
18	9.866127462	0	0.1	0.1	0	100	12	0.0002660751343	1	
19	9.898720901	0	0.1	0.1	0	200	12	0.0002715587616	2	
20	9.899873319	0	0	0.1	0.2	200	12	0.0002553462982	0	
21	9.914316362	0.1	0.1	0	0	100	12	0.0002562999725	0	
22	9.997753939	0.1	0	0	0.02	100	5	0.0002663135529	0	
23	10.03236391	0.1	0.1	0	0.2	100	5	0.0002672672272	0	
24	10.06559212	0.1	0.1	0.1	0.02	100	12	0.0002796649933	2	
25	10.1730436	0.1	0	0.1	0.2	200	12	0.0002756118774	0	
26	10.18292764	0	0	0.1	0.2	100	5	0.0002925395966	2	

Table B.2 continued from previous page

Nr	MSE	srr	str	srt	sst	Score	minimum	Max laser distance	Non zero ratio	Enclosed areas
27	10.27843246	0.1	0	0.1	0	200	12	0.0002479553223	0	
28	10.31840619	0.1	0	0.1	0.2	200	5	0.0002570152283	0	
29	10.36744311	0.1	0.1	0	0.02	100	12	0.0002851486206	1	
30	10.40320488	0	0.1	0	0.2	100	5	0.0002694129944	0	
31	10.57708213	0.1	0	0.1	0.02	100	12	0.0002906322479	2	
32	10.57985811	0	0.1	0.1	0	100	5	0.0002760887146	0	
33	10.59560797	0	0	0.1	0.2	100	12	0.0002920627594	0	
34	10.61013852	0	0	0.1	0.02	100	12	0.0002880096436	0	
35	10.61410781	0.1	0.1	0	0.02	100	5	0.0002901554108	0	
36	10.63349219	0.1	0.1	0.1	0	200	5	0.0002756118774	1	
37	10.68619002	0.1	0	0.1	0.02	200	12	0.0002584457397	0	
38	10.70102842	0.1	0.1	0.1	0	100	12	0.0002822875977	1	
39	10.74803067	0	0	0.1	0.2	200	5	0.0002644062042	1	
40	10.81060507	0	0.1	0	0	100	12	0.0002644062042	1	
41	10.83133239	0.1	0	0	0.02	200	5	0.0002670288086	0	
42	10.89755772	0	0.1	0.1	0.2	200	5	0.0002677440643	1	
43	10.90412703	0.1	0.1	0.1	0.2	200	12	0.0002653598785	0	
44	10.93283545	0	0.1	0	0	200	5	0.0002474784851	0	
45	11.01114581	0.1	0	0.1	0.02	200	5	0.000257730484	1	
46	11.01811551	0	0	0	0	200	12	0.00026512146	0	
47	11.06340893	0	0.1	0	0.2	200	5	0.0002639293671	0	
48	11.07217296	0	0.1	0	0	100	5	0.0002734661102	2	
49	11.13165477	0	0	0.1	0.02	200	5	0.0002589225769	0	
50	11.14860612	0.1	0.1	0.1	0	200	12	0.0002701282501	0	
51	11.18789729	0.1	0.1	0.1	0.2	100	5	0.0002827644348	3	
52	11.18980318	0.1	0.1	0.1	0.02	100	5	0.0002820491791	3	

Table B.2. continued from previous page

Nr	MSE	srr	str	srt	sft	Score	minimum	Max laser distance	Non zero ratio	Enclosed areas
53	11.21740202	0	0.1	0	0.2	200	12	0.0002582073212	0	
54	11.22519983	0.1	0	0	0.2	100	5	0.0002820491791	1	
55	11.25451642	0	0.1	0	0.02	100	12	0.000298500061	1	
56	11.30755926	0	0.1	0.1	0.02	100	5	0.0002839565277	1	
57	11.34933001	0.1	0	0.1	0	100	5	0.0002675056458	0	
58	11.51052414	0.1	0	0	0	100	12	0.0002646446228	0	
59	11.56834015	0.1	0.1	0.1	0	100	5	0.0002765655518	1	
60	11.61813572	0.1	0	0	0.2	200	5	0.0002639293671	0	
61	11.66441652	0	0.1	0	0.2	100	12	0.0002694129944	0	
62	11.71808728	0	0.1	0.1	0.2	200	12	0.0002768039703	1	
63	11.77223883	0.1	0.1	0	0.02	200	12	0.0002706050873	0	
64	11.90798088	0	0.1	0	0.02	100	5	0.0002844333649	0	
65	11.96931483	0	0.1	0.1	0.02	200	12	0.0002663135529	0	
66	12.00886092	0.1	0.1	0.1	0.2	200	5	0.0002717971802	0	
67	12.04224898	0	0.1	0	0	200	12	0.000269651413	0	
68	12.08477331	0	0	0.1	0.02	100	5	0.0002906322479	1	
69	12.27843607	0.1	0.1	0	0.2	200	5	0.0002658367157	1	
70	12.28193363	0.1	0	0	0.02	200	12	0.0002601146698	0	
71	12.51622613	0.1	0	0.1	0.2	100	12	0.0002875328064	1	
72	12.62167204	0	0.1	0	0.02	200	5	0.0002744197845	0	
73	12.71582882	0.1	0	0	0.2	100	12	0.0003092288971	2	
74	12.78048286	0	0	0.1	0.02	200	12	0.0002834796906	0	
75	12.7833762	0	0.1	0	0.02	200	12	0.0002720355988	0	
76	12.82387507	0	0.1	0.1	0.2	100	12	0.0002973079681	0	
77	12.88697453	0	0	0	0.02	200	12	0.0002727508545	1	
78	13.05424896	0	0	0.1	0	100	12	0.0003082752228	2	

Table B.2 continued from previous page

Nr	MSE	srr	str	srt	sft	Score	minimum	Max laser distance	Non zero ratio	Enclosed areas
79	13.09328643	0	0	0	0.2	100	5	0.0002987384796	0	
80	13.13662931	0.1	0.1	0.1	0.02	200	12	0.000280380249	0	
81	13.13956661	0	0.1	0.1	0.02	200	5	0.0002763271332	0	
82	13.15616577	0.1	0.1	0	0	200	12	0.0002672672272	0	
83	13.23719917	0	0	0	0.02	200	5	0.0002686977386	0	
84	13.48350511	0.1	0.1	0	0.02	200	5	0.0002701282501	0	
85	13.7492556	0	0	0	0.2	200	12	0.0002801418304	0	
86	13.77021888	0	0	0	0.2	200	5	0.0002777576447	1	
87	13.83664114	0	0.1	0.1	0	200	5	0.0002913475037	1	
88	13.92067128	0.1	0	0.1	0.2	100	5	0.0003125667572	3	
89	14.3238	0.1	0.1	0	0	200	5	0.0002856254578	0	
90	14.37679127	0.1	0	0	0	200	5	0.0002760887146	1	
91	14.50009101	0.1	0.1	0	0.2	200	12	0.0002870559692	2	
92	14.52029817	0	0	0.1	0	200	5	0.0002837181091	0	
93	14.92667489	0.1	0	0	0.02	100	12	0.000301361084	4	
94	15.07697047	0.1	0	0	0	200	12	0.000289440155	0	
95	15.33711272	0	0	0	0.02	100	5	0.0002951622009	3	
96	15.79760295	0.1	0	0	0.2	200	12	0.000287771225	1	

TABLE B.3: Gmapping: Double Hallway results

Nr	MSE	srr	str	srt	sft	Score	minimum	Max laser distance	Non zero ratio	Enclosed areas
1	6.517846713	0	0	0	0.02	100	12	0.000298500061	0	
2	6.561282503	0.1	0.1	0	0	100	12	0.0003015995026	0	
3	6.651770729	0.1	0.1	0.1	0.02	100	5	0.0003023147583	1	
4	6.806089568	0	0.1	0.1	0	100	12	0.0002992153168	0	

Table B.3 continued from previous page

Nr	MSE	srr	str	srt	sft	Score	minimum	Max laser distance	Non zero ratio	Enclosed areas
5	6.807596964	0.1	0	0	0.02	100	12	0.0003044605255	1	
6	6.830401752	0.1	0	0.1	0.2	100	5	0.0002982616425	0	
7	7.379133617	0	0.1	0.1	0.02	100	12	0.0003025531769	1	
8	7.524928981	0	0	0	0	100	5	0.0002954006195	0	
9	7.560463847	0.1	0.1	0	0.2	100	5	0.00031208992	1	
10	7.765518474	0.1	0	0.1	0	200	12	0.0002746582031	2	
11	8.082584899	0	0	0	0.02	100	5	0.0003242492676	2	
12	8.329936206	0.1	0.1	0	0.02	100	12	0.000321149826	0	
13	8.409336844	0	0	0.1	0.02	100	12	0.000316619873	0	
14	8.434494163	0	0.1	0.1	0.02	100	5	0.0003111362457	0	
15	8.628187023	0.1	0	0	0	100	5	0.0003216266632	1	
16	8.649328743	0.1	0.1	0.1	0.2	100	5	0.0003151893616	0	
17	8.73655645	0.1	0	0.1	0.2	200	5	0.0002906322479	0	
18	8.866397466	0	0	0.1	0.2	100	12	0.0003056526184	1	
19	8.907315744	0.1	0	0.1	0	100	5	0.0003037452698	0	
20	8.912540024	0	0	0	0.2	200	12	0.000292301178	2	
21	8.989880188	0.1	0.1	0.1	0.2	200	12	0.0003089904785	1	
22	9.040706735	0	0	0	0	100	12	0.0003221035004	0	
23	9.286546251	0	0.1	0.1	0.2	100	5	0.0003108978271	0	
24	9.366579246	0.1	0	0	0.02	100	5	0.0003073215485	0	
25	9.453403523	0.1	0	0	0	100	12	0.0003108978271	0	
26	9.469359111	0.1	0	0.1	0.2	200	12	0.0003128051758	1	
27	9.477707815	0.1	0.1	0.1	0.2	200	5	0.0002620220184	0	
28	9.574252966	0.1	0	0.1	0.2	100	12	0.0003185272217	1	
29	9.666560983	0.1	0.1	0	0	100	5	0.0003132820129	0	
30	9.716726028	0.1	0	0.1	0.02	100	12	0.0003306865692	0	

Table B.3 continued from previous page

Nr	MSE	srr	str	srt	sst	Score	minimum	Max laser distance	Non zero ratio	Enclosed areas
31	9.733187994	0	0	0.1	0.02	200	12	0.000328540802	2	
32	9.741557955	0	0.1	0	0.2	100	12	0.0003163814545	0	
33	9.744350913	0	0	0.1	0.2	100	5	0.0003433227539	0	
34	9.91398216	0	0.1	0	0.02	100	12	0.00031208992	1	
35	10.12772392	0	0.1	0.1	0.2	200	5	0.0003023147583	0	
36	10.20306076	0.1	0.1	0.1	0.02	100	12	0.0003213882446	0	
37	10.60933141	0	0.1	0	0.02	200	12	0.0003085136414	1	
38	10.61415913	0	0.1	0	0	100	12	0.0003311634064	0	
39	10.74918286	0.1	0	0	0.2	100	12	0.000335931778	0	
40	11.12150255	0.1	0.1	0.1	0	200	12	0.0003070831299	1	
41	11.13857218	0	0	0.1	0.02	200	5	0.000316619873	1	
42	11.28593289	0	0	0.1	0.2	200	5	0.0003063678741	0	
43	11.33575308	0.1	0.1	0	0.02	200	12	0.0003087520599	1	
44	11.55783224	0	0.1	0	0	200	12	0.0002942085266	2	
45	11.84362519	0.1	0.1	0	0.2	100	12	0.0003395080566	0	
46	11.89221726	0	0.1	0	0	200	5	0.0003173351288	0	
47	11.90897459	0	0.1	0.1	0.2	200	12	0.0003349781036	0	
48	12.11041546	0.1	0	0	0	200	5	0.0002844333649	0	
49	12.51022807	0	0	0	0.2	200	5	0.0003256797791	2	
50	12.75171679	0.1	0.1	0.1	0.02	200	5	0.0003106594086	0	
51	12.96675964	0.1	0.1	0.1	0.02	200	12	0.0003304481506	1	
52	13.2118305	0	0.1	0.1	0.02	200	12	0.0003135204315	2	
53	13.26323238	0	0	0	0.02	200	5	0.0003392696381	3	
54	13.34816955	0.1	0.1	0	0.2	200	12	0.0003294944763	3	
55	13.98692385	0.1	0	0.1	0.02	200	12	0.0003209114075	0	
56	14.01284042	0.1	0.1	0.1	0	200	5	0.0003197193146	0	

Table B.3 continued from previous page

Nr	MSE	srr	str	srt	stt	Score	minimum	Max laser distance	Non zero ratio	Enclosed areas
57	14.17859187	0.1	0	0.1	0	200	5	0.0003316402435	2	
58	14.26486791	0.1	0	0.1	0.02	200	5	0.0003461837769	4	
59	14.28342088	0	0	0.1	0.02	100	5	0.0003530979156	0	
60	14.30838849	0.1	0	0	0	200	12	0.0003182888031	1	
61	14.44531867	0	0.1	0.1	0	200	12	0.0003173351288	0	
62	14.62233172	0.1	0.1	0.1	0	100	12	0.0003490447998	0	
63	14.7423947	0	0	0.1	0	200	5	0.0003600120544	4	
64	14.91747212	0	0.1	0	0.2	200	5	0.0003435611725	3	
65	14.95256605	0.1	0.1	0	0.02	200	5	0.0003609657288	1	
66	14.96373703	0	0	0	0.02	200	12	0.0003223419189	3	
67	14.96721736	0.1	0	0	0.2	200	5	0.0003232955933	1	
68	15.23227845	0.1	0	0	0.02	200	5	0.0003178119659	0	
69	15.42439118	0.1	0.1	0	0.2	200	5	0.0003192424774	2	
70	15.61855633	0.1	0.1	0	0	200	5	0.0003445148468	3	
71	16.11807422	0	0	0.1	0	200	12	0.0003485679626	2	
72	17.11094936	0.1	0	0	0.02	200	12	0.0003399848938	3	
73	17.28696993	0	0	0	0	200	12	0.0003576278687	0	
74	17.34049883	0	0.1	0	0.2	200	12	0.0003640651703	2	
75	19.22922685	0.1	0.1	0	0	200	12	0.0003459453583	0	

TABLE B.4: Gmapping: Elevators results

Nr	MSE	srr	str	srt	stt	Score	minimum	Max laser distance	Non zero ratio	Enclosed areas
1	14.93401111	0	0	0	0	100	5	0.001325130463	21	

TABLE B.5: Gmapping: Elevators Extended results

Nr	MSE	srr	str	srt	stt	minimum Score	Max laser distance	Non zero ratio	Enclosed areas
1	13.09692774	0.1	0	0.1	0.02	200	12	0.001775979996	29
2	13.80526659	0.1	0.1	0.1	0	200	12	0.001490592957	32
3	14.82018973	0.1	0	0.1	0.2	200	12	0.001759767532	27
4	16.23083898	0.1	0	0	0	200	5	0.001705884933	22
5	17.45594883	0.1	0.1	0	0.2	200	5	0.001652956009	37
6	18.14275366	0.1	0.1	0	0	200	12	0.001812458038	26
7	19.21168386	0	0	0.1	0.02	200	5	0.001860618591	36
8	19.59142004	0.1	0	0	0.02	200	5	0.00169968605	26
9	20.06964474	0.1	0	0.1	0.02	200	5	0.001923799515	24
10	20.48280039	0	0	0.1	0	200	12	0.001814365387	31
11	21.58312484	0	0.1	0.1	0.02	200	5	0.001889705658	26
12	26.08676606	0	0.1	0	0.02	200	12	0.001908063889	16
13	26.61652017	0	0	0.1	0.02	200	12	0.001836776733	35
14	32.96377046	0.1	0.1	0	0.02	200	5	0.00218462944	29
15	34.9151411	0	0.1	0.1	0.2	100	5	0.002099514008	35
16	87.07106233	0.1	0	0.1	0	100	12	0.003103494644	33

TABLE B.6: Gmapping: Elevators Experiment results

Nr	MSE	srr	str	srt	stt	minimum Score	Max laser distance	Non zero ratio	Enclosed areas
1	7.2019722	0	0.1	0.1	0.02	100	5	0.001425981522	21
2	8.137313035	0.1	0	0.1	0.2	100	5	0.001401901245	16
3	8.464335623	0.1	0	0.1	0.2	200	5	0.001352071762	30
4	9.41214818	0	0.1	0	0.2	100	12	0.001488685608	21
5	9.993503993	0.1	0.1	0.1	0.2	100	5	0.001516342163	23

Table B.6 continued from previous page

Nr	MSE	srr	sfr	srt	sst	Score	minimum	Max laser distance	Non zero ratio	Enclosed areas
6	10.37139394	0.1	0	0.1	0.2	100	12	0.001518726349	17	
7	11.1217625	0	0	0	0.2	200	12	0.00146484375	27	
8	11.65525043	0.1	0.1	0.1	0	100	12	0.001535415649	10	
9	11.7915281	0	0	0.1	0.2	200	12	0.001419305801	36	
10	11.94488499	0.1	0.1	0.1	0.02	100	12	0.001465797424	12	
11	12.02953154	0	0	0.1	0.2	100	5	0.001601934433	23	
12	12.10855675	0	0	0.1	0.2	100	12	0.001572847366	15	
13	12.15531003	0.1	0	0.1	0	100	12	0.001560211182	19	
14	12.30569938	0	0	0.1	0	200	5	0.00131726265	18	
15	12.38575613	0	0	0	0.02	100	5	0.001569509506	16	
16	12.47340479	0	0.1	0	0	100	12	0.001495361328	13	
17	12.51978834	0.1	0.1	0	0.2	100	5	0.001526117325	13	
18	12.65189645	0.1	0	0.1	0.02	100	12	0.001601219177	20	
19	12.71402827	0	0	0.1	0.02	100	12	0.001560688019	15	
20	12.86575188	0	0.1	0.1	0.2	100	12	0.001588821411	8	
21	12.93944913	0.1	0.1	0	0	100	12	0.001549005508	8	
22	13.30115449	0	0	0.1	0	100	12	0.001554727554	16	
23	13.34502431	0.1	0	0.1	0.02	100	5	0.001661062241	13	
24	13.36343192	0.1	0	0	0	200	5	0.001425981522	22	
25	13.67769126	0.1	0	0	0.2	100	12	0.001626491547	25	
26	13.75222724	0	0	0.1	0.02	200	5	0.001401662827	23	
27	14.04808264	0.1	0	0	0.02	100	12	0.001639604568	8	
28	14.33844863	0	0.1	0.1	0	100	5	0.00159573555	16	
29	14.3907405	0	0.1	0.1	0.02	100	12	0.001662492752	15	
30	14.49002559	0.1	0	0	0	100	5	0.001622915268	13	
31	14.77283946	0.1	0.1	0	0.2	100	12	0.001636505127	12	

Table B.6 continued from previous page

Nr	MSE	srr	str	srt	stt	Score	minimum	Max laser distance	Non zero ratio	Enclosed areas
32	14.82440688	0.1	0.1	0.1	0.02	100	5	0.001643419266	24	
33	14.93805544	0.1	0	0.1	0	100	5	0.00168633461	15	
34	15.21594881	0	0	0.1	0	100	5	0.00158572197	12	
35	15.28942168	0	0.1	0	0.02	100	12	0.001627922058	7	
36	15.37485723	0.1	0	0.1	0.02	200	5	0.001471042633	21	
37	15.46629549	0.1	0.1	0	0	200	5	0.001507043839	28	
38	15.57370803	0	0.1	0.1	0.2	100	5	0.001481533051	16	
39	15.88976675	0.1	0.1	0.1	0.2	200	12	0.001508235931	11	
40	15.90151384	0.1	0.1	0.1	0	100	5	0.001562595367	14	
41	16.65530408	0.1	0.1	0.1	0.02	200	5	0.001660585403	28	
42	16.90094119	0	0.1	0.1	0.02	200	12	0.001701116562	33	
43	17.17722075	0	0.1	0	0	100	5	0.001610040665	22	
44	17.42461692	0.1	0.1	0	0.02	100	5	0.001690864563	16	
45	17.48737622	0.1	0	0.1	0	200	5	0.001525878906	16	
46	18.19941654	0.1	0	0	0.02	100	5	0.00167632103	25	
47	19.03315343	0.1	0.1	0	0.02	100	12	0.001605987549	19	
48	19.25367156	0.1	0.1	0.1	0.2	200	5	0.001721858978	33	
49	19.33853191	0.1	0.1	0	0	100	5	0.001766443253	19	
50	20.38397372	0	0.1	0.1	0	100	12	0.001534700394	14	
51	22.01318306	0	0	0	0.2	200	5	0.001662015915	40	
52	22.49206343	0.1	0.1	0	0.02	200	12	0.001645326614	17	
53	22.93179898	0	0	0.1	0	200	12	0.001725435257	19	
54	23.981555	0.1	0	0	0	100	12	0.001870393753	10	
55	24.41013381	0	0	0	0	100	5	0.001526832581	36	
56	24.46078592	0	0	0	0.02	200	5	0.001687765121	26	
57	24.53413088	0	0	0	0.2	100	5	0.001863718033	23	

Table B.6 continued from previous page

Nr	MSE	srr	sfr	srt	stt	Score	minimum	Max laser	Non zero	Enclosed
								distance	ratio	areas
58	29.43708745	0	0.1	0.1	0.2	200	200	12	0.001718759537	14
59	32.19287338	0.1	0.1	0	0.2	200	200	12	0.001863002777	28

Bibliography

- [1] T. Andre, D. Neuhold, and C. Bettstetter. "Coordinated Multi-Robot Exploration: Out of the Box Packages for ROS". In: *Proc. of IEEE GLOBECOM WiUAV Workshop*. Dec. 2014.
- [2] Oliver Brock and Oussama Khatib. "High-speed navigation using the global dynamic window approach". In: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. Vol. 1. IEEE. 1999, pp. 341–346.
- [3] Alexander Buyval, Ilya Afanasyev, and Evgeni Magid. "Comparative analysis of ROS-based monocular SLAM methods for indoor navigation". In: *Ninth International Conference on Machine Vision (ICMV 2016)*. Vol. 10341. International Society for Optics and Photonics. 2017, 103411K.
- [4] Rachael N Darmanin and Marvin K Bugeja. "Autonomous Exploration and Mapping using a Mobile Robot Running ROS." In: *ICINCO (2)*. 2016, pp. 208–215.
- [5] Anton Filatov et al. "2D SLAM Quality Evaluation Methods". In: *arXiv preprint arXiv:1708.02354* (2017).
- [6] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. "The dynamic window approach to collision avoidance". In: *IEEE Robotics & Automation Magazine* 4.1 (1997), pp. 23–33.
- [7] Udo Frese and Gerd Hirzinger. "Simultaneous localization and mapping-a discussion". In: *Proceedings of the IJCAI Workshop on Reasoning with Uncertainty in Robotics*. Seattle. 2001, pp. 17–26.
- [8] Jorge Fuentes-Pacheco, José Ruiz-Ascencio, and Juan Manuel Rendón-Mancha. "Visual simultaneous localization and mapping: a survey". In: *Artificial Intelligence Review* 43.1 (2015), pp. 55–81.
- [9] A. Gardecki and M. Podpora. "Experience from the operation of the Pepper humanoid robots". In: *2017 Progress in Applied Electrical Engineering (PAEE)*. June 2017, pp. 1–6. DOI: 10.1109/PAEE.2017.8008994.
- [10] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. "Improved techniques for grid mapping with rao-blackwellized particle filters". In: *IEEE transactions on Robotics* 23.1 (2007), pp. 34–46.
- [11] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling". In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE. 2005, pp. 2432–2437.
- [12] Ronald Cumbal Guerron. "Socially-Aware Navigation: Guiding Behavior with Pepper". MA thesis. Delft University of Technology, Oct. 2017.
- [13] Wolfgang Hess et al. "Real-Time Loop Closure in 2D LIDAR SLAM". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1271–1278.
- [14] S. Kohlbrecher et al. "A Flexible and Scalable SLAM System with Full 3D Motion Estimation". In: *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE. Nov. 2011.

- [15] Mathieu Labbé and François Michaud. “Long-term online multi-session graph-based SPLAM with memory management”. In: *Autonomous Robots* (Nov. 2017). ISSN: 1573-7527. DOI: 10.1007/s10514-017-9682-5. URL: <https://doi.org/10.1007/s10514-017-9682-5>.
- [16] M. T. Lázaro et al. “A lightweight navigation system for mobile robots”. In: *ROBOT 2017: Third Iberian Robotics Conference*. Sevilla, Spain, Nov. 2017.
- [17] Montiel J. M. M. Mur-Artal Raúl and Juan D. Tardós. “ORB-SLAM: a Versatile and Accurate Monocular SLAM System”. In: *IEEE Transactions on Robotics* 31.5 (2015), pp. 1147–1163. DOI: 10.1109/TR0.2015.2463671.
- [18] Raúl Mur-Artal and Juan D. Tardós. “ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras”. In: *IEEE Transactions on Robotics* 33.5 (2017), pp. 1255–1262. DOI: 10.1109/TR0.2017.2705103.
- [19] Albin PÅlsson and Markus Smedberg. “Investigating Simultaneous Localization and Mapping for AGV systems”. In: ().
- [20] Vittorio Perera et al. “Setting Up Pepper For Autonomous Navigation And Personalized Interaction With Users”. In: *CoRR abs/1704.04797* (2017). arXiv: 1704.04797. URL: <http://arxiv.org/abs/1704.04797>.
- [21] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.
- [22] Sean Quinlan and Oussama Khatib. “Elastic bands: Connecting path planning and control”. In: *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*. IEEE. 1993, pp. 802–807.
- [23] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. “Planning of multiple robot trajectories in distinctive topologies”. In: *Mobile Robots (ECMR), 2015 European Conference on*. IEEE. 2015, pp. 1–6.
- [24] Joao Machado Santos, David Portugal, and Rui P Rocha. “An evaluation of 2D SLAM techniques available in robot operating system”. In: *Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on*. IEEE. 2013, pp. 1–6.
- [25] *Technical overview*. http://doc.aldebaran.com/2-5/family/pepper_technical/index_pep.html. Accessed: 2018-04-12.
- [26] Sebastian Thrun and John J Leonard. “Simultaneous localization and mapping”. In: *Springer handbook of robotics*. Springer, 2008, pp. 871–889.
- [27] Stephan Wirth and Johannes Pellenz. “Exploration transform: A stable exploring algorithm for robots in rescue environments”. In: *Safety, Security and Rescue Robotics, 2007. SSRR 2007. IEEE International Workshop on*. IEEE. 2007, pp. 1–5.
- [28] Brian Yamauchi. “A frontier-based approach for autonomous exploration”. In: *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*. IEEE. 1997, pp. 146–151.