Localization and Mapping with Kinect Camera using RBGD-SLAM

Master Thesis

Submitted by: Nguyen, Truong Thanh

Matriculation number: 1009851

First examiner: Prof. Dr. Peter Nauth

Second examiner: Prof. Dr. Andreas Pech

Department of Information Technology
UNIVERSITY OF FRANKFURT

A dissertation submitted to the University of Frankfurt in accordance with the requirements of the degree of MASTER OF ENGINEERING(INFORMATION TECHNOLOGY).

DATE OF SUBMISSION: 17.08.2017

Abstract

The Simultaneous Localization and Mapping (SLAM) problem asks if it is possible for a mobile robot to be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map. A solution to the SLAM problem has been seen as a **holy grail** for the mobile robotics community as it would provide the means to make a robot truly autonomous. SLAM is an essential task for the autonomy of a robot. Nowadays, the problem of SLAM is considered solved when range sensors such as lasers or sonar are used to built 2D maps of small static environments. However for large-scale long-term SLAM, robot has to deal with unknown initial positioning caused by either the kidnapped robot problem or multi-session mapping.

The objective of this thesis is to implement one or more functionalities based on camera-based sensors in a mobile autonomous robot. A mobile robot prototype has been configured to run ROS (Robot Operating System), a middleware framework that is suited to the development of robotic systems. The system uses RTAB-Map (Real-Time Appearance Based Mapping) to survey the surroundings and a built navigation stack in ROS to navigate autonomously against easy targets in the map. The method uses a Kinect for Xbox 360 for create 3D map,a 2D laser scanner as the main sensor and wheel encoder for odometry. Test results, obtained from both live and simulated trials, indicate that the robot is able to form 3D and 2D map of the surroundings. The method has weaknesses that are related to the ability to find visual features. Further testing has demonstrated that the robot can navigate autonomously, another 2D SLAM method call Hector SLAM was implemented to combined with RTAB-Map also gave good result, but there is still room for improvement. Better results can be achieved with a new movable platform and further tuning of the system. In conclusion, ROS works well as a development tools for robots, and the current system is suitable for further development. Hector SLAM and RTAB-Map can combined together to get a better result in mapping. RTAB-Maps suitability for use on an industrial installation is still uncertain and requires further testing.

DEDICATION AND ACKNOWLEDGEMENTS

This acknowledgement is intended to thank all those who guided and supported me in carrying out this thesis work at **Frankfurt University of Applied Sciences**, Germany.

I am grateful to **Prof. Dr. Peter Nauth**, Frankfurt University of Applied Sciences for giving me an opportunity to carry out project in this esteemed University and allowing me to work on such interesting project. I also want to thank **Prof. Dr. Andreas Pech**, professor at Frankfurt University of Applied Sciences, for supervise me during my thesis work.

I want to thank **Mr.Sudeep Sharan** and **Mr. J. Umansky**, Frankfurt University of Applied Sciences guiding and helping in carrying out the thesis project work smoothly.

Finally, I wish to express thanks to **Dr.Mathieu Labble**, University SHERBROOKE for help me during my work and I also want to express my thanks to my **colleagues and friends** who supported me during the project period.

AUTHOR'S DECLARATION

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..... DATE: .....

ABBREVIATION

RTAB-Map Real Time Appearance Base Mapping **ROS** Robot Operating System

SLAM Simultaneous Localization and Mapping

RGBD-SLAM Red Green Blue Depth-SLAM

TCP Transmission Control Protocol

EKF Extended Kalman Filter

LIDAR Light Detection and Ranging

LTM Long Term Memory

STM Short Term Memory

SURF Speeded Up Robust Features

SIFT Scale-invariant feature transform

URDF United Robot Description Format

COLLADA COLLAborative Design Activity

XML eXtensible Markup Language

RVIZ ROS visualization

RTAB-Map Real-Time Appearence Based Mapping

UAV Unmanned Aerial Vehicle

CLM Concurrent Localization and Mapping

LMS100 Laser Scanner

VNC Virtual Network Computing

TABLE OF CONTENTS

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

This thesis presents and documents this master's project, on robotic develop which was carried out at the Department of Information technology in Summer 2017. Master thesis is worth 30 credits, and the project duration is set to 20 full-time weeks with the possibility of extension in case of a valid reason. The work presented in this thesis is carried out as an independent effort, which is supervised by Professor. Dr Peter Nauth and Professor. Dr Andreas Pech through regular status meetings.

1.1 The Project Proposal - Mobile Autonomous Robot

The robot system that was used in this project has been developed over the course of many preceding master and specialization projects. The long term goal of these projects is to develop mobile autonomous robot concepts to assist people in indoor environment, like pouring water and serve for people. The topic of this thesis is based on the project propose by Professor.Dr Peter Nauth at department of Information Technology, which suggests some possible application for such a robot:

- The robot should autonomous navigate in an indoor environment, like a room or an corridor.
- Robot should avoid the obstacles when it move from A to B position, the obstacles here include static and dynamic obstacles.
- Robot should classify object, for instance recognize the cup and the bottle, after that robot could pouring water from bottle to cup.
- Allow personnel to perform remote inspection and maintenance through remote control.

1.2 Implementation Overview

Deciding on a Goal

To make robot autonomous navigate in the environment, the vision based navigation is proved to be an outstanding algorithm to choose. A robot with the ability to build a map of the surroundings and relocate autonomously is considered to be a good starting point for further development of vision based solutions.

Limitations

During the implementation process, the priority is how to run the system smoothly. Robustness, optimization and elegance has been abandoned in favor of the opportunity to increase the scope of the project.

Selecting Tools and Hardware

As a continuation of Volksbot project, the robot was equipped with a 2D Laser scanner, a SICK LMS100 sensors for measuring ranges of up to 20 m. To create 3-D map, the robot was equipped with a 3D camera, a Kinect for XBOX 360 capable of perceiving depth images at a high frame rate (30Hz).

To use the Kinect, a Kinect driver needs installing in ROS, usable drivers can be Freenect or openNI, however the latter was obsolete today so Freenect-drive was chosen.

The work and solutions presented in this thesis revolve around the process of integrating ROS with the mobile robot. Installing ROS on Ubuntu Linux is by far the easiest way to begin using the framework. For this reason, and to avoid interfering with another project on the same robot, it was decided that an additional computer running Linux should be fitted to the robot. The new robot platform configuration is shown in figure 1.1.

Two supporting tools were implemented in addition to the robot software: a simple concept of control station with remote computer and a hand held remote control with on a smart phone or tablet. An overview of the complete system is shown in figure 1.2.

Goal of the Thesis

The goal of this thesis is to make a research about SLAM and try to create a map of the University Campus while determine the position of the robot in the map in the same time. First, A Kinect camera and 2D laser-scanner SICK will be used to draw a map of three rooms and the corridor in building 8, Frankfurt University of Applied Science. The whole system will be put on an wheeled robot, the wheel encoder will be used to provide robot's odometry. This is a specific application based on ROS (Robot Operating System) to demonstrate operating mode of ROS.

The reasons of choosing RTAB-map to test on robot include: Firstly, it is quite a new approach in SLAM, this method provides a 3D-map instead of 2D-map like other methods, so a lot of research opportunities still exist. Secondly, the RTAB-algorithm aims to find the optimal paths among several patterns, so it allows to save long-term memory when the robot performs multi-mapping.

Functionality

(a) (b)

FIGURE 1.1. (a) Robot left side. (b) Robot front

- 1. Simultaneous localization and mapping based on computer vision.
- 2. Mapping over multiple sessions.
- 3. Autonomous navigation to a simple goal.
- 4. 3D and 2D obstacle avoidance in navigation mode.
- 5. The robot can be controlled from the on-board keyboard.
- 6. The robot can be controlled by Remote Control.
- 8. The robot can receive velocity commands over WiFi.

1.3 Thesis structure

Chapter 2 - Background Theory

Chapter 2 introduces the background theory which is necessary to understand the steps of implementation presented in chapter 4.

Chapter 3 - Sensor and Software

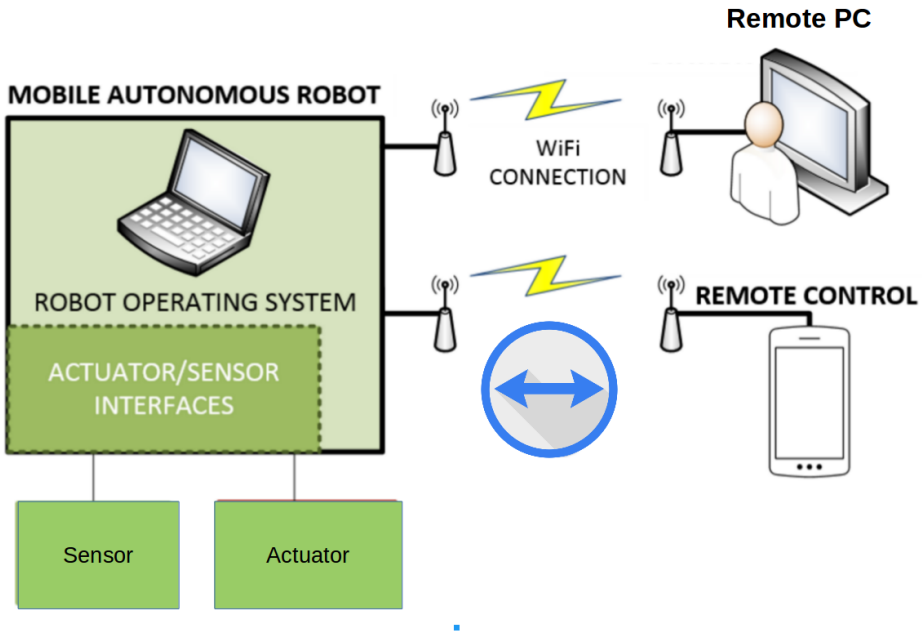


FIGURE 1.2. System Concept. An on-board computer using ROS to handle actuators and sensors. Remote operation is available through an OCS or a hand-held device with wifi.

Chapter 3 introduces the Volksbot RT3 and SICK LMS 100 coordinate system, also an introduction about the basic of a ROS simulation tools, Gazebo and Rviz.

Chapter 4 - Implementation

Presents the implementations of the whole system in the project. The structure of the robot system, both hardware and software, is documented.

Chapter 5 - Results and Evaluation

This chapter explains how the implementations were tested, and presents the ensuing test results.

Chapter 6 - Discussion and Conclusion

The discussion part contains an assessment of the results and implementations from chapters 4 and 5. The concluding part will highlight how well the objectives in the initial problem description are achieved. The final result and recommendations for future work is summed up in a final conclusion at the end of this chapter, as well.

CHAPTER 2

THEORETICAL BACKGROUND

Navigation is one of the most challenging competences required of a mobile robot. Success in navigation requires success at the four building blocks of navigation: perception, the robot must interpret its sensors to extract meaningful data; localization, the robot must determine its position in the environment; cognition, the robot must decide how to act to achieve goal; and motion control, the robot muss modulate its motor output to achieve the desired directory.

Mapping is the problem of integrating the information gathered with the robot's sensors into a given representation. It can be described by the question "What does the world look like?"[4] Central aspects in mapping are the representation of the environment and the interpretation of sensor data. In contrast, localization is the problem of estimating the pose of the robot relative to a map. In other words, the robot has to answer the question, "Where am I?" Typically, one distinguishes between pose tracking, where the initial pose of the vehicle is known, and global localization, in which no a priori knowledge about the starting position is given.

2.1 Introduction

This chapter will provide the background theory which is necessary to understand the implementations described in chapter 4. Section 2.1.1 introduces some robot terminology and concepts. Section 2.2 provides a thorough introduction to ROS, which is the framework of choice in this implementation. Next sections provide basic insights into SLAM and final section provides a thorough introduction about autonomous navigation in ROS respectively.

2.1.1 Brief Introduction to Robot Terminology

Joints and Links

A robot can be described by a set of rigid links connected to each other by joints. A link is described by a set of kinematic attributes based on its shape and mass. A joint between two links describes the freedom of movement between the coordinate systems, or frames, of each link. The link can also define the linear translation and rotations from parent frame to child frame. When a set of links and joints are put together, they will define the kinematic tree of the robot, i.e. how the robot and its components can move. Typical joint classes are:

- Static Transforms between links are constant.
- Revolute A rotary motion between the links - like a door hinge or a knee.
- Prismatic A linear motion between the links.
- Continous Unbounded rotary motion. Typically used for rotating wheels.

2.2 Robot Operating System (ROS)

2.2.1 Introduction

ROS is a collection of software libraries, tools and drivers intended for robot software development. A ROS installation can be tailored to meet the demands of a wide range of robots with varying complexity. ROS is usually installed in the form of an already built Debian-package. These packages are only compatible with a few versions of Ubuntu which are specified on the ROS homepage. When installed and configured, ROS will run on top of Linux, and can be perceived as an extension of Linux itself[16]. Installing ROS from source is possible, but not recommended. Historic roots of ROS can be traced back to Stanford University at the beginning of the 2000s. At Stanford, several robotics software frameworks, including Stanford AI Robot (STAIR) and the Personal Robot (PR) program, were created to provide dynamic, flexible and well tested foundations for further robot development and research. In 2007, a nearby start-up company and robot incubator, Willow Garage, sought to build upon these concepts, and initiated a collaborative and open development process of a new software framework. This framework eventually became ROS. The framework can be used under the BSD open-source license. Today, ROS comes in many forms and comprises hundreds of advanced packages, algorithms and drivers, making it applicable for hobbyists, industrial automation, research and everything in between.

Similar to an operating system, ROS files are also organized on the hard disk in a particular fashion. In this level, we can see how these files are organized on the disk. Fig.2.1 show how ROS files and folder are organized on the disk.

Here are the explanations of each block in the file system

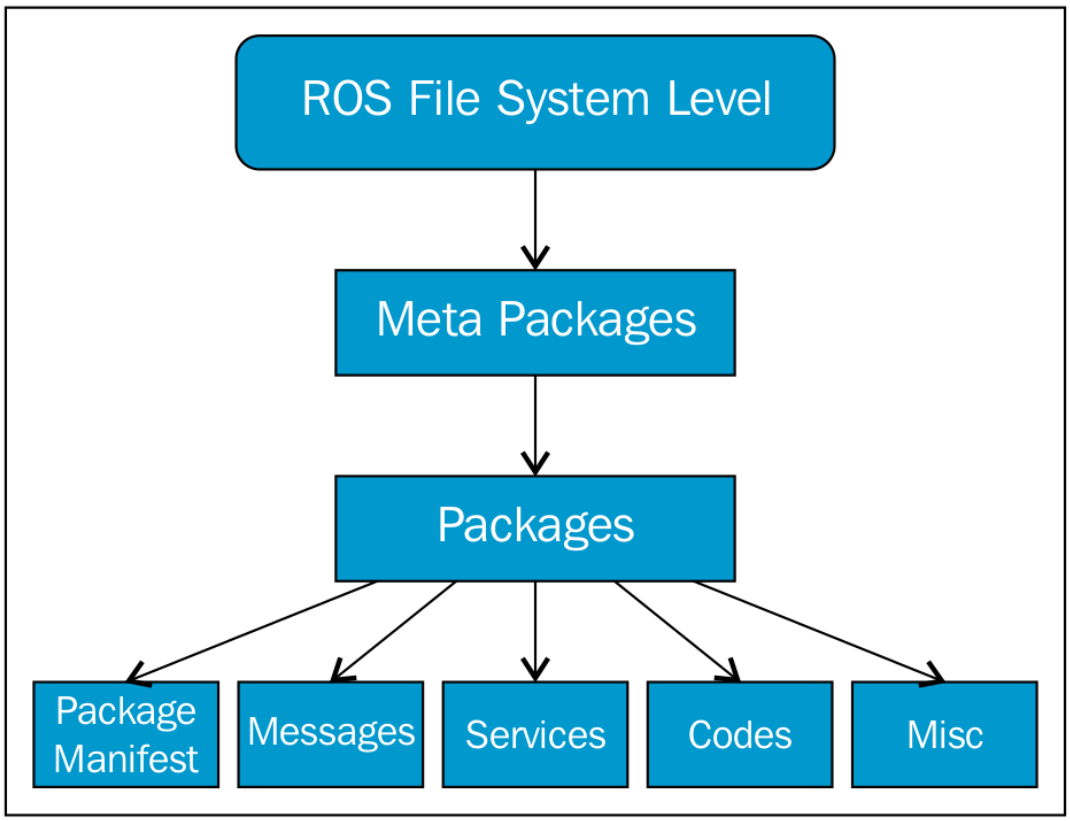


FIGURE 2.1. ROS system server Source:www.http://ros.wiki

- **Packages**: The ROS packages are the most basic unit of the ROS software. It contains the ROS runtime process (nodes), libraries, configuration files, and so on, which are organized together as a single unit. Packages are the atomic build item and release item in the ROS software.

- **Package manifest**: The package manifest file is inside a package that contains information about the package, author, license, dependencies, compilation flags, and so on. The package.xml file inside the ROS package is the manifest file of that package.

- **Meta packages**: The term meta package is used for a group of packages for a special purpose. In an older version of ROS such as Electric and Fuerte, it was called stacks, but later it was removed, as simplicity and meta packages came to existence. One of the examples of a meta package is the ROS navigation stack.

- **Meta packages manifest**: The meta package manifest is similar to the package manifest; the differences are packages included inside which as runtime dependencies and declare an export tag.

- **Messages (.msg)**: The ROS messages are a type of information that is sent from one ROS process to the other. We can define a custom message inside the msg folder of a package (my_package/msg/ MyMessageType.msg). The extension of the message file is .msg .

- **Services (.srv)**: The ROS service is a kind of request/reply interaction between processes. The reply and request data types can be defined inside the srv folder inside the package (my_package/srv/MyServiceType.srv).

- **Repositories**: Most of the ROS packages are maintained using a Version Control System (VCS) such as Git, subversion (svn), mercurial (hg), and so on. The collection of packages that share a common VCS can be called repositories. The package in the repositories can be released using a catkin release automation tool called bloom .

2.2.2 Important ROS Concepts

The following descriptions are included in order to provide a complete, self-contained description of the project implementation. Similar descriptions can be found on the official ROS website 1 , as well as in any book on ROS (for example or the more comprehensive.

The ROS Graph: A ROS system comprise a set of small programs that communicate with each other through messages. These programs become nodes in the ROS graph. The nodes communicate with each other by publishing and subscribing to topics that form the edges of the graph. A topic must have the format of one of the specific data types provided by ROS. For example, a node which receives temperature data from a thermometer, may publish the data as a topic on the ROS system with the type sensor_msgs/Temperature. There are many other data formats, e.g. velocity messages, geometry_msgs/Twist; images, sensor_msgs/Image; odometry messages, nav_msgs/Odometry and so on. Each node in the graph are typically POSIX processes, and the edges are TCP connections.

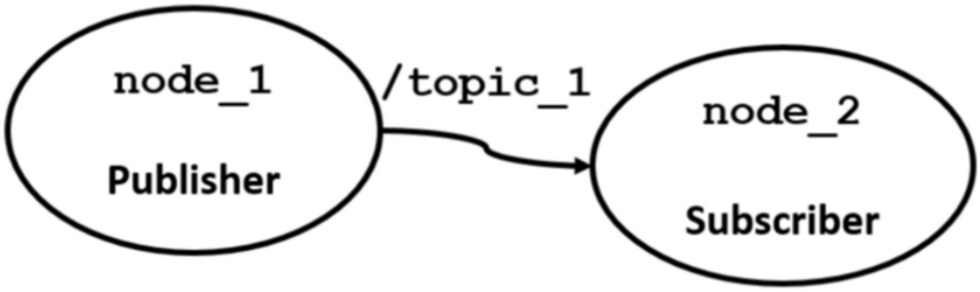


FIGURE 2.2. A minimal ROS graph. There are two nodes, node_1 and node_2. node_1 publishes data, i.e. a topic, by the name topic_1. node_2 can receive the data by subscribing to topic_1.

roscore: roscore is an essensial part of any ROS system as it enables nodes to communicate with each other. An instance of roscore must be started before launching any nodes. When a node is started, it will inform roscore of which topics it publishes and which topics it wish to subscribe to. Then, roscore will provide the information which allows the node to form a peer-to-peer connection to other nodes.

tf: tf is a coordinate system transformation library used in ROS. Parts of a ROS system can listen to broadcasted transforms in the form of messages, tf/tfMessage, which describe a coordinate system transform between one or more parent-child link pairs. tf also provides timing information it the messages. This is a very important feature because a node may depend on synchronized sensor streams with many different coordinate frames. Comparing a laser scan with a point cloud that was received seconds ago may lead to errors.

Project Structure and the catkin Build System

A ROS project will usually utilize the catkin build system. catkin replaces Rosbuild which is used for ROS Fuerte and earlier (this project uses ROS Kinetic). The source code in a ROS system is organized into packages. Each package provides a specific functionality to the system. Some packages can be downloaded and installed from a remote repository, while other packages will be created by the in-house developers for their specific robotic system. In this project, locally created ROS-packages were placed into a catkin workspace. This workspace contains the original source code and build specifications. Implementation specific details are provided in chapter 4. A general workspace structure is as follows:

```
workspace_folder/
src/
CMakeLists.txt
package_1/
CMakeLists.txt
package.xml
...
package_n/
CMakeLists.txt
package.xml
devel/
...
build/
...
```

Packages are build by running catkin_make from the command line. Each package comes with two files: CMakeLists.txt and package.xml. In CMakeLists.txt, a developer can link to additional libraries, e.g. OpenCV. The package.xml file, also known as the package manifest, contains a

description of the package and its dependencies. The developer can specify details such as version number, licenses and contact information to the responsible maintainer.

roslaunch

roslaunch is a ROS package tool used to launch multiple nodes from a single command line. This is useful for larger projects with many nodes, interactions and parameters. Exactly which nodes to launch is defined in XML-files with the .launch extension. In a launch file, the developer can group nodes together, pass arguments to the nodes and launch other launch files. Launch files can be launched from the command line as follows:

```
roslaunch <package name> <launch file name>.launch <argument1>:=true
```

2.2.3 An Overview of ROS-Related Tools

Robot Modelling In URDF URDF is an XML-like format for describing robots. The robot description is made up of links and joints. Each link description contains information of, e.g., its shape, inertial tensor, collision boundaries and other attributes. The links are connected to each other by joints[6].

Visialization in Rviz

rviz is an invaluable tool for visualizing on-line robot behavior. Simply put, rviz is created to visualize what the robot sees, and how it plans ahead. Many of the images in the following chapters were obtained in rviz.

Simulation in Gazebo

Gazebo 7 is a rigid body real-time simulator with good interfaces to ROS. A developer can build a robot model by using URDF, and spawn this model into a virtual 3D world in Gazebo. The simulator is integrated into ROS by using the ros_gazebo package. This simulator was used extensively over the course of this project, and allowed testing of all the implemented features, including SLAM and navigation.

2.2.4 Notable Robots Running ROS

PR2 - Personal Robot 2 PR2, developed by Willow Garage is one of the first robots designed to run ROS, and also one of the most advanced and capable robots with ROS today. PR2 is build for research and development of service robot applications. The navigation stack used in this thesis has been tested on the PR2. [4] describes how the PR2 used the navigation stack to autonomously navigate 42 km (26.2 miles). PR2 is available for sale at the price of 280,000.004(2016).

TurtleBot TurtleBot is a cheaper ROS-ready alternative to PR2. It is consists of a mobile base with differential drive, and a shelf system for mounting laptop computers and sensors.

Robonaut 2 Robonaut 25, a dexterous humanoid robot, currently resides within the International Space Station (ISS) roughly 400 km above the earth's surface. In 2014, a SpaceX Dracon capsule brought ROS as well as a pair of legs for Robonaut up to the ISS. Robonaut is designed for

research on how robots can support the crew in maintaining and operating the space station. A potential application of Robonaut is to perform extra vehicular activities and other maintenance tasks, thus freeing up valuable time for the crew.

Industrial Hardware The ROS-industrial program[15] provides hardware interfaces to various industrial equipment. An example is ABB's IRB-2400, where ROS-industrial provides package for motion planning software (MoveIt!) and trajectory downloading[6].

2.3 Simultaneous Localization and Mapping(SLAM)

SLAM, also known as Concurrent Localization and Mapping (CLM), is a class of solutions for determining an agents location and pose in an unknown environment, while simultaneously mapping the same environment.

SLAM problems arise when the robot does not have access to a map of the environment; nor does it have access to its own poses. Instead, all it is given are measurements $z_{1:t}$ and controls $u_{1:t}$.

There are two main forms of the SLAM problem, both are equally practical important. One is known as the online SLAM problem: It involves estimating the posterior over the momentary pose along with the map:

$$p(x_t, m | z_{1:t}, u_{1:t}) \tag{2.1}$$

Here x_t is the pose at time t, m is the map, and $z_{1:t}$ and $u_{1:t}$ are the measurements and controls, respectively. This problem is called the online SLAM problem since it only involves the estimation of robot pose at time t.

The second SLAM problem is called the full SLAM problem. In full SLAM, we seek to calculate a posterior over the entire path $x_{1:t}$ along with the map, instead of just the current pose x_t. In other words, with full SLAM we can visualize the trajectory of robot.

$$p(x_{1:t}, m | z_{1:t}, u_{1:t}) \tag{2.2}$$

Fig.2.3 shows the overview of SLAM problem, u_t is the command we gave to the robot, or prediction data. while z_t is data robot get from laser scan or Kinect-camera, or we can call them measurement data. From these inputs we can create a map and estimate the robot pose in this map.

SLAM is a hard problem to solve because the mapping between observations and the map is unknown, map and pose estimated correlate with each other. In addition, picking wrong data associations can have catastrophic consequences because robot will update its position based on wrong information. A key feature in SLAM is detecting previously visited areas to reduce map errors, a process known as loop closure detection which will also be covered in this thesis.

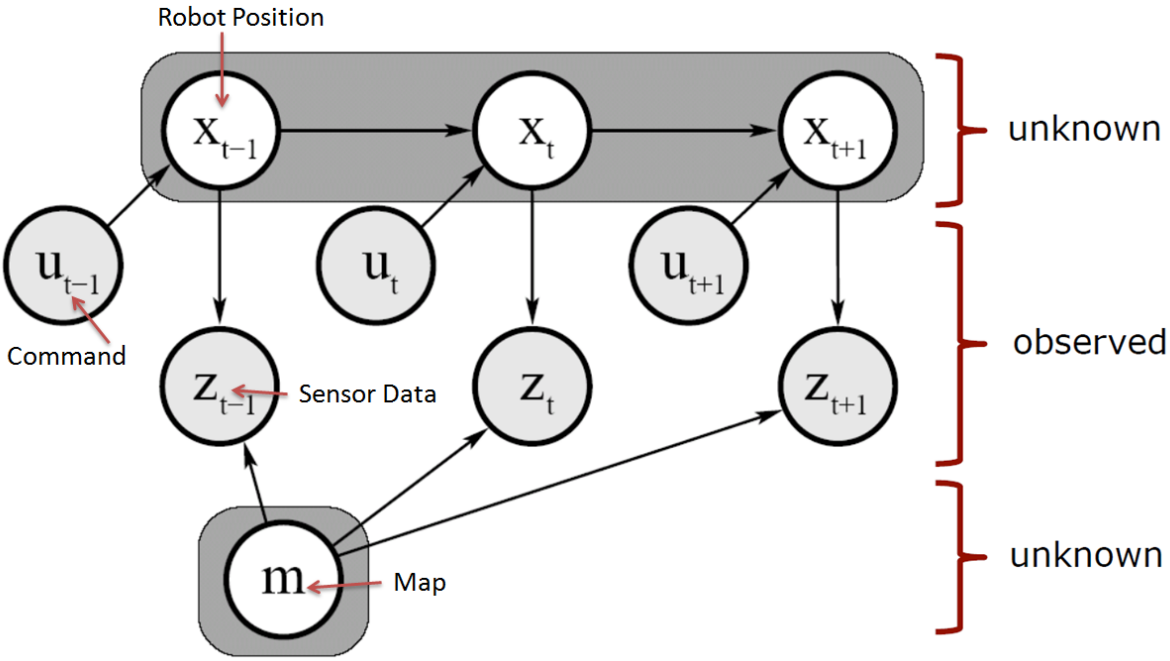


FIGURE 2.3. In SLAM, Robot pose x_t and map m are estimated base on measurement z_t and controls u_t.[13]

In this thesis, **RTAB-Map** which used RGBD-SLAM algorithm will be studied and implemented in a mobile robot "Volksbot". The algorithm represents nodes as poses and links as odometry and loop closure transformation. It is suitable for mapping in large-scale environment and multi-session[5].

Parallel, **Hector SLAM** which bases on LIDAR scan aligned with the horizontal plane is implemented and tested on Volksbot. It will be compared with RTAB-Map or combined the result with RTAB-Map for a more accurate result.

2.3.1 RTAB-MAP

RTAB-Map is developed by IntRoLab at Universite de Sherbrooke in Canada. It is a SLAM system developed for long term operations in large environments. RTAB-Map is distributed as a ROS package and supported on ROS Hydro, Indigo and Kinetic. It can be used with a hand-held RGBD or a stereo camera as well as with a camera and a laser range-finder placed on a robot. The source code and ROS wrapper is currently maintained, and new features and bug-fixes are added regularly. In this thesis, RTAB-Map is the core feature, besides navigation, that has been integrated into the robot named Volksbot. Some factors which motivated the use of RTAB-Map are:

- It is a RBGD-SLAM method which requires an RGB-D sensor, for example a Kinect.
- It includes 3D obstacle detection.
- It has a memory management system intended for large scale multi-session mapping.
- RTAB-Map can be used for object detection. This can be done by linking RTAB-Map to OpenCV and the non-free feature detectors Scale-invariant feature transform (SIFT) and Speeded Up Robust Features (SURF)[7].

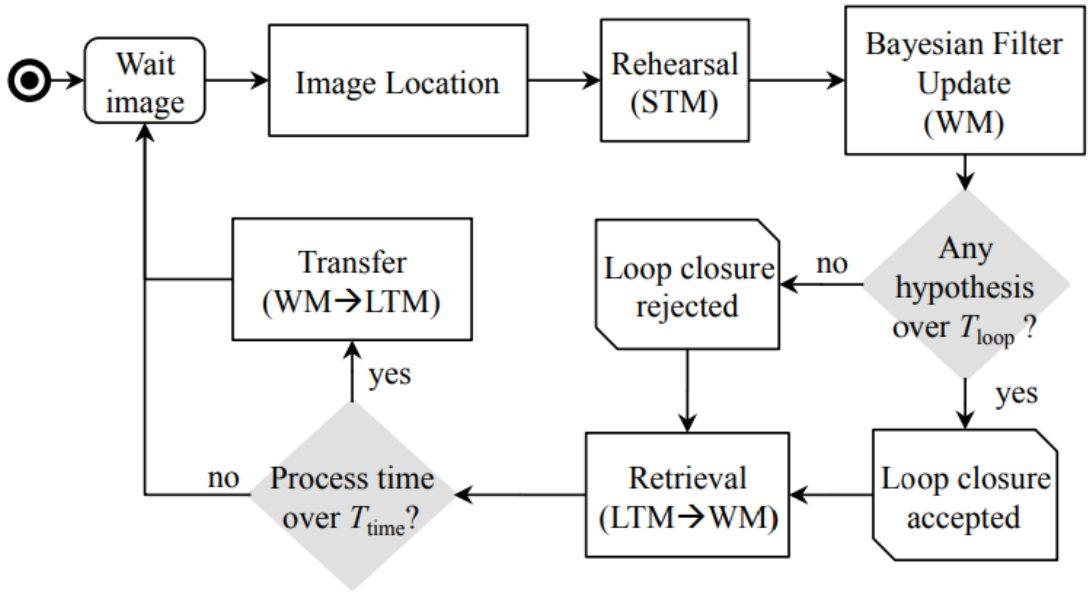


FIGURE 2.4. Architecture of the RTAB-Map memory management loop closure detection steps. [11]

Figure 2.4 shows the RTAB-Map memory management loop closure detection steps. To briefly describe, it has two distinctive solutions to the SLAM problem: Visual loop closure detection and a memory management system for large data sets. It uses the local appearance descriptor for description and a Bayes filter for evaluating the loop closure hypothesis. The map is constructed by linking new acquired images with previous ones based on the loop closure probability values and it is fully incremental. RTAB-Map uses (a) Working Memory (WM) - to keep the most recent and frequent observed locations (b) Long Term Memory (LTM) - storing other observed locations. When a match is found between the current location and one stored in WM, associated locations stored in LTM will be remembered and updated and (c) Short Term Memory (STM) - storing the poses and retrieving them from the long term memory when loop closing is required. As a result,

RTAB-Map generates a 3D point clouds of the environment for visualization and creates a 2D occupancy grid map for navigation. A more detailed descriptions of the loop closure detection and memory management approach is provided in session 2.3.1.2, while the SLAM method is presented in reference [18].

Generally, RTAB-Map allows to use an odometry from visual odometry node (provided in RTAB-Map package), which extracts from Kinect camera, however that estimation may fail due to the low number of detected features. Therefore, in this thesis, RTAB-Map gets the odometry from wheels (or combination of the two) to have a safer and more robust result as indicated in figure 2.5. The graph shows that the used inputs include scan data from laser scanner, visual odometry from Kinnect Camera and wheels odometry from robot's wheel encoder. The output is a 3D-map and robot position in this map, finally, all information will be display by ROS visualization program(Rviz)

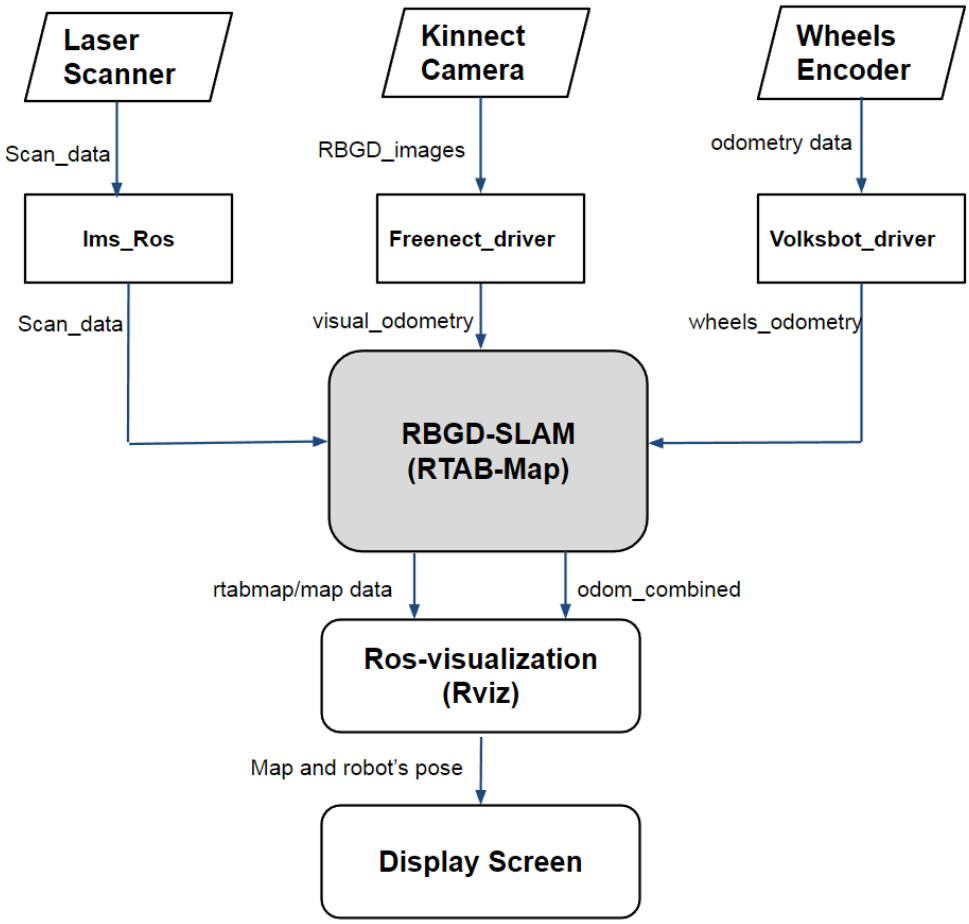


FIGURE 2.5. Architecture of the RTAB-Map include input from Kinnect camera, robot odometry to RTAB-Map and output is map and robot pose .

2.3.1.1 Graph base mapping

RTAB-Map uses a graph to represent the map and every nodes in the graph corresponds to a pose of the robot during mapping. New locations L_t, represented by nodes, are continuously added to the system's working memory as time passes[1]. In this method, the graph edges are referred to as links. There are two types of links: neighbor links and loop closure links. Each node is a location in the map, and the links contain geometrical transformation between the node locations. Figure 2.6 illustrated the graph concept.

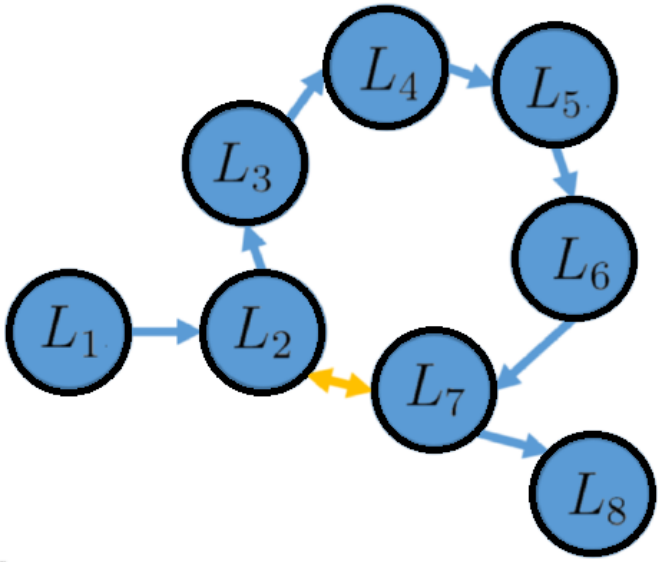


FIGURE 2.6. Conceptual illustration of a graph created by RTAB-Map over time $1 \leq t \leq 8$. A loop closure hypothesis was accepted at t = 7, as shown by the yellow arrow. Feature descriptors in L_2 and L_7 are sufficiently similar to accept this as a loop closure.

2.3.1.2 Loop Closure Detection

At regular predefined intervals, RTAB-Map will take a snapshot image I and run this image through a feature detector (e.g. SURF or SIFT). The obtained image descriptors will be quantized to a corresponding descriptor vocabulary, i.e. a bag-of-words[13]. The bag of words constitutes the signature of location L at which the image was taken. When the new location L is added to the graph, RTAB-Map will update a posteriori probability density function for each loop closure hypothesis S by means of a Bayes filter. A loop closure hypothesis is accepted if the belief that the robot is at a new location is below a threshold T loop . Then a link in the traversal graph will be created to represent a closed loop as.[16]. This process is summarized in figure 2.7.
In RTAB-Map, the RGB image, from which the visual words are extracted, is registered with a depth image, i.e., for each 2D point in the RGB image, a 3D position can be computed using the

calibration matrix and the depth information given by the depth image. The 3D positions of the visual words are then known. When a loop closure is detected, the rigid transformation between the matching images is computed by a RANSAC(Random Sample Consensus)[9] approach using the 3D visual word correspondences. If a minimum of I inliers are found, loop closure is accepted and a link with this transformation between the current node and the loop closure hypothesis node is added to the graph. If the robot is constrained to operate on a single plane, the transformation can be refined with 2D iterative-closest-point (ICP) optimization [18] using laser scans contained in the matching nodes.

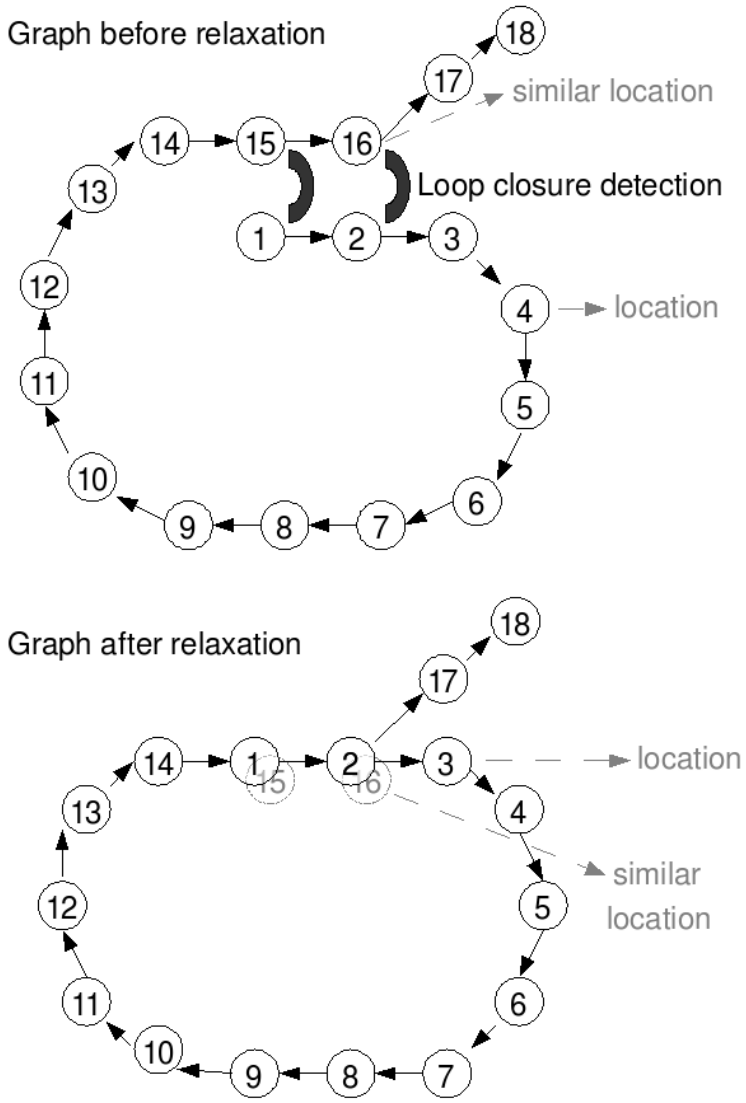


FIGURE 2.7. A loop closure hypothesis was accepted at t = 15 and 16. Feature descriptors in 15 and 1, 16 and 2 are sufficiently similar to accept this as a loop closure.

2.3.2 Hector SLAM

Hector SLAM [22] is a 2D SLAM approach capable of 3D motion estimation. In its original form, the method is suitable for the systems with low-end computational power and size, and for mapping small environment. 2D SLAM is based on LIDAR scans aligned with the horizontal plane. The full Hector SLAM implementation consists of a 2D SLAM subsystem loosely coupled and synchronized with an Extended Kalman Filter (EKF) used as a 6DOF pose estimator.

The 2D pose of the LIDAR is estimated through a scan matcher, the process of aligning the current LIDAR scan with the map generated over the past steps. To take advantage of this 2D map which usually be considered to be more precise, in this thesis, the scan matcher of Hector SLAM will be used to provide the odometry for RTAB-Map. Hector SLAM can be downloaded and used in ROS in the form of a pre-build package, hector_slam [1].

2.3.2.1 Laser Scan matcher

In Hector SLAM the 2D pose of the LIDAR is estimated through Laser scan matcher. The laser_scan_matcher package is an incremental laser scan registration tool [3]. It is written to run on ROS. The package allows to scan match between consecutive sensor_msgs/LaserScan messages, and publishes the estimated position of the laser as a geometry_msgs/Pose2D or a tf transform. The package can be used without any odometry estimation provided by other sensors. Thus, it can serve as a stand-alone odometry estimator. Alternatively, users can provide several types of odometry input to improve the registration speed and accuracy[20]. The more information in scan_matching theory are describe in "Probabilistic Robotic book"[2], in fig.2.8 show how scan matching algorithm works.

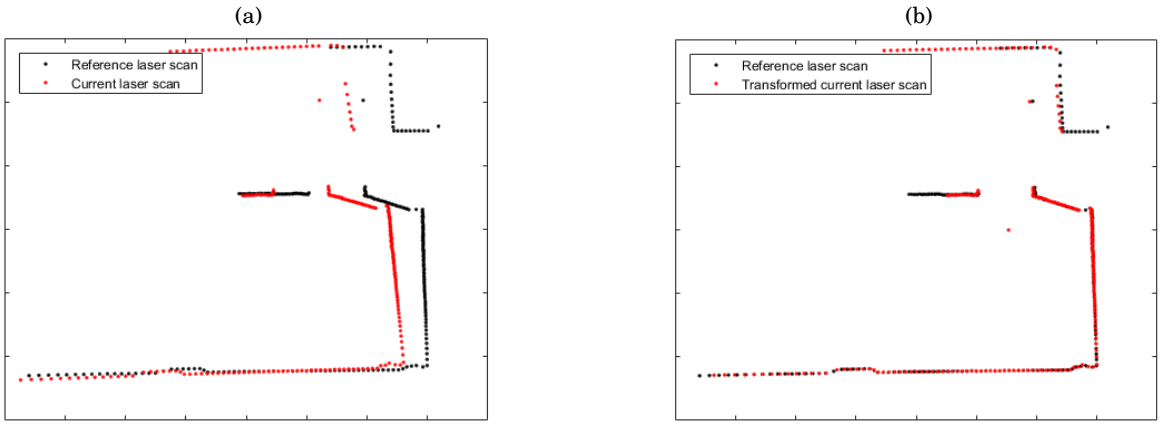


FIGURE 2.8. (a) Reference laser scan and current laser scan before matching, (b) Reference laser scan and current laser scan after run Scan Matching Algorithm . Source:https://de.mathworks.com

[1] http://wiki.ros.org/hector_slam

2.3.3 Occupancy Grid Map

Hector SLAM and RTAB-Map used Occupancy Grid Map to represent the environment for navigation. This kind of map is a location-based map and rasters the world into discrete cell with set size (resolution with meter/pixels). Using Occupancy Grid Maps, the real world will be represented by the three different kinds of space:

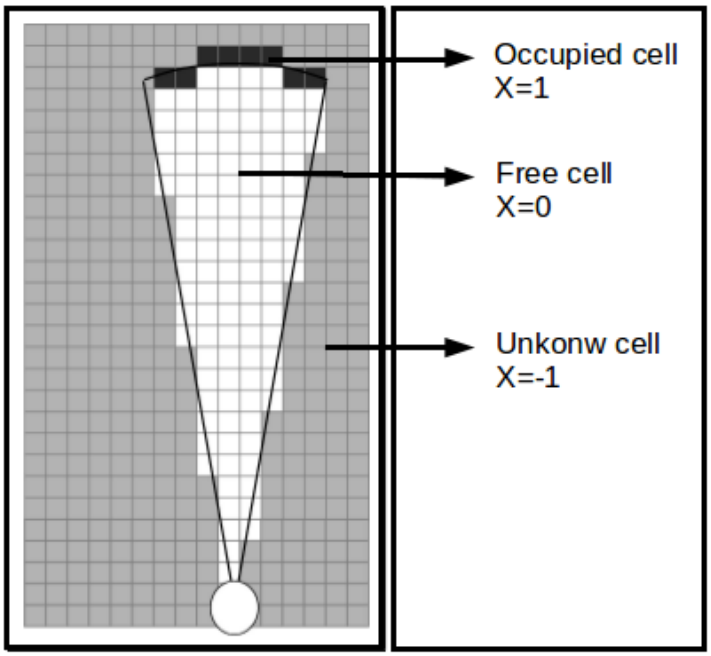


FIGURE 2.9. The darkness of each grid cell corresponds to the likelihood of occupancy. The lightness are free space and the gray is unknown space.

- **Free space**: Free space is the space in the real world, where the robot can move without colliding with an obstacles. Free space is represented by white pixel in a visualization of an occupancy grid map.

- **Occupied space**: Occupied space describes the space in the map where obstacles like walls or doors are in the real world and the robot cannot pass without collision. Visualizing this kind of space in any occupancy grid map, the pixel are black.[17]

- **Unknown space**: Unknown space describes the space in a map, if it is not explored and no information about it state is available. This kins of space represented as gray pixels for a visualization of an occupancy grid map. This case is helpful for exploring an unknown environment.

Figure 2.9 shows an example for a recorded occupancy grid map with the different types of space. The white space is free, the black space occupied by obstacles and the light gray colored space is unexplored.

Internally, an occupancy grid map can be represented as one or two dimensional arrays with the corresponding values, 0 for free cell and 1 for occupied cell. For the case that a cell is not explored, the value of the cell is -1.

2.3.4 An comparison between two SLAM methods

Table 2.1 shows the comparison between Hector-SLAM and RTAB-Map. The Hector-SLAM does not play the main role in this thesis because it is not able to incorporate wheel odometry and cannot autonomously navigate. However, Hector SLAM can use for mapping because Hector map is more accurate.

As can be seen from the Table 2.1 the best option is RTAB Map - mainly for the possibility of using the odometry from Volksbot (possibly combined with the visual odometry provided by Kinect-camera) and publishing all required data through ROS topics.

Hector SLAM	**RTAB Map**
+Fast update for the map	- Slower but it can provide a fast visual odometry
+ Can re-localize after looking at known position	- Problems with re-localization, but it can use the wheel odometry.
+ Well readable map, also published through ROS	+ Well readable map, also published through ROS
+ All important data is published through ROS topics	+ All important data is published through ROS topics
- Cannot utilize the odometry provide by robot, but can generate odometry from laser scans.	+ Can utilize the odometry provided by the Volksbot robot
+ Wide range of parameters	+ Wide range of parameters

TABLE 2.1. Comparison between Hetor-SLAM and RTAB-Map

2.4 Navigation Stack Package

ROS provides a pre-built navigation stack for 2D navigation. The navigation stack can plan a path and send velocity commands to the mobile base controller based on the sensor input, a goal pose, a map and the frame of the base_link. Figure 2.10 shows how the internal components of the node move_base, and how it interacts with the rest of the ROS system. The navigation stack

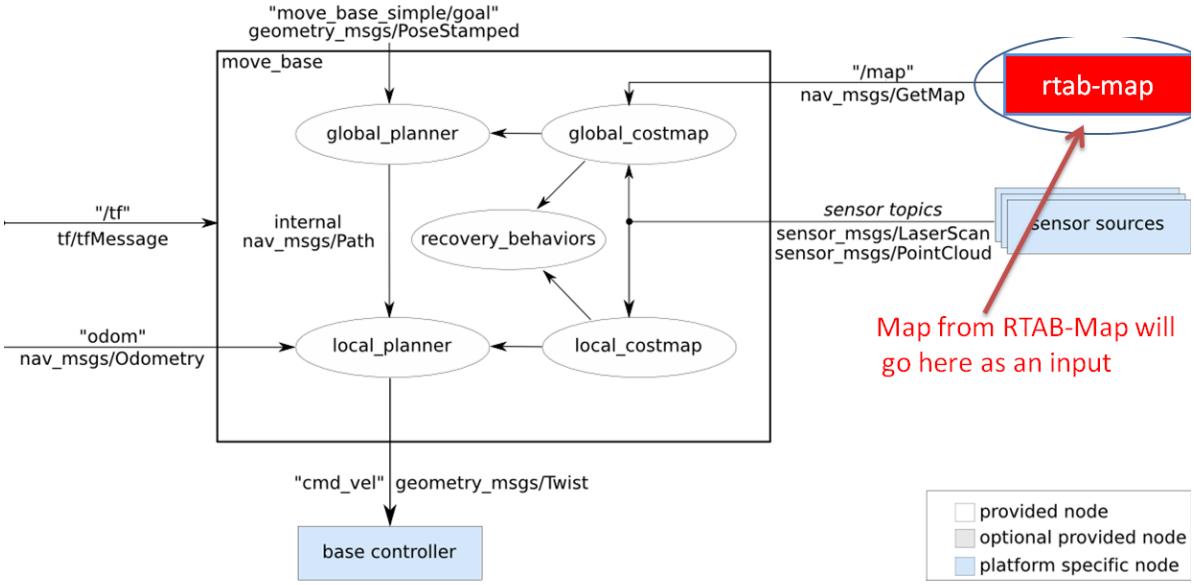


FIGURE 2.10. move_base and the navigation stack, the map from RTAB-Map is provided here as an input. (Image credits: ros.org)

publishes velocity commands for translation and rotation in the xy-plane. Velocity commands are published in the form of a geometry_msgs/Twist message with the default topic name "cmd_vel". Mobile bases must be constructed as either holonomic or differential drives. The following two subsections introduce the global and the local costmaps respectively. These costmaps are in fact the grid maps where each squares on the grid can be viewed as a node in a graph which is used by path finding algorithms to search for the shortest path. These maps are both based on the ROS package costmap_2d. This package will inflate an area in a radius around the detected obstacles, in which the traversal is associated with a certain cost. By inflating the obstacles, in some sense similar to increase their size, the path planners can now treat the entire robot as a single point. The point represents the origin of the robots base frame.

In this thesis, Navigation Stack receives maps from RTAB-Map and scan data from Laser Scanner as well as wheels odometry from wheels encoder to produce the command velocity to the robot for running on the path planning provided by the global planner. Figure 2.11 helps to illustrate this process in a more concrete way. Robot can also control directly from keyboard by use volksbot_teleop package, this is ROS package used to sent command to cmd_vel topic to control the robot.

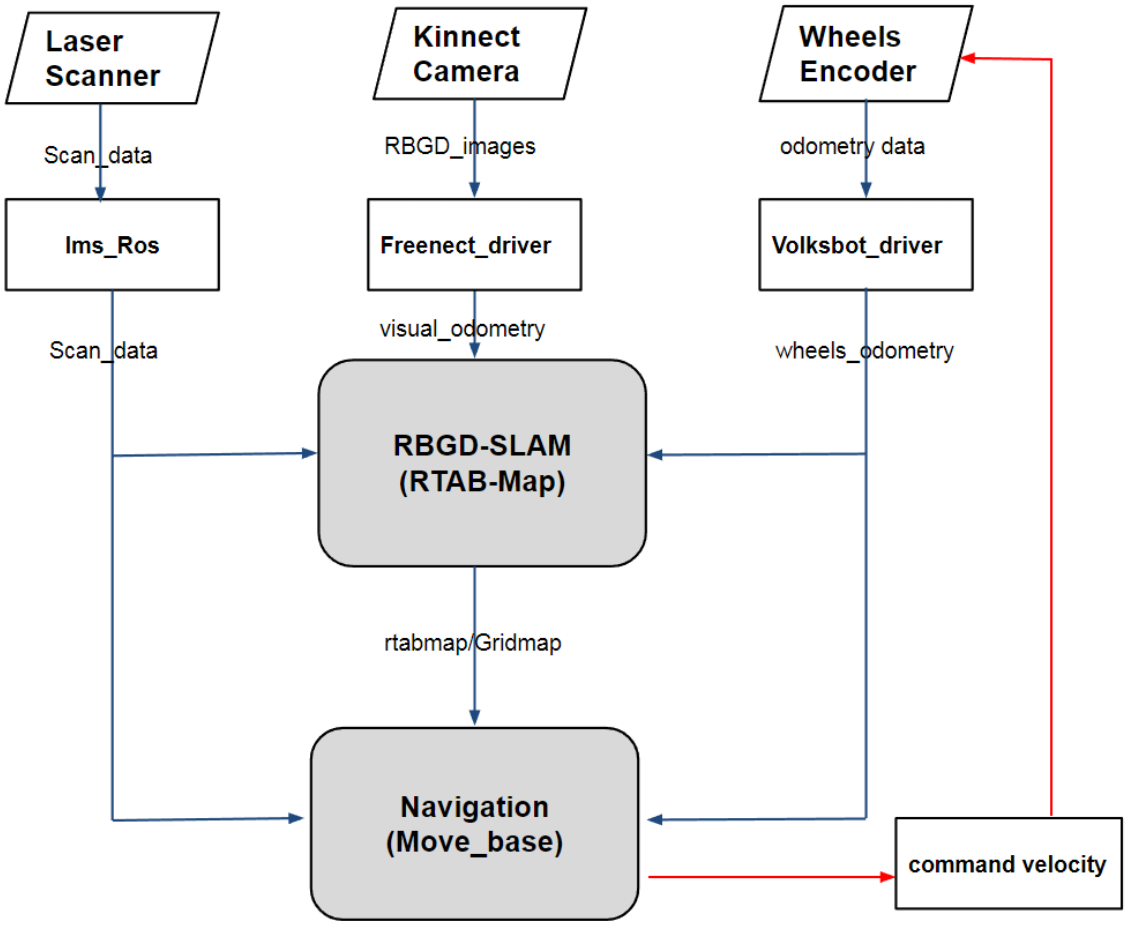


FIGURE 2.11. An overview how navigation stack work in this thesis, the input to navigation stack include scan data, grid map and wheels-odometry, output is the velocity command for robot

2.4.1 Global and Local Costmap

This is a ROS package in Navigation Stack, which provides an implementation of a 2D costmap taken sensor data from the world, builds a 2D or 3D occupancy grid of data, depending on whether a voxel based implementation is used, and inflates costs in a 2D costmap based on the occupancy grid and a user specified inflation radius. These packages also provide support for map_server base initialization of a costmap, rolling window based costmap, and parameter based subscription to and configuration of sensor topics.

The costmap_2d package provides a configurable structure that maintains information about where the robot should navigate in the form of occupancy grid. The costmap uses sensor data and information from the static map to store and update information about obstacles in the world through the costmap_2D::Costmap2DROS object. The cost map 2D object provide a purely two

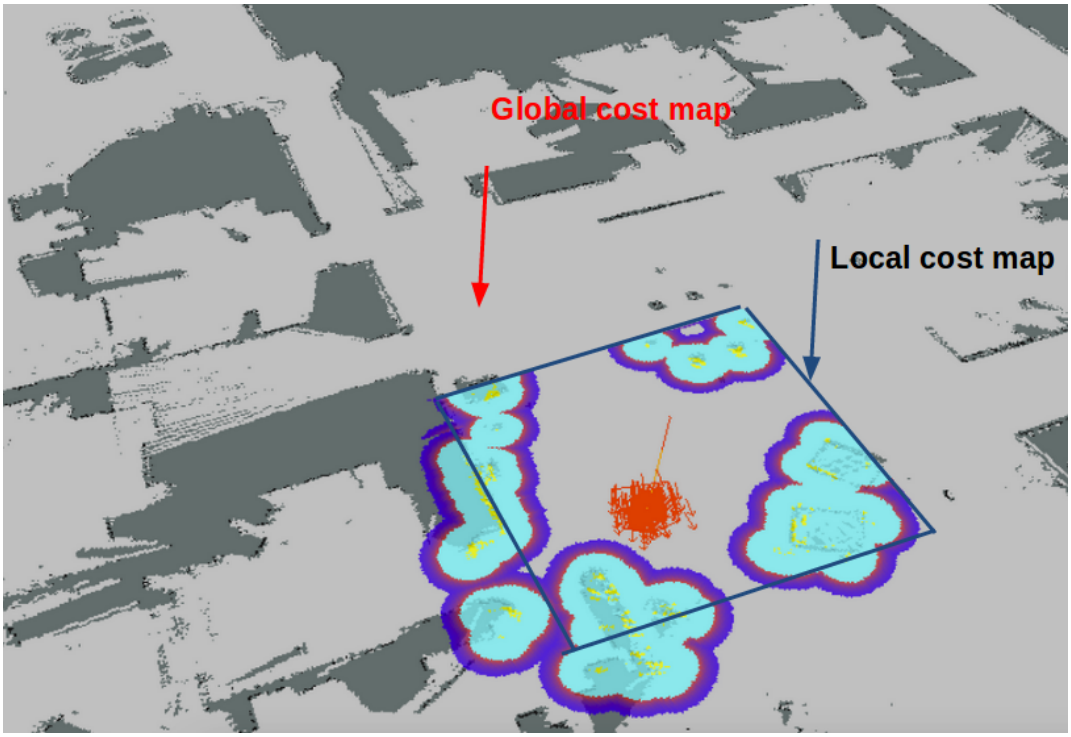


FIGURE 2.12. The purple cells represent obstacles in the costmap, the blue cells represent obstacles inflated by the inscribed radius of the robot, and the red square represents the footprint of the robot. For the robot to avoid collision, the footprint of the robot should never intersect a purple cell and the center point of the robot should never cross a blue cell

dimensional interface for it user, meaning that query about obstacles can only made in columns. For example, a table and a shoes in the same position in the X,Y plane, but with different Z positions would be result in the corresponding cell in the Costmap_2D, object costmap having identical cost value. This is designed to the help planning in planar spaces.

The main interface is costmap which maintains much of the ROS related functionality. It contain a costmap_2D. which use to keep track of each of the layer. Each layer instantiated in the costmap2DROS using pluginlib and is add to layered costmap. The layer them self may combined individually allowing arbitrary change to the costmap to be make through the C++ interface.

2.4.2 Global Planner

The ROS navigation stack is equipped with a basic global planner. This global planner is fitted with a set of basic path planning algorithms: e.g. Dijkstra and A*, where Dijkstra is the default option. Global path planning is performed on the global costmap which in turn based on the 2D grid map published by a map server. A pre-mapped area is not strictly necessary for the global planner to plan a path. Instead, it will just plan a naive path which can be corrected by the local

FIGURE 2.13. Global planner displayed in Rviz. (Image credits: ros.org)

planner as the robot moves along the planned trajectory.

2.4.2.1 Dijkstra

The Dijkstra Algorithms takes the start cell and labels it as visited. Up from this cell all neighbors are added to the list, which are saved as already inspected cells but not visited yet and calculate the cost g(x) to travel from this point to another. In the next step, the algorithm picks the cell with the minimals cost g(x) and labels it as a visited one. All neighbors of this cell are labeled as inspected, including those had not yet been labeled as visited or inspected before. If the goal is reached, the search will break and the minimal costs are known. And the robot can follow the edges pointing towards the lowest edge cost.

2.4.2.2 A*-Algorithm

The main difference between the Dijkstra and A* is that not only the costs $g(x)$ are used to calculate the optimal path, an additional heuristic $h(x)$ will also be used. Therefore, the cell with the minimum heuristic will be inspected next. Importance is that A* only works, if $h(x)$ does not underestimate costs to the goal. A widely used heuristic is the Euclidean distance. The rest of

the algorithm does not change. The idea of a heuristic makes the A* algorithm faster then the Dijkstra. It can be seen that the Dijkstra algorithm is a special case of the A* algorithm with $h(x)$ = 0.

2.4.3 Local Planner

Actual velocity commands from move_base are published by the local planner. The local planner will receive the global plan from the global planner, and calculate a new trajectory based on currently observed obstructions as well as the robot footprint and kinematics. Local obstructions are expressed in a dynamic grid map, i.e. the local costmap, which is based on a subset of the global map combined with real time sensor data. The dynamic local cost map enables the robot to avoid temporary and moving obstacles. The local planner can use either the Trajectory Roll-out or Dynamic Window Approach (DWA) algorithm for trajectory planning.

2.4.3.1 Marking and Clearing

The costmap automatically subcribles to sensor topic over ROS and updates itself accordingly. Each sensor is used to either mark (insert obstacle information to the costmap), clear(remove obstacles information from the costmap), or both. A marking operation is just an index into an array to change the cost of a cell. A clearing operation, however consists of raytracing through the grid from the origin outward of each observation reported. If a tree dimensional structure is used for obstacle information, obstacle information from each column is projected down into two dimensions to put into the costmap.

2.4.3.2 Occupied, Free, and Unknown Space

While each cells in the costmap can have one of 255 different cost values, the its underlying structure is capable of representing only three. Specifically, each cell in this structure can be either free, occupied, unknown. Each status has special cost value assigned to its upon projection into the costmap. Columns that have a certain number of occupied cell are assigned as a lethal cost, columns whose certain number of unknown cells assigned as free_space cost.

2.4.3.3 Map Update

The costmap performs map update cycles at the rate specified by the update frequency parameter. Each cycle, sensor data comes in, marking and clearing operations are performed in the underlying occupancy structure of the costmap, and this structure is projected into the costmap where the appropriate cost values are assigned as discrible above. After this, each obstacle inflation is performed on each cell with a cost. This consists of propagating cost values outwards from each occupied cell out to a user-specified inflation radius. The detail of this inflation process are outlined below.

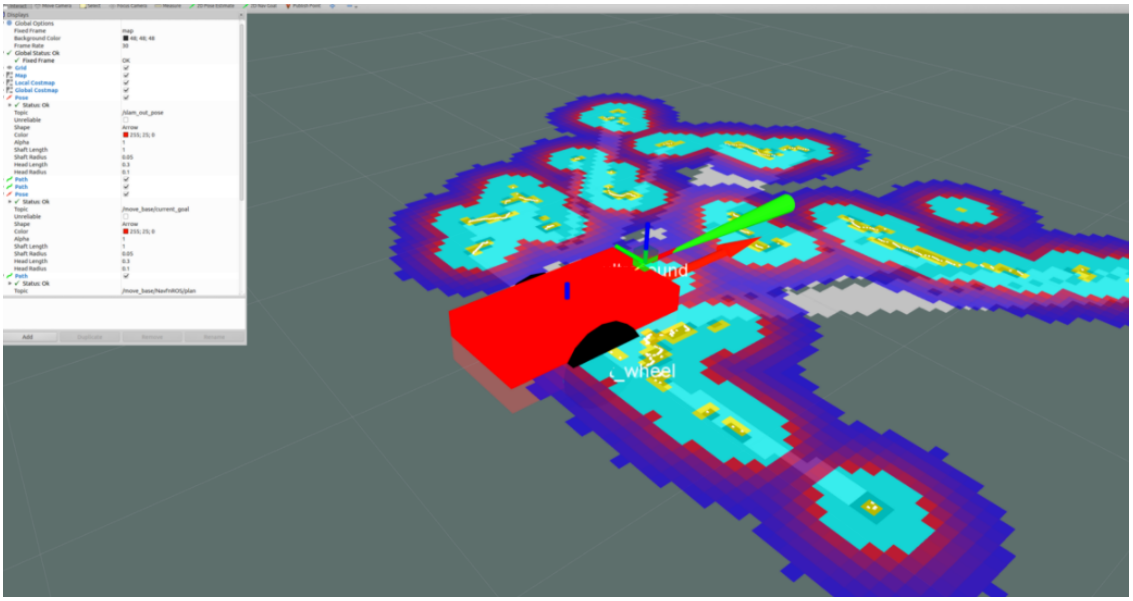


FIGURE 2.14. The robot will be stuck in inflated area of the obstacle if this parameter was set too large.

2.4.3.4 Inflation

Inflation is the process of propagating cost values out from occupied cells that decrease with distance. For this purpose, there are 5 specific symbols for costmap values as they relate to a robot.

- "Lethal" cost means that there is an actual obstacle in a cell. So if the robot's center were in that cell, the robot would obviously be in collision.
- "Inscribed" cost mean that cell is less than the robot's inscribe radius away from an actual obstacle. So the robot is certainly in collision with some obstacle if the robot center is in a cell that is at or above the inscribed cost.
- "Freespace" cost is assumed to be zero, and it means that there is nothing that should keep the robot from going there.
- "Unknown" cost means there is no information about a given cell. The user of the costmap can interpret this as they see fit.
- All other costs are assigned a value between "Freespace" and "Possibly circumscribed" depending on their distance from a "Lethal" cell and the decay function provided by the user.

The rationale behind these definitions is that we leave it up to planner implementations to care or not about the exact footprint, yet give them enough information that they can incur the cost of tracing out the footprint only in situations where the orientation actually matters.

2.4.4 Recovery Behaviors

When the robot becomes stuck, it can be configured to perform a set of recovery behaviors. The default recovery behaviors available to move_base are to clear the costmap, i.e. to remove obstacle information from the costmap, and to rotate in place, in the hope of discovering a new clear path. A typical sequence of behaviors is to clear the costmap, try to locate a path and then attempt an in place rotation before doing a second attempt to plan a path. If the entire series of recovery actions fails to reveal a clear path, the move_base node will abort and consider the goal state to be infeasible[8]. Fig.2.15 is the diagram describing in detail the recovery approach when robot thinks it is stuck somewhere.

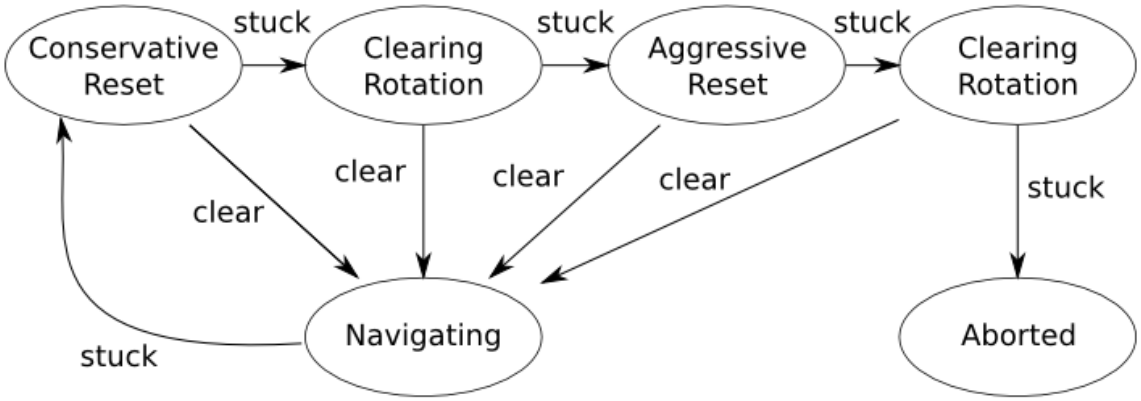


FIGURE 2.15. Summary of the executed behaviors to unstuck the robot. Source of figure: http://wiki.ros.org/move base.

2.5 Odometry-based Motion Model

An important aspect of SLAM is the odometry data. The goal of the odometry data is to provide an approximate position of the robot as measured by the movement of the wheels of the robot, to serve as the initial guess of where the robot pose might be. The difficult part about the odometry data and the laser data is to get the timing right. The laser data at some time t will be outdated if the odometry data is retrieved later. To make sure they are valid at the same time one can extrapolate the data. It is easiest to extrapolate the odometry data since the controls are known. It can be really hard to predict how the laser scanner measurements will be. If one has control of

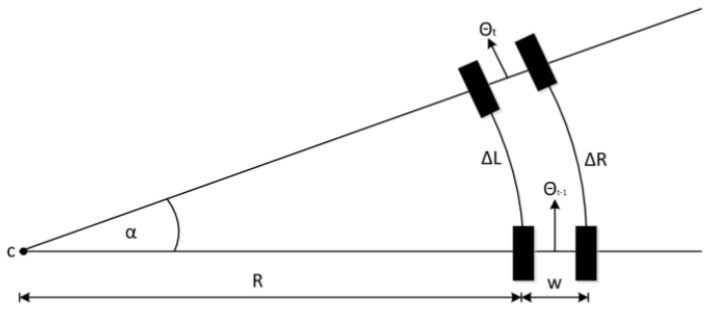


FIGURE 2.16. Illustration of the motion: robot moves forward and turns left.

when the measurements are returned it is easiest to ask for both the laser scanner values and the odometry data at the same time.

The robot's differential odometry provides the information, how many ticks the left and the right motor have done between time steps t_{-1} and t. The origin of the coordinate system is in between the two wheels for t_0. The last known pose of the robot is given by x_{t-1}, y_{t-1} and θ_{t-1}. The width between the centers of the two wheels ω is known. Also, the motor ticks between t-1 and t of the motor are given by $tick_{self}$ for the left motor and $tick_{right}$ for the right motor. It is assumed that the robot move forward and performs a left turn. For that movement, the right motor rotate with a higher rate than the left motor. The principle motion is shown in figure 2.16. Δ_L and Δ_R can be determined as follow

(2.3) $$\Delta_R = \alpha.(R + \omega)$$

(2.4) $$\Delta_L = \alpha.R$$

α and R are unknown in the both equations. The subtraction of equations 2.1 and 2.2 yield to:

(2.5) $$\Delta_R - \Delta_L = \alpha.R$$

Therefore

(2.6) $$\alpha = \frac{(\Delta_R - \Delta_L)}{\omega}$$

and

(2.7) $$R = \frac{\Delta_L}{\alpha}$$

R is only computable if α is not zero. If α is zero, the robot will move straight forward and the point c would be in infinity. This case will be describe later in this section. For the upcoming equations, α is assumed not to be zero. Δ_L and Δ_R can be also expressed as follows:

$$\Delta_L = tick_{left}.c_{mm} \tag{2.8}$$

$$\Delta_R = tick_{right}.c_{mm} \tag{2.9}$$

c_{mm} describe the factor to get driven distance in millimeters from the motor ticks. To calculate the new pose of the robot figure 2.1 indicate that the robot move around the center of movement c. The Catersian coordinate will be calculated by the following equation:

$$\begin{pmatrix} c_x \\ c_y \end{pmatrix} = \begin{pmatrix} x_{t-1} \\ y_{t-1} \end{pmatrix} - (R + \tfrac{\omega}{2}).\begin{pmatrix} \sin(\theta_{t-1} + \alpha) \\ -\cos(\theta_{t-1} + \alpha) \end{pmatrix}$$

The module operation in equation 2.10 is necessary to be sure that the healing is within the range of [0;2π]. If the ticks of the left motor equals the tick of the right motor, the robot move straight forward. Therefore, the calculation of the new pose is quite simple:

$$x_t = x_{t-1} + l.\cos(\theta_{t-1}) \tag{2.10}$$

$$y_t = y_{t-1} + l.\sin(\theta_{t-1}) \tag{2.11}$$

$$\theta_t = \theta_{t-1} \tag{2.12}$$

the unknown variable l is given by (only valid for the case: $tick_{left}$ equals $tick_{right}$:

$$l = tick_{left}.c_{mm} = tick_{right}.c_{mm} \tag{2.13}$$

The shown motion model is a noise free model. In practice, systematic and non-systematic errors will add noise which will result in variations from the calculated pose.

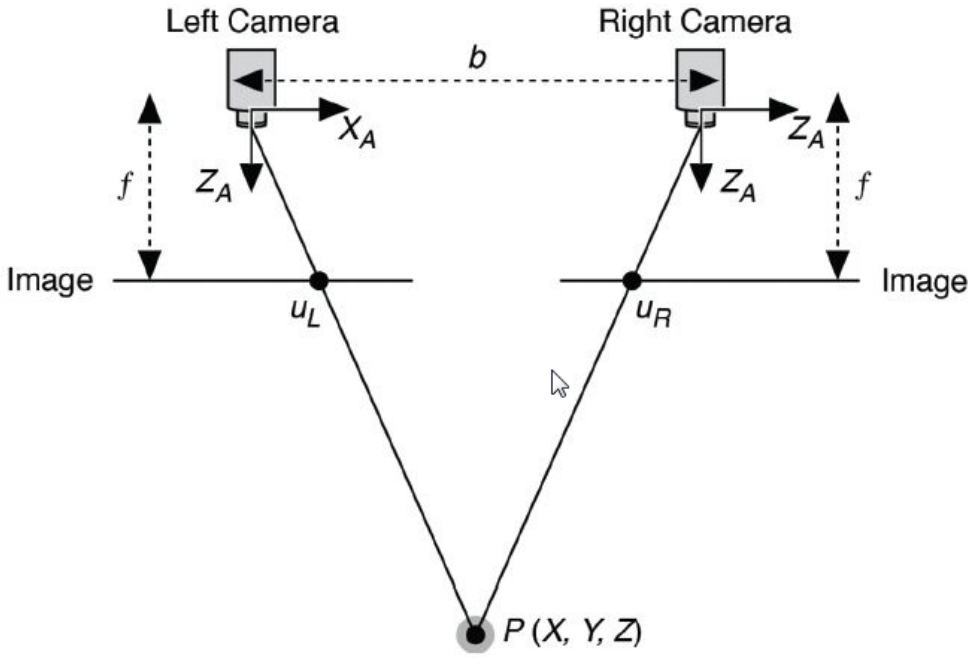


FIGURE 2.17. Depth of a point is given by the following formula: Depth = f * b/d, where f = focal length, b = distance between cameras, and d = the disparity or distance between corresponding points

2.6 Depth Cameras

A depth camera can be described as a regular color video camera with the ability to create spatial images. In the context of this thesis, a depth camera can more precisely be described as a RGB-D camera, where the letters RGB-D are short for red, green, blue and depth. In a regular RGB camera, a spatial scene will be projected onto a rectangular pixel grid where each pixel contains intensity values for red, green and blue colors. These pixel values represents the detected scene. A major problem with RGB cameras is the significant loss of information. The information loss is mostly a consequence of 3D to 2D projection and digital quantization. RGB-D cameras have the means to reduce this information loss by mapping the pixel values to spatial coordinates. The atomic parts in 3d images are usually represented as points in a point cloud or cubic volumes, also known as voxels.

Different variations of depth cameras will usually fall into one of two categories: active or passive. Passive sensors perceive the surroundings as it is, without actively interfering with the environment as a part of the sensing process. A typical passive RGB-D sensor is the stereo camera. Fig 2.17 describes how simple stereo vision system calculate the depth of a point. For depth cameras, the projection is usually in the form of laser or infra red light. The Kinect sensor used in this project is an example of an active RGB-D sensor.

CHAPTER 3

HARDWARE AND SOFTWARE

The following section gives an overview of given hardware and software, which have been used to implement and test algorithms for SLAM problem. The coordinate systems of the different sensors and the coordinate system of the robot will be described, too.

3.1 Hardware

3.1.1 Volksbot RT3

The Volksbot RT3 of the Frauenhofer IAIS will be used. The Volksbot RT3 consist two actuated wheels and two castor-wheels. The motors for the two actuated wheels can be controlled via a C++ software library. Figure 3.1 show different view of the volksbot RT3.

Looking from behind onto the control panel of the robot, the mapping of the motors is as follows: The left motor is the motor on the left hand side and the right motor is the motor on the right hand side. The dimensions of the Volksbot RT3 are shown in figure 3.1 and summarized in table 3.1.

Dimension of Robot	**Value**	**Unit**
Length	580	mm
Width	520	mm
Height	315	mm
Wheel seperation	425	mm

TABLE 3.1. Summary of important dimensions of the Volksobot.

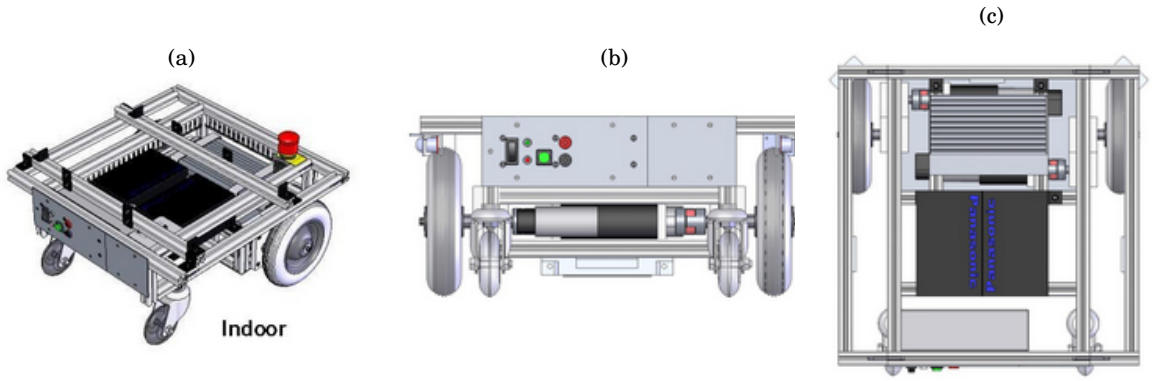


FIGURE 3.1. Different views of the Volksbot indoor from Frauenhofer IAIS.

3.1.2 Wheel encoder

The volksbot RT3 uses a differential odometry to determine the pose of the robot. The odometry provides how many ticks both motors have done during a time interval Δt. The count of the motor ticks can be be determined by reading the current encoder value for every motor.

To handle the differential odometry of the Volksbot RT3, the volksbot_uos_driver[1] is used. This driver defines a right orientated coordinate system. In section 3.3.2, the coordinate system of the odometry is defined as left orientated. Therefore, the driver is adapted to the left orientated coordinate system.

3.1.3 Kinect for Xbox 360

Kinect for Xbox 360 is the RGB-D sensor used in this project. The Kinect sensor is equipped with a depth sensor, a regular color camera, a microphone array and a tilt motor(figure 3.4). The color camera in combination with the depth sensor forms what is usually referred to as a RGB-D sensor[11], i.e. a combined color and depth camera.

Kinect Hardware Specifications

Sensor specifications are given in table 3.2.[21] Note that the range values may differ from what is available in Microsoft's own SDK, as the shortest distance was measured manually in ROS. The depth values for Kinect for Xbox 360 range from 0.8m to 4m. Other distances are either unknown, too close or too far away. Kinect for Windows can operate in 'near mode', i.e. it can measure distances from 0.4m to 3m.

For the power supply for Kinect, a Panasonic 12V battery is used. The battery can be charge with 14v and 2A power supply. The battery was also mount on robot platform but easy to dismount for charging.

[1]Available at: https://github.com/uos/volksbot_driver (branch: master)

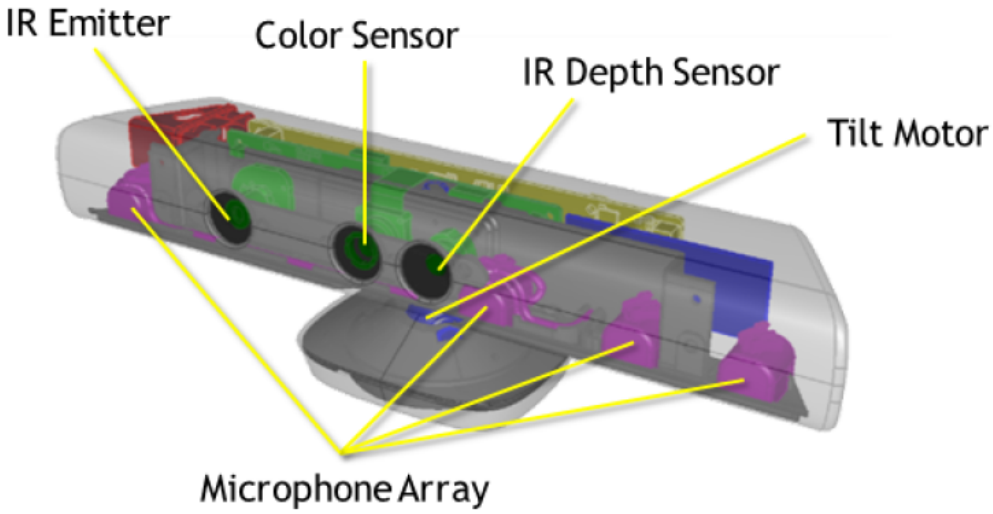


FIGURE 3.2. Kinect sensor components. (Image credits: Microsoft[kinc]).

Kinect for	Xbox 360 Specification
Viewing Angle	43° by 57° horizontal field of view
Image Resolution	640x480.
FPS	30 Hz(given 640x480 depth and color video).
Minimum Depth (measured)	0.5m.
Maximum Depth	8m.
Normal(Reliable) Depth Range	$0.8m < x < 4m$.

TABLE 3.2. Kinect for Xbox 360 Specifications

3.1.4 Laser Scanner Sick LMS100

For the perception of the environment, the laser scanner SICK LMS100 will be use. Every measurement generate a set of range values, where every range value correspond to a single laser beam. Figures 3.3 visualize one possible measurement of a scan line. The range from the scanner to an object can be calculated via the following measurement principle.: time of light, phase shift or triangulation.

The maximal measurement range in depth of the Sick LMS100 is 50 meters and the angular measurement with is 270°. the measurement of the range by the laser scanner is erroneous. the overall error for the range measurement is given by the statical error and the systematic error. the statical error is 20 millimeters and the systematic error is ±40 millimeters. therefore, the overall error is 60 millimeters.

The laser scanner doesn't give the exact values but in the range, The tolerance is not so big. For instant, with the distance of 10 centimeter, the LMS outputs the range from 9 cm to

FIGURE 3.3. SICK LMS 100 and visualization of a measurement from a laser scanner(Image credits: https://www.sick.com).

Variants	Special features
Operating Voltage	10.8V - 30V
Power	20W.
Body color	Blue.
Weight	1.3kg.
Parameters	106x102x152mmm.
Temperature range	0°-50°.
Application	Indoor
Field of view	270°.
Resolution of angle step	0.25°/0.5°.
Scanning frequency	25Hz/50Hz.

TABLE 3.3. SICK LMS100 Specifications.

10.7 cm with the tolerance is 1 cm. The result is not stable, in one angle, the distance changes continuously.

3.2 Coordinate System

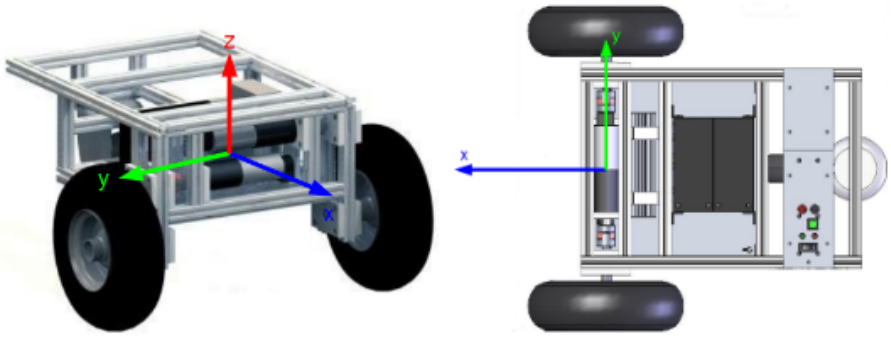
This section describes the different coordinate systems of the robot platform and it's sensor. The transformation from a sensor coordinate system to robot coordinate system will be describe here.

3.2.1 Coordinate System of the Robot

A coordinate system defines a plane or space by axes from a fixed point called the origin. Robot targets and positions are located by measurements along the axes of coordinate systems.

The base coordinate system has its zero point in the base of the robot, which makes movements predictable for fixed mounted robots. It is therefore useful for jogging a robot from one position to another.

The coordinate system of the Volksbot is centered in the middle of the axis between the two wheel. It is a left oriented coordinate system, where the x-asis is along the moving direction of the robot, the y-axis is perpendicular to the x-axis and positive to the right hand side. The z-axis complements the coordinate system. Figure 3.2 shows the describe coordinate system of the robot platform.



(a) Source for picture of robot: [32] (b) Source for picture of robot: [32]

FIGURE 3.4. Coordinate system of the robot.

Odometry: The Odometry coordinate system equal to the coordinate system of the robot. There is no transformation between the Odometry and the robot coordinate system required, because both coordinate systems are centered in the middle of the robot and left oriented systems.

3.2.2 Coordinate system of the Sensors

Laser scanner: The origin of the laser scanner's coordinate system is centered in the middle of the scanner. It is a 3D left orientated system. The x-axis is along the moving direction of the robot and the y-axis is positive at the side to the motor on the right hand side and complements the left orientated coordinate system.

The scanner's coordinate system has displacement to the robot's coordinate system. It is a translation in the x direction of 200 millimeters. To transform the Cartesian-Coordinate of a scan point to the robots coordinate system, the following transformation has to be calculated.

$$\begin{pmatrix} x_{Robot} \\ y_{Robot} \\ z_{Robot} \end{pmatrix} = \begin{pmatrix} x_{LMS} \\ y_{LMS} \\ z_{LMS} \end{pmatrix} = \begin{pmatrix} 0.2 \\ 0.0 \\ 0.0 \end{pmatrix}$$

No rotation has to be applied, because both coordinate system are left oriented system and the axis have same oriented in space.

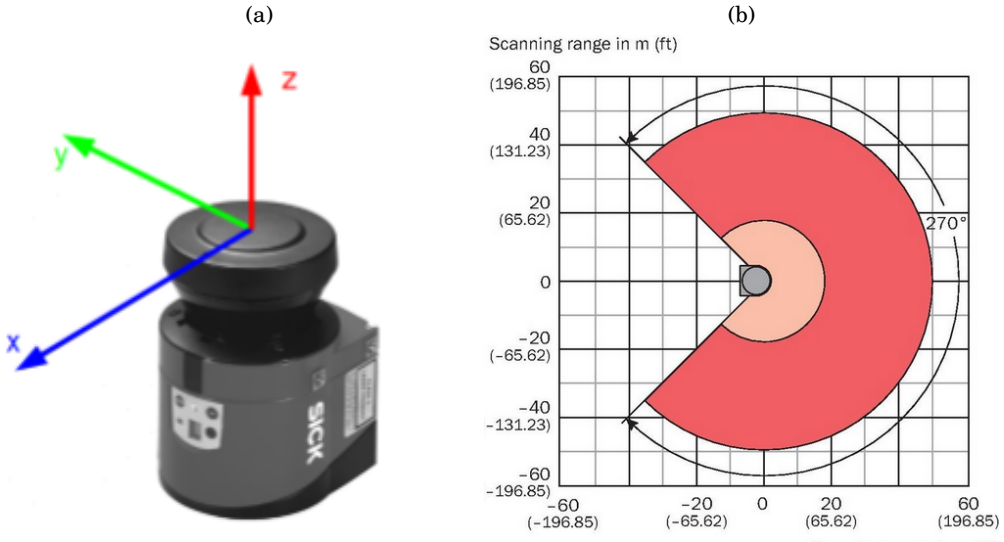


FIGURE 3.5. Coordinate system of the laser scanner. Source for the figure of the LMS100.

3.3 Software

3.3.1 Robot Operating System (ROS)

To implement and test the solutions for the functions and tasks, the Robot Operating System (ROS) will be use. ROS is an open-source operating system for robots. ROS is a collection of software framework for robot software development. It is a Linux based system which provides a communication layer to the robot hardware. ROS provides different libraries and tools to implement algorithms and applications to the robot. Currently different version of ROS exist. The newest released is "Kinetic", and this will be used for this thesis.

Different programming languages can be used to implement the new applications and algorithms. The languages, which can be used are C++, Python, Octava and Lisp. The different libraries are available for all for these four programming language[3]. The ROS implementation consist of nodes, messages, topics and services. A node is a process the executes algorithms and computations. An application can have many different nodes, working simultaneously. Single node are able to communicate and transfer informations via so called messages. A message

always consists of a header and multiple fields, which define the content of the message. ROS implements predefined messages for different contexts, for example:

Standard Message: This package of message types contains wrapper for all primitive types of ROS and the type *Empty*. The type *Empty* can be used to send a signal to another node, e.g. for notification. For instance, this will be used to initialize the AMCL for global localization.

Sensor messages: This package contains messages to communicate sensor related information. There are predefine messages for a laser scanner, the odometry, and Inertial Measurement Unit (IMU) or Global Positioning System (GPS) and many more.

Geometry messages: These message describle geometric objects like points in 2D or 3D space, poses with or without uncertainties and further types.

Navigation messages: Tis packet provides messages to communicate a map, the robot's pose related information and other information for navigation tasks.

Tools

Furthermore, ROS has many tools for visualization, for instance, we can using RVIZ to display the sensor data or view structure of a ROS application(nodes and topics) using rgt-graph. Nodes are visualized as ellipses and topics as rectangles with the names insides. The node is connected via the line with the topic. An examples for the visualization of the sensor data with tool RVIZ is shown in figure 3.6. Is show a small part of 3D-map make by Turtlebot in a small space.

All Topic can be logged and play-backed for debugging and simulation with real data, using the tool ros-bag. Also, all active topics can be listed and their messages printed on the console, using the command line tools of ros-topic.

ROS consists of many packages, which provide different drivers and algorithms for robot. There exits drivers for common sensors in robotics, like laser scanner, motor controller, IMU or GPS. These driver nodes broadcast the sensor related information in own topics. Also, common algorithms in robotics, like SLAM, motion planning for robot arms or automation navigation and utilities libraries, like calculating the transformation between different coordinate frames or particle filter, are available. However, there are many other packages for specific robot platform and other task, too.

3.3.2 Gazebo

For the simulation of the different algorithms of this thesis and maybe further exploration, gazebo will be use. Gazebo is a stand alone open source project for ROS simulation. Gazebo allows to create 3D models of arbitrary environments and robots and their sensors. Figure 3.8 show and

example of a 3D model of and chaotic environment and a 3D model of a robot.

To create a 3D model of a robot, the **United Robot Description Format**(URDF) file format can be used. URDF is a XML based description format. In general, the root link for the description is the base of the robot. All leafs of this root are sensors, wheels and other components. Using the XML marco language xarco, the creation of such URDF will be simplified. For example, if all wheels of a robot are equally the same in physical properties and looking, a macro for the wheels can be created and the position of the robot will specify, where the wheel will be set.

3D models of an arbitrary environment can be includes via the used of COLLADA files. COLLADA is also a XML based description language, like URDF, but it will be used for 3D models of building and other objects. This COLLADA files can be imported into the SDF files of Gazebo, which is also XML based[6]. SDF is the internal format of Gazebo to store the different models and worlds. An advantage of Gazebo is that the topic of the laser scan will be generated from the 3D model and available by listening on the corresponding topic. By adding a plugin for the odometry, the odometry topic can also simulated by Gazebo. This makes it easy to use Gazebo with ROS nodes, listening on these sensor topics. Not only the laser scanner and the odometry topics can be simulated using gazebo, but also further plug-ins, other sensor can be simulated and published.

3.3.3 Rviz

Rviz (ROS visualization) is a 3D visualizer for displaying sensor data and state information from ROS. Using rviz, we can visualize Baxter's current configuration on a virtual model of the robot. We can also display live representations of sensor values coming over ROS Topics including camera data, infrared distance measurements, sonar data, and more.

- RVIZ is perfect for figuring out what went wrong in a vision system. The list on the left has a check box for each item. You can show or hide any visual information instantly.
- The best part of RVIZ is the interactive marker. This is the part where you can be really creative. It makes selecting a certain area in 3D relative easy. You can therefore adjust your vision system manually while it is still running such as select a certain area as your work space and ignoring other region.

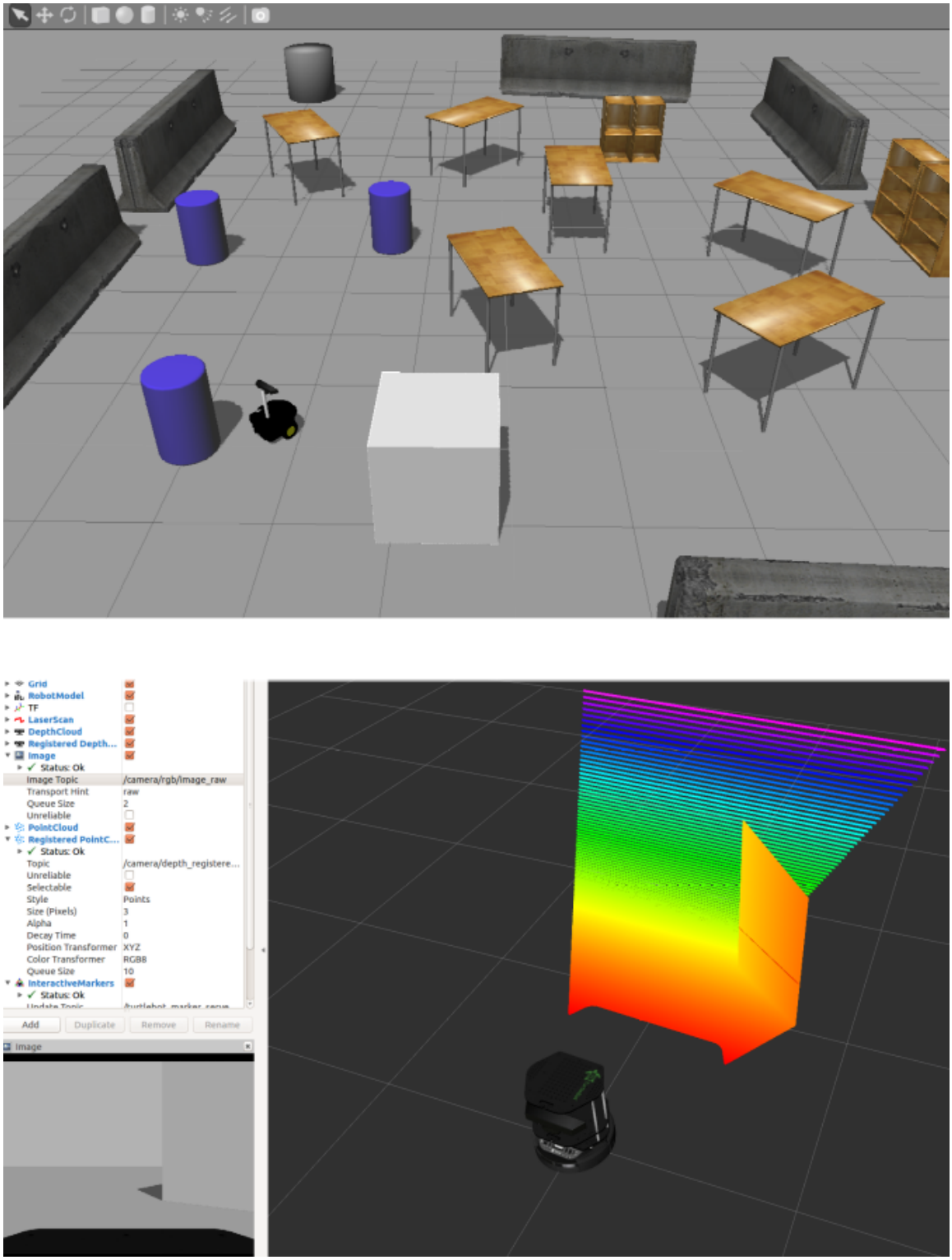


FIGURE 3.6. Simuate worlds in Gazebo and 3D point cloud in Rviz.

CHAPTER 4

SCENARIOS, IMPLEMENTATION AND EXPERIMENTS

4.1 Introduction

The robot operating system(ROS) runs on a laptop computer which is placed on the robot. This system is responsible for the core functionality of the robot, which includes sensor and actuator management, mapping, navigation and manual control. Fig 4.1 explains the program structure in more detail. The flow can be explained as follow:

- RTAB-Map receives two inputs: One input is odometry got from odom-fusion node and the second input is scan data from laser node. RTAB-Map output is occupancy grid map, this map will be both sent to nodes "Map" to display on screen and also to "Navigation Stack" node for calculating the path-planning.
- Received the occupancy grid map from RTAB-Map, the Navigation Stack then will combine it with the scan data from laser scan and odometry data from odom-fusion node to produce the output - the velocity command to the robot.

There are three device equipped in Volksbot.

- **Maxon EPOS motor control** Motor control provides an interface between the ROS computer and each wheel motor which was provided by Frauhofer. The two motors are used to run robot and provide odometry to predict the robot pose. They can connect to laptop via usb cable.
- **SICK laser scanner** The SICK laser scanner usually connects with robot via a router, with ip address 10.12.184.173. However, the wifi-connection quality is not really good because the router usually disconnects. Therefore, an Ethernet connection was established between

SICK laser scanner an ROS-computer. The computer must have static IPv4 is 10.12.184.174 and net-mask is 255.255.255.0.

- **Kinect Camera** A Kinect-Xbox 360 is mounted in front of the Volksbot. Height of camera compared to ground is 0.5 meter. The power supply for this Kinect is 12V Battery. The battery was also mounted on base_top of robot, an it needs to charge after 5 hours.

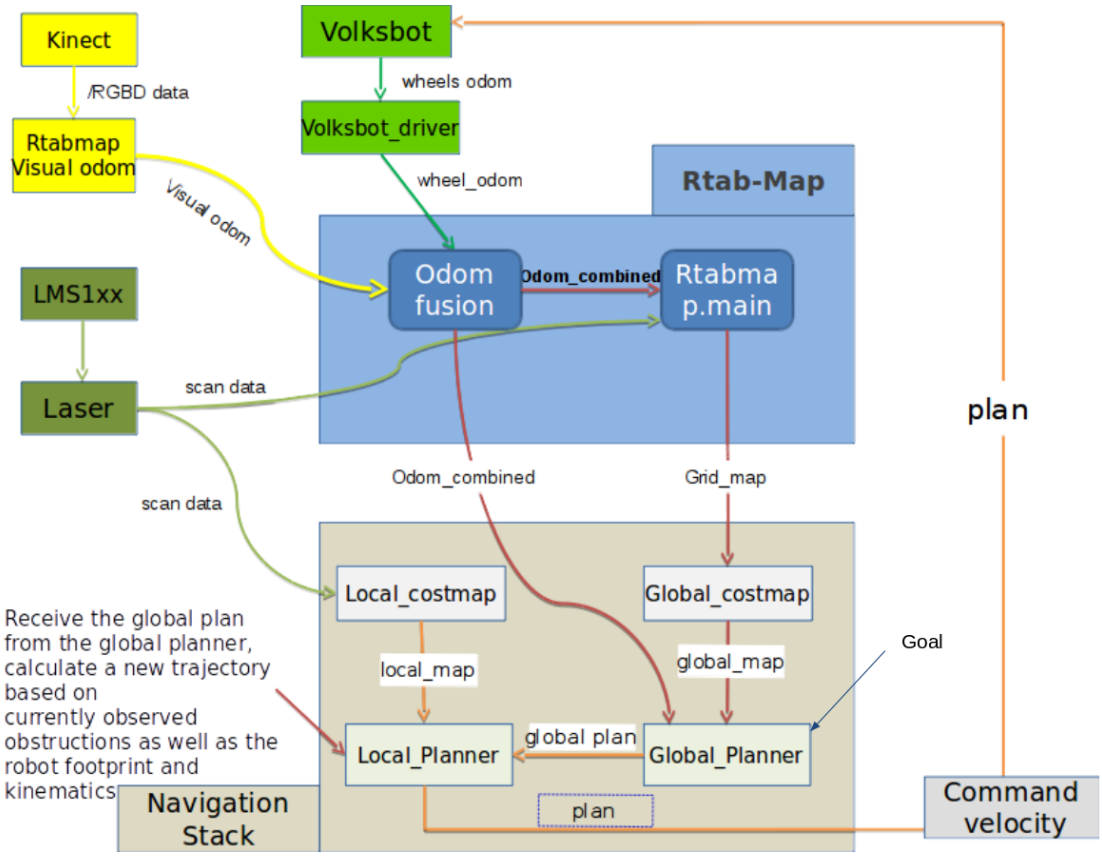


FIGURE 4.1. An overview of all devices connection and the flow data between them

4.2 Hardware Setup

4.2.1 On-Board Computer and upgrade robot platform

The robot was not equipped an on-board computer in the initial condition, program ran on PC and sent the command to robot via ethernet cable. However, to mapping mission, robot must navigate in large-scale environment. Therefore, a Lenovo-ThinkPath installed Linux 16.04 and ROS-kinetic is mounted on the top of robot as a main processor. This laptop will connect with a wi-fi router and can remote control from PC screen by team viewer or Remmina Remote Desktop

client. The platform was upgraded to improve the height of the Kinect-camera, because at old position the Kinect range could not reach the high obstacles, which leaded to an incomplete 3D-map. The change of robot platform shown in fig. 4.2

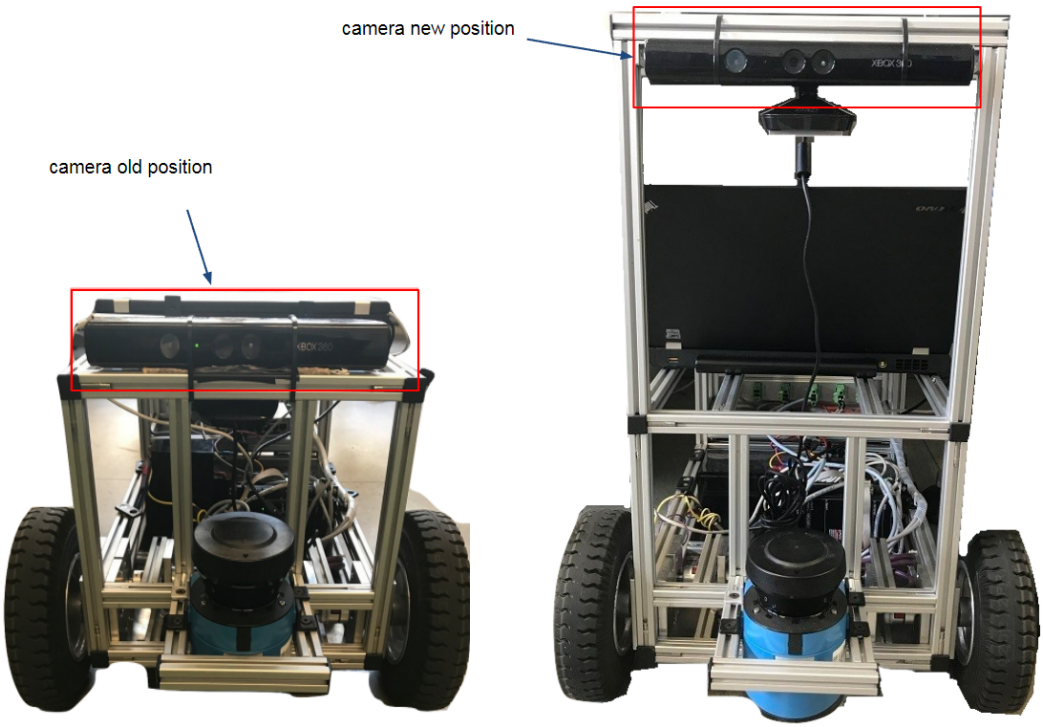


FIGURE 4.2. Upgrade part of robot.

4.2.2 Sensor Calibration and Setup

Support driver for the Kinect and the SICK LMS100 sensor was integrated into this system. Odometry from the two wheel encoders was included with the Volskbot_driver. Figure 4.3 illustrates how the sensors and the Maxon motor are connected to the ROS computer and how they are supplied with power.

Calibrating both the Kinect and the SICK LMS100 is a straight forward procedure with ROS. The SICK LMS100 needs to be calibrated to decrease the laser range from $(-30°, 240°)$ to $(0°, 180°)$, the calibration process executes on SOPAS engineering tools. Calibrating the Kinect is actually not strictly necessary because the lens distortion is very low. However, because there is a calibration tool available in ROS 2 that is easy to use, there is no reason for not calibrating. Calibration is highly recommended in the RTAB-Map configuration tutorial for ROS.

The calibration procedure is as follows:

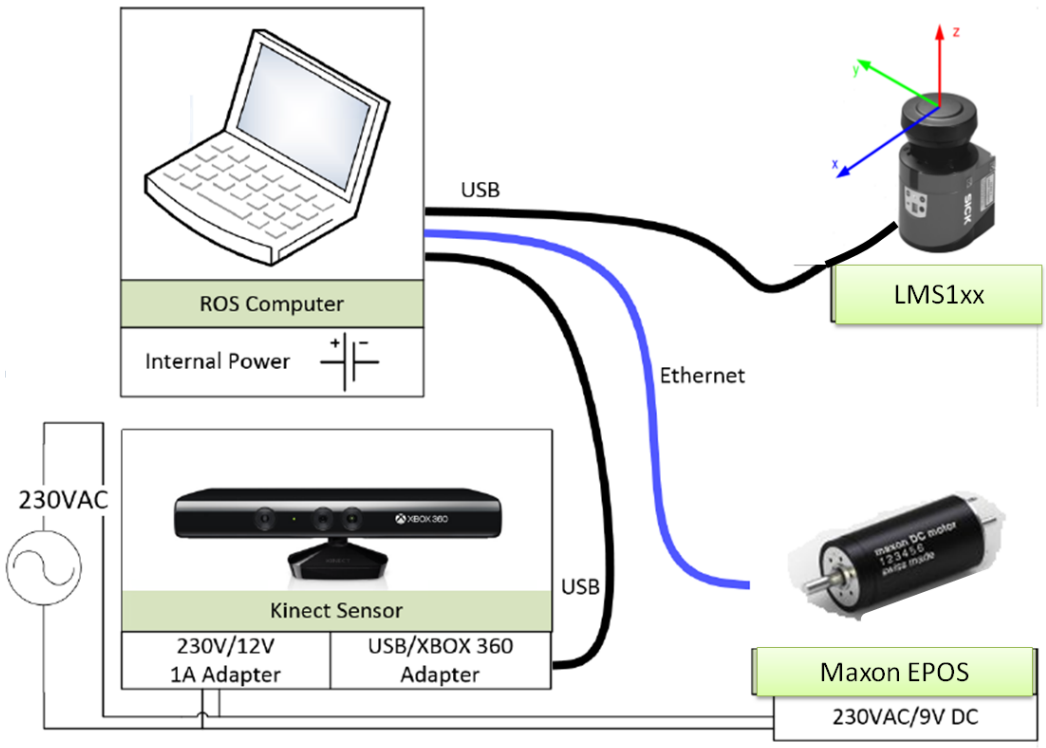


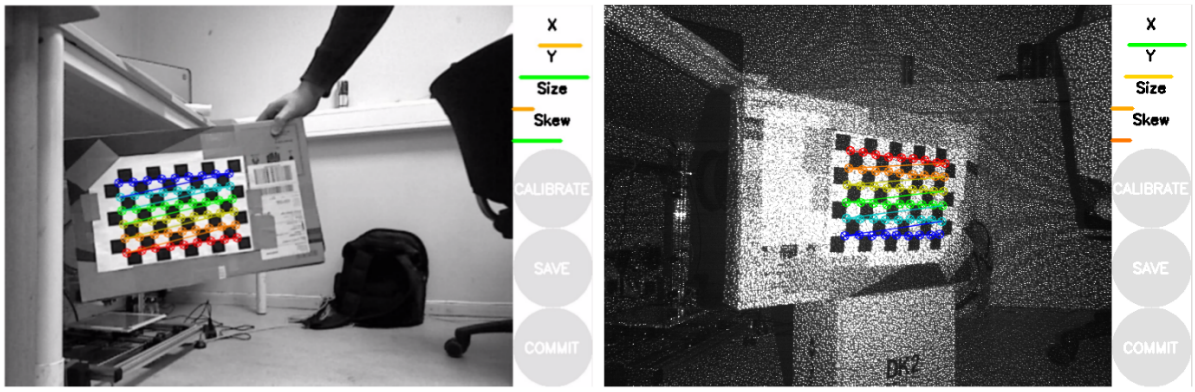
FIGURE 4.3. Sensor and power supply connections.

- Print out a chessboard pattern and tape it to a flat surface. It is beneficial to use a large paper size, for example A3, to make the pattern easier to detect over a larger range of distances.
- Calibrate the RGB camera by using the calibration tool. Use the RGB video stream.
- Calibrate the IR camera by using the calibration tool. Stream from the IR camera this time. For this procedure, it is recommended to cover the IR projector, because the IR speckle pattern makes it difficult to detect the chessboard (figure 4.4.b).

The calibration program needs to know the number of inner corners of the chessboard pattern, and the size of the squares. The chessboard used in this project has 6 by 9 inner corners and the size of the squares is 0.0275m. Larger pattern squares will make it easier to detect the chessboard pattern over a larger range of distances.

4.3 ROS Integration Overview

The robot software implementation is placed within a catkin workspace (see section 2.2.2) that contains all the project specific files that are necessary to build and run the robot system. The overarching file system is shown in figure 4.5. The packages used in this thesis include .



(a) RGB camera calibration. The camera can be calibrated when a sufficient number of samples have been obtained.

(b) IR camera calibration. The chessboard pattern will be difficult to detect, because the IR projector is not blocked.

FIGURE 4.4. Depth camera calibration.

- Volksbot_driver
- LMS1xx
- Frennect
- RTAP_map
- Navigation

Each package will be introduced in the following sections. The robot software implementation is placed within a catkin workspace that contains all the project specific files that are necessary to build and run the robot system.

The purpose of presenting the file systems is to clarify which parts of the system have been implemented. A short description of each package is given in table 4.1, and their place in the workspace hierarchy is shown in the next figure.

4.3.1 Simulation

Robot simulation was done in Gazebo; a simulation tool with good interfaces to ROS. The same ROS graph was used for both the simulated and real version of the robot, except from the sensors

Packages within catkin_workspace/src	
Volksbot_driver	Provide odometry to give predict position of the robot
LMS1xx	provide Laser scan data via topic /scan to give measurement position of Volksbot.
Frennect	Kinnect driver, Provide RGB-D image and display them in voxel to create a 3Dmap in Rviz.
RTAP_map	Intergrated Kalman Filer, Particle-Filer, receive information from Laser and Odometry to estimate the final position of the robot.
Navigation	This packages receive Map from rtabmap, sensor-stream data and odometry from whell endcoder and output the safe command velocity for the motor.
volksbot_gazebo	This packages launch the simulation Volksbot platform link with Kinect and Laser scanner via tf tree.

TABLE 4.1. List of custom using packages in the project.

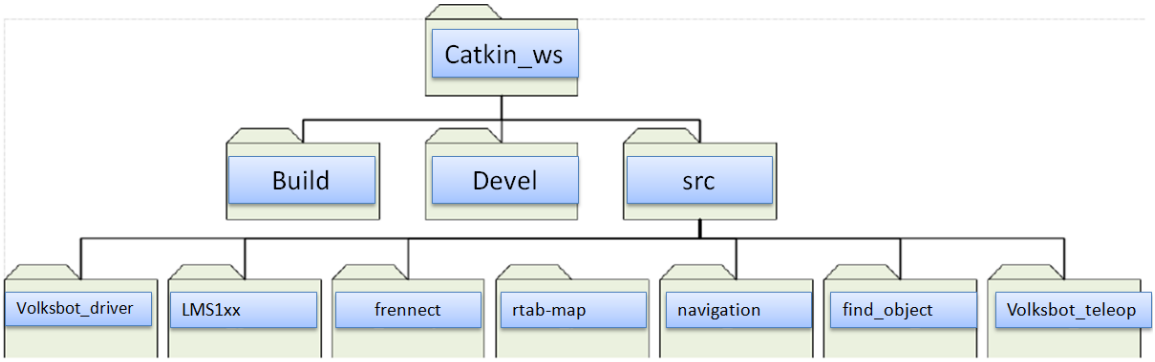


FIGURE 4.5. Overarching file system. The ROS packages are located within src.

and actuators, and some minor parameter changes. To properly simulate the robot, Gazebo needs a way to simulate the differential drive and the sensors in addition to the physical description of the robot. An expanded URDF description of the robot, volksobt.urdf, intended for Gazebo was placed within the volksbot/urdf package.

Robot Description Plugins The URDF model in volksbot_sim.urdf was expanded with plugins that simulate the motor control card, the SICK LMS and the Kinect. The plugins are provided by Gazebo and intended for use in ROS. The motor controller is simulated by the volksbot_driver plugin. Attributes for gazebo's sensor plugins are based on the technical specifications for the real sensors.

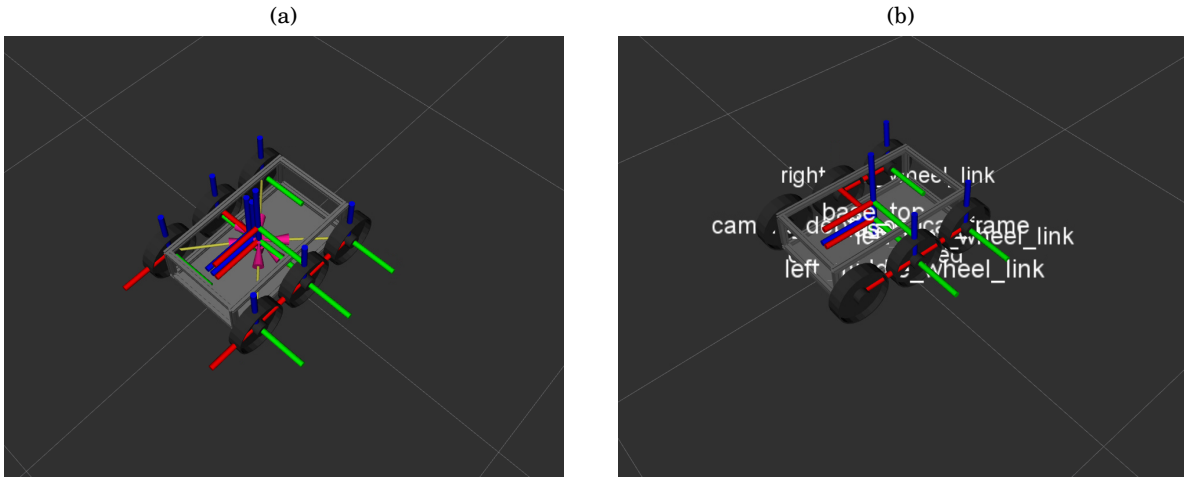


FIGURE 4.6. (a) Complete URDF model when view in gazebo (b) URDF model in rviz with all link frames and transformations.(c) Sensor input placed with correct transformations from base_link.

4.3.2 ROS Nodes for Motion Control

All nodes for motion control are located within the package mobile_base. This package is organized as shown in figure 4.9.

Velocity Command Flow

There are four sources that can generate velocity commands for the robot. The common for all of them is that they use the message format geometry_msgs/Twist. The names for each of the four velocity messages are as follows:

- /teleop Local keyboard input.
- /rtabmap/goal Input from 2D Nav button in Rviz.
- /move_base_simple/goal : Default Input from 2D Nav button.
- /cmd_vel Commands from the navigation stack in ROS. /cmd_vel is the default velocity command topic for the navigation stack.

To select the velocity command we can change directly in rviz. There are two topics having link with topic /cmd_vel are move_base and teleop. The message flow between source and sink is illustrated as a ROS graph in figure 4.9. Topic /planner/move_base will sent the command continuously in case autonomous navigation, while teleop_keyboard will receive form keyboard button and an transform to command for robot to tell it move forward, turn left, turn right,in teleoperate case.

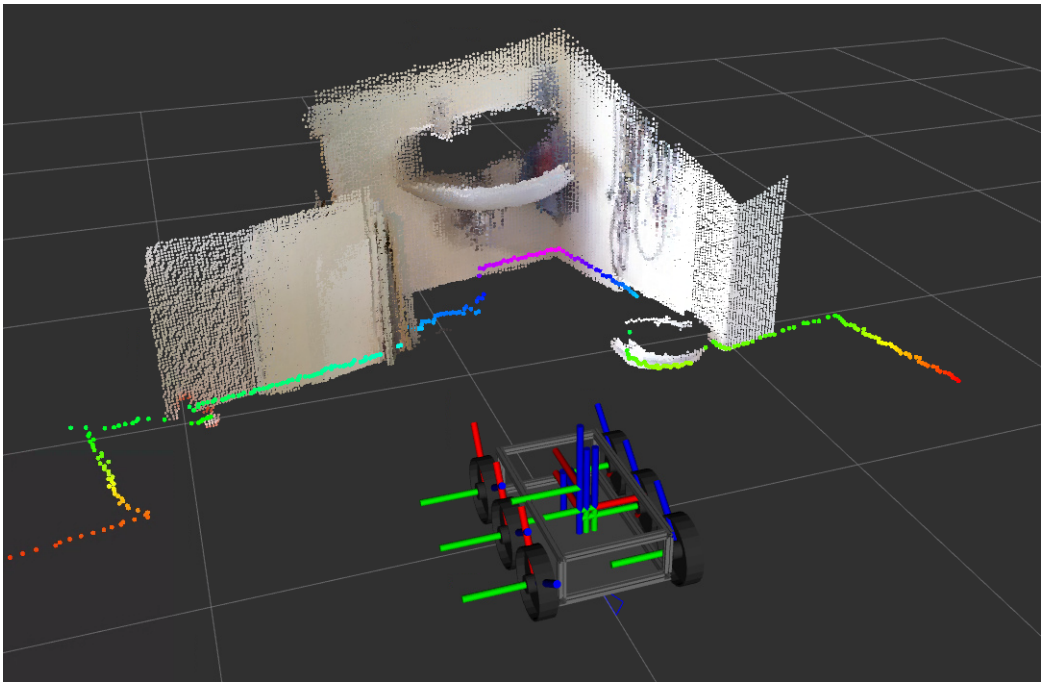


FIGURE 4.7. Sensor input placed with correct transformations from base_link.

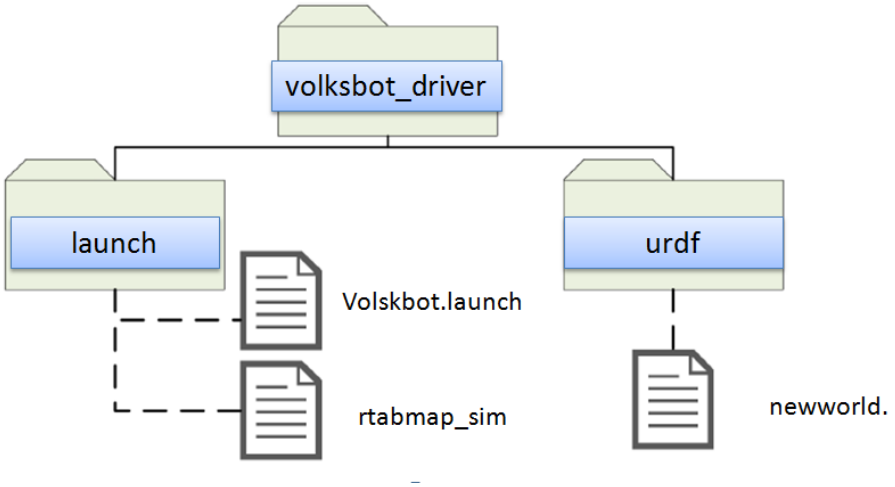


FIGURE 4.8. Folders and files specific for the simulator.

4.4 Mapping-setting up RTAB-map

This robot is using Real-Time Appearence Based Mapping (RTAB-Map) for SLAM. This section presents how RTAB-Map was configured for this robot[16]. In this project, RTAB-Map was initially installed in the form of a released binary for ROS Kinetic. Because of an Odometry bug which was fixed two weeks ago in the source code of rtabmap_ros, but not in the released binary,

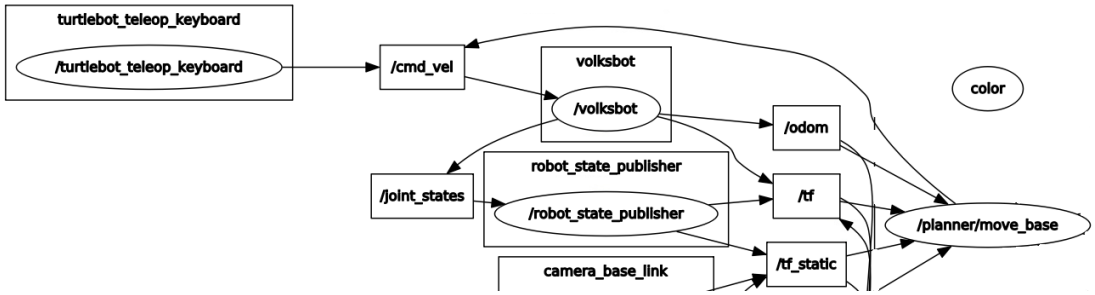


FIGURE 4.9. Nodes and topics for motion control.

this package had to be built and installed from source.

4.4.1 Configuration

As all ROS programs, RTAB-Map will run as a node that subscribes and publishes topics. The first task in configuring RTAB-Map is to connect the robots sensor data to the RTAB-Map node. The node, called rtabmap, can build 2D occupancy grids and/or 3D point cloud representations of the environment. In this project, RTAB-Map is configured to do both. The configuration is based on a guide provided by the developers. To perform SLAM, the mapping node subscribes to odometry, 2D laser scans and camera information. There are five possible sensor configurations with the Kinect:

- **1 - Kinect + LIDAR + Odometry** This is the configuration that was used in this project. Odometry data is generated by a the wheel encoder from Volksbot_driver.
- **2 - Kinect + Odometry + Fake 2D laser from Kinect** 2D laser scans are generated by passing depth images from the Kinect through the node depthimage_to_laserscan.
- **3 - Kinect + LIDAR** Sensor data can be sent directly to rtabmap. Odometry data is generated From LIDAR sensor.
- **4 - Kinect + Odometry** This configuration is suitable for uneven surfaces, i.e. when the vehicle is not constrained to a plane. Supports roll, pitch and yaw rotations.
- **5 - Kinect** In this mode, odometry will be generated by the rgbd_odometry node bundled with the rtabmap_ros package. This node publishes odometry messages based on feature correspondences in consecutive RGBD images received from the camera.

rtabmap subscribes to two image topics. One topic for depth images, /openni/depth/image_raw, and another for the rgb image, /openni/rgb/image_raw.

4.4.2 Adding 3D Obstruction Detection

The package rtabmap_ros contains a 3D obstacles detector in addition to the mapping node rtabmap. To enable 3D obstruction, point cloud data from the Kinect is passed through the nodelet obstacles_detection. The filtered point cloud is sent to move_base where it will be used in the local costmap. The code below is Kinect point cloud in file costmap_param.yaml.

```
point\_cloud\_sensorA: {
   sensor_frame: base_footprint,
   data_type: PointCloud2,
   topic: obstacles_cloud,
   expected_update_rate: 0.5,
   marking: true,
   clearing: true,
   min_obstacles_height: 0.04
   }
```

4.4.3 Navigation

The Navigation Stack is fairly simple on a conceptual level. It takes in the information from odometry and sensor streams and outputs velocity commands to send to a mobile base. Using the Navigation Stack on an arbitrary robot, however, is a bit more complicated. As a pre-requisite for navigation stack, the robot must be running ROS, have a tf transform tree in place, and publish sensor data using the correct ROS Message types. Also, the Navigation Stack needs to be configured for the shape and dynamics of a robot to perform at a high level. To set up navigation_stack for robot, the robot needs to adapt following requirement: ROS installed Transform Configuration Sensor Information Odometry Information Base Controller Mapping (this provide by RTAB-Map)

Autonomous navigation in this project is limited to use the navigation stack in ROS, which was introduced above. The stack will be used in its most basic form. The configuration was initially like in Appendix A . The implementation consists of one launch file for the node move_base, and a set of configuration files for each component within move_base[4]. Figure 4.10 shows how these files were structured. As with RTAB-Map, the configuration process consist of connecting move_base to the rest of the network, by deciding which topics it shall subscribe to.

4.4.3.1 Local Planner Parameters

The configuration file for the local planner was configured with some initial settings before testing in the simulator and on the live system. The final parameters can be found in the file local_planner_params.yaml.

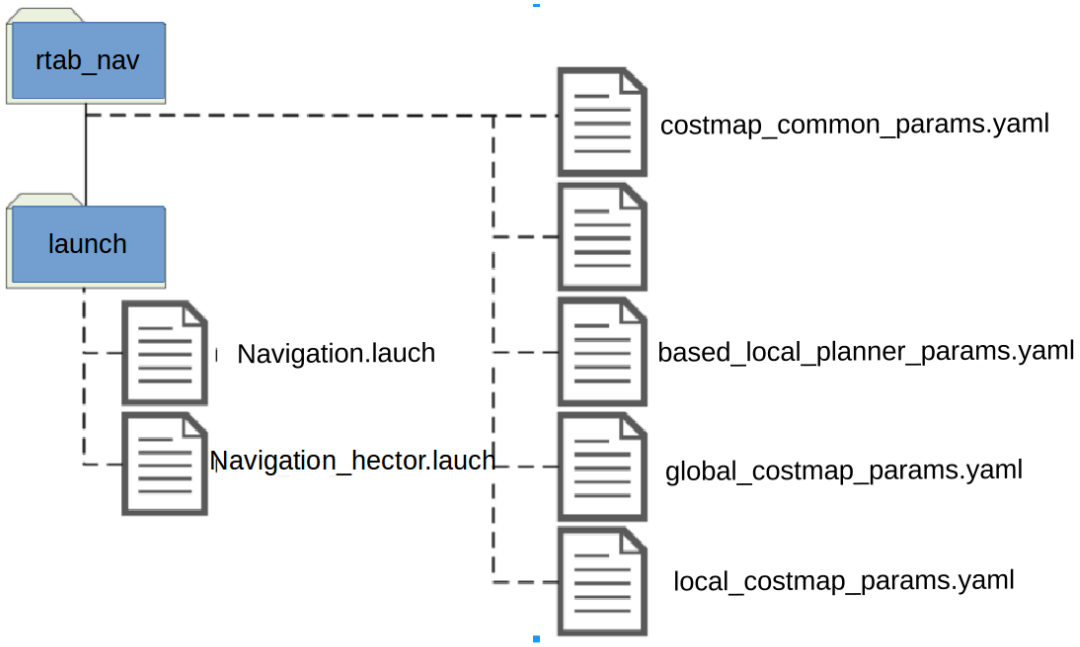


FIGURE 4.10. Files for configuring and launching the navigation stack.

4.4.3.2 Common Costmap Parameters

Because the same costmap package is used for both the local and global costmaps, there are some configuration parameters that are common for these costmaps. The tunable parameters include the robot footprint and the costmap inflation radius.

4.4.3.3 Obstruction Detection

Configuration of the 3D obstruction detector, consists of linking a point cloud topic to the node obstacles_detection, and pass the resulting filtered point-cloud to move_base[10]. The local costmap configuration file local_costmap_params.yaml must be configured to receive this point cloud by adding a point_cloud_sensor field to the file. The result of a successfully configured 3D obstacles detector is shown in figure 4.11.

In the first figure, a point cloud representation of the obstruction is presented. In this figure, the local costmap is based on the detected point cloud, so robot can detect the container as an obstacle. In second figure, the obstruction in the Gazebo simulator is shown. However, the LIDAR only detects the wheels below the container, so it cannot detect the container as a big obstacle but only two small container's wheels as obstacles.

In fig 4.12, 3-D Obstacles detection with the live robot is shown. The nodelet obstacles_detection filters out the floor and publishes a point cloud which can be sent to the move_base node. The yellow arrow points to the local cost map, which is based on real-time sensor data and used by

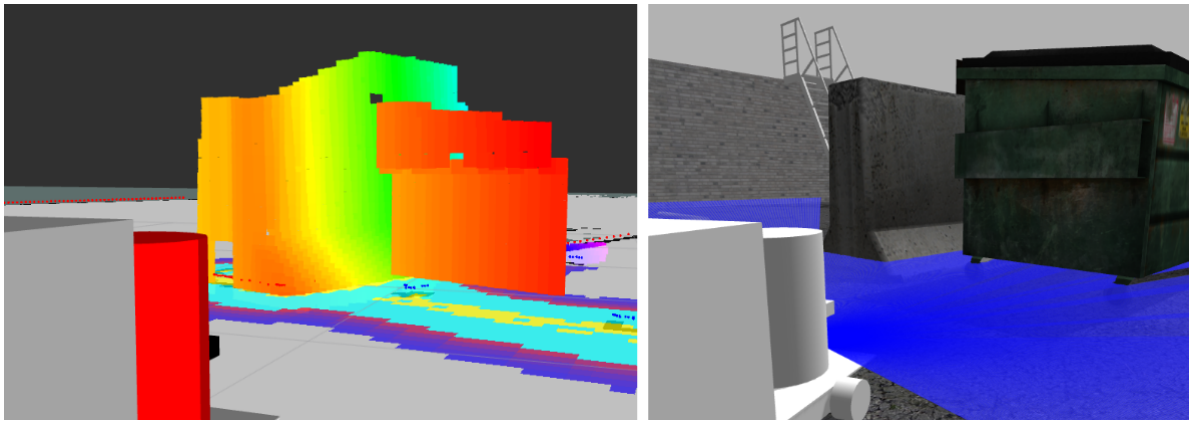


FIGURE 4.11. Detecting obstructions in 3d http://wiki.ros.org/robot_localization.

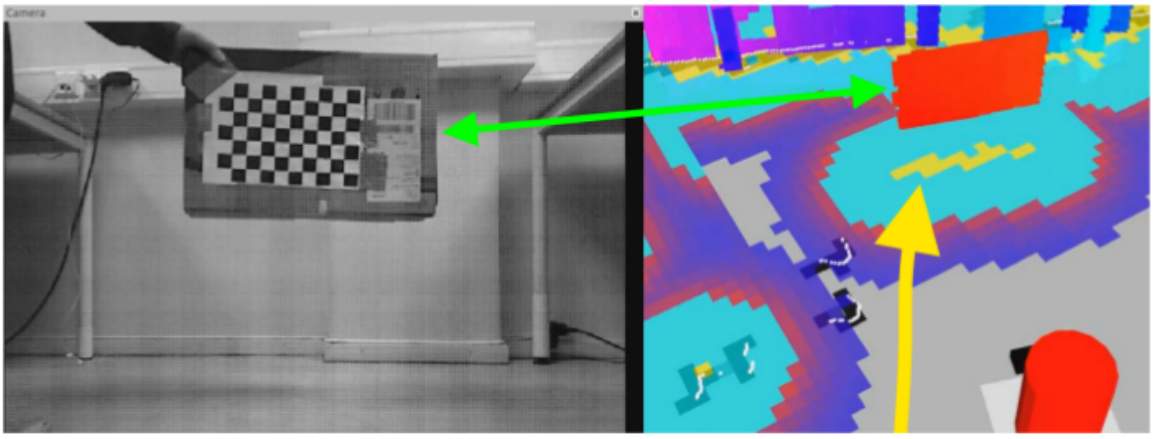


FIGURE 4.12. 3D Obstacles detection with the live robot. The nodelet obstacles_detection filters out the floor and publishes a point cloud which can be sent to the move_base node. The yellow arrow points to the local cost map, which is based on real-time sensor data and used by the local planner.

the local planner.

4.4.4 Package Installation

This subsection is a tutorial to install all the packages using in this thesis. The overview of how packages distribution is shown in figure 4.13.

Before installing those packages, user must install ROS and catkin workspace first. ROS Kinetic ONLY supports Wily (Ubuntu 15.10), Xenial (Ubuntu 16.04) and Jessie (Debian 8) for debian packages. For step by step installing ROS for Ubuntu, all information are described very

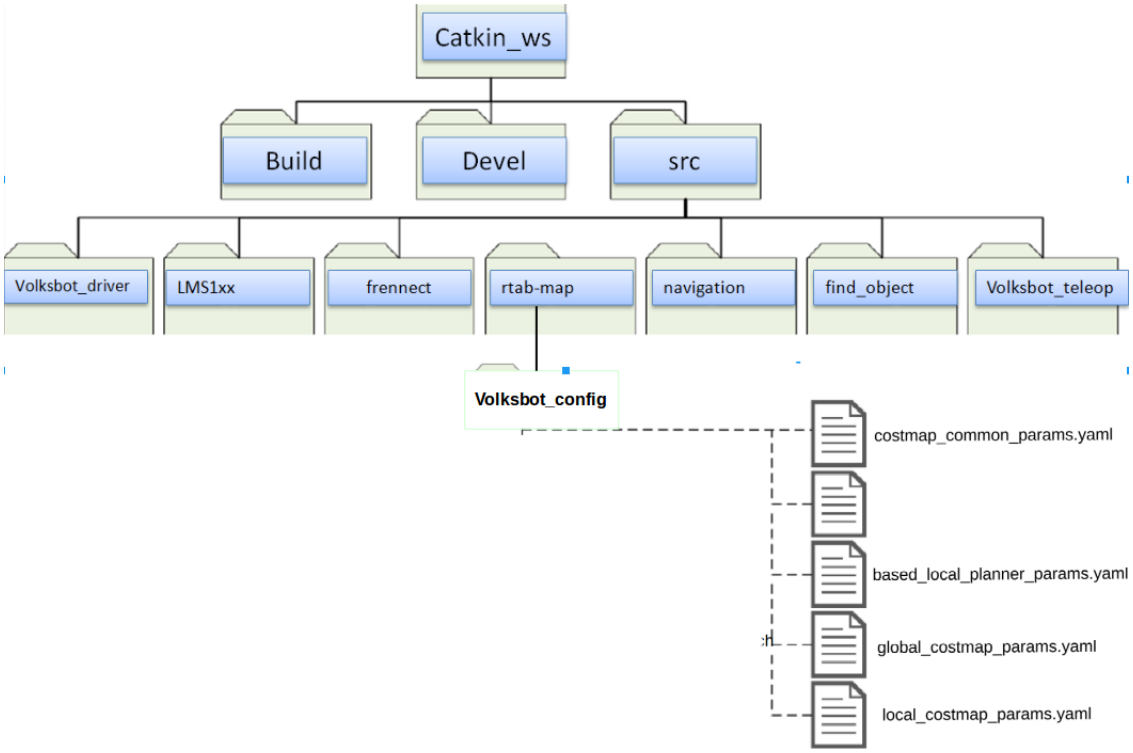


FIGURE 4.13. Detecting obstructions in 3d http://wiki.ros.org/robot_localization.

clearly in this wiki.ros.org. [1] There are two ways to install ros-package, first is install from binary, with this way the program install very simply with only one command in Terminal

```
$ sudo apt-get install <package_name>
```

And this package Path to /opt/ros/kinetic/. Even though it is simple, you cannot change the source code as longs as repairing to the program when it has error. Therefore, author recommended to install from source, although this needs longer time and a lot of command, it will decrease the unexpected error later. To install from source, firstly we muss install catkin_workspace. This is the compile environment where you can put all ros-package here and complile them with command catkin_make.

Let's create and build a catkin workspace:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

The catkin_make command is a convenient tool for working with catkin workspaces. Running it the first time in your workspace, it will create a CMakeLists.txt link in your 'src' folder. Additionally, if you look in your current directory, you should now have a 'build' and 'devel' folder. Inside the 'devel' folder, there are now several setup.*sh files. Sourcing any of these files

[1] Available at: http://wiki.ros.org/kinetic/Installation/Ubuntu

will overlay this workspace on top of your environment. To understand more about this, see the general catkin documentation: catkin. Before continuing, source your new setup in.*sh file. To make sure your workspace properly overlayed by the setup script, ROS_PACKAGE_PATH environment variable must be ensured to include the directory you're in.

```
$ source devel/setup.bash
$ echo $ROS_PACKAGE_PATH
/home/youruser/catkin_ws/src:/opt/ros/kinetic/share
```

4.4.4.1 ROS-Driver for Odometry

To handle the differential odometry of the Volksbot RT3, the volksbot-uos-driver[2] is used. This driver defines a right orientated coordinate system and depends on libepos2. In section 3.3.2, the coordinate system of the odometry is defined as left orientated. Therefore, the driver is adapted to the left orientated coordinate system.

Volksbot_driver Installation.

```
$ sudo add-apt-repository ppa:kbs/kbs
$ sudo apt-get update
$ sudo apt-get install libepos2-dev
$ git clone https://github.com/uos/volksbot_driver.git
$ cd catkin_ws/src
$ catkin_make
```

Launch the Volksbot

To launch Volksbot_Odometry, just run the launch file volksbot_bringup.launch

```
$ cd Volksbot_thanh
$ roslaunch Volksbot_bringup.launch
```

This launch file is a self-create file, include command to start Epos motor_driver. Kinect-camera and depth-image of Kinect, SICK laser scanner lms1xx and the tf to keep contact between these frames. The relationship of these coordinate frame is maintained in a tree structure buffered in time.

4.4.4.2 Implement Laser Scanner

The SICK laser scanner can output range measurements from the angle of 45° to 275°. It has vertical solution of 0.25° , 0.5° or 1.0°, meaning that the width of the area the laser beam measure is 0.25°, 0.5° or 1.0° wide. A typical laser scanner output will look like:... 2.98, 2.99, 3.00, 3.01, 3.00, 3.49. The output from the laser scanner tells the ranges from right to left in terms of meters. If the laser scanner for some reason cannot tell the exact length for a specific angle it will return hight value. We are using threshold to tell if the value is an error.

SICK laser driver

[2]Available at: https://github.com/uos/volksbot driver (branch: master)

The lms1xx package contain a basic ROS driver for the SICK LMS1xx. This package should work with SICK LMS 1XX laser range finders.

We can install this package using apt:

- **Node** LMS100: LMS100 is a ROS node for libLMS1xx.
- **Published topics** scan (sensor_msgs/LaserScan): Scan data form the laser.
- **Parameters** host (string, default : 10.12.184.176) frame_id: The laser data frame. This frame should be at the optical center of the laser, with the x-axis

```
$ sudo apt-get install ros-kinetic-lms1xx
$ roslaunch LMS1xx lms1xx.launch
$ rospy find_laser.py
```

We can change IP address in file lms1xx_node.cpp.IP adress of laser scanner can be found in SOSPAS tools of SICK company, or we scan IP address of Laser by run python files find_laser.py.

The main code of lms1xx node is in file LMS1xx_node.cpp, in this file the range of the laser beam can be changed by limiting the threshold in the code. This is very important because if the range of the laser is too low, it can detect our robot as an obstacle, which will make robot stuck in it's own footprint, the problem that we never want to happen. Besides, there is another way to make the laser_scanner not detect the robot as an obstacle. Instead of limiting the laser range, we can change the scan angle from 270° to 180°. Nevertheless, this method has two disadvantage, first the scan angle was decreased, that means robot needs more time to create a map. Beside that, ROS does not support SICK laser scan to change scan angle, so SOPAS engineering tool is used to change the angle of laser scanner. A user-name and password needed for login to SICK lms1xx via SOPAS, at present the user name is "Aministrator" and passwort is "client".

4.4.5 Configure the main program

rtabmap.lanch file is the main files to create a map and while at the same time localize the Volksbot in this map. RBGD_mapping[14] is the one part in project rtabmap, which receive Data form camera and laser scan to create a Grid_map and define the position of robot in this map. The initial point of robot is (0.0, 0.0, 0.0). The result can be visualize and modify in Rviz.

To receive information from the Laser scanner we need to remap topic "odom" to out the Volksbot_odometry in rtabmap.launch. And topic scan also need to remap to lms1xx laser_scan topic. The following parameter must be changed in rtabmap.launch:

```
<group ns="rtabmap">
<node name="rtabmap" pkg="rtabmap_ros" type="rtabmap" output="screen"
 args="$(args␣rtabmap_args)">
        <param name="frame_id"       type="string" value="base_link"/>
        <param name="subscribe_laserScan" type="bool"   value="true"/>
        <param name="use_action_for_goal" type="bool"   value="true"/>
```

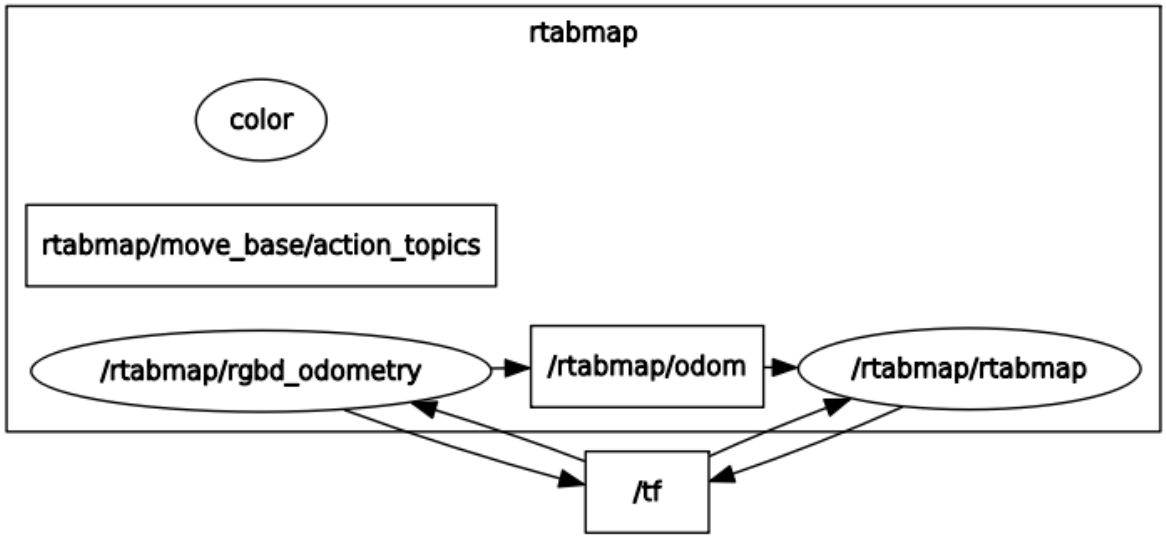


FIGURE 4.14. Nodes graph of the rtabmap topic.

```
        <remap from="scan"              to="/scan"/>
        <remap from="odom"              to="/odom"/>
```

This launch files run after volksbot_bringup.launch run, run the following command to launch rtabmap.launch.

```
$ roslaunch rtabmap_ros rgbd_mapping.launch frame_id:=base_footprint
$ rtabmap_args:="--delete_db_on_start" rtabmapviz:=false subscribe_scan:=true
```

The command above will run file rgbd_mapping.launch in folder rtabmap_ros, rtabmap_args: ="delete_db_on_start" mean robot will delete all previous map before create the new map for long_term_memory capacity save and do not conflict between the present and previous map. Fig.4.14 is the rqt_graph, which shows the communication between the nodes in topics rtabmap, tree node rtabmap/rgbd_odometry, rtabmap/odom and rtabmap/rtabmap communicate with each other via transform tf.

The parameter is needed to understand to use in rtabmap_node. To understand and know what parameters can be change, and what they mean, the following command is needed:

```
$ rosrun rtabmap_ros rtabmap --params
$ rosrun rtabmap_ros rgbd_odometry --params
```

There are three ways to change the parameters.

- 1) Use a config.ini file.

  The default config.ini should be automatically generated once you launch rtabmap through terminal by " rtabmap" command. Its location is /Documents/RTAB-Map/config.ini. You can

copy this config.ini file into your own package and change your launch file cfg parameter into the corresponding location.

For example, you can set Odom/Strategy=1 in *config.ini*, then in a package called arti_vision, you can put the *config.ini* in the config folder and rename it into *rtabmap.ini*. Calling *rtabmap.launch* with the cfg argument:

```
<arg name="rtabmap_args" default="$(find␣arti_vision)/config/
rtabmap.ini" />
```

- 2) Use <param> tag

  This is normal ROS way of reading parameters, in the launch file below the node, put the following command to change the parameter of this node.

```
<param name="Odom/Strategy" value="0"/>
```

- 3) Add parameters in arguments of the node. If you just want to temporarily play with some parameters, you can quickly put it under the rtabmap_args argument:

```
<arg name="rtabmap_args" default="--Odom/Strategy␣0"/
```

4.4.5.1 Visual Odometry

Tuning visual odometry parameter is very important, it directly affects how well RTAB-Map generally performs. If the computer performance is limited, the following parameters can be changed to increase the odometry frequency, and thus, the robot won't get lost.

```
#Change the Visual Odometry Stategy into Frame to Frame
<!-- 0=Frame-to-Map (F2M) 1=Frame-to-Frame (F2F) -->
<param name="Odom/Strategy" value="1"/>
#Change the Matching Correspondences into Optical Flow
<!-- Correspondences: 0=Features Matching, 1=Optical Flow -->
<param name="Vis/CorType" value="1"/>
#Reduce the Maximum Features
<!-- maximum features map size, default 2000 -->
<param name="OdomF2M/MaxSize" type="string" value="1000"/>
<!-- maximum features extracted by image, default 1000 -->
<param name="Vis/MaxFeatures" type="string" value="600"/>
```

Increase in the Frame Rate of Camera and Reduce the Resolution is also a very effective way. An increase in camera's frame rate directly affects how fast the VO can track the robot. Reduction in resolution can significantly increase the speed of processing but may reduce the accuracy.

4.4.5.2 Odometry Auto-Reset

When calling reset_odom service, rtabmap will start a new map, hiding the old one until a loop closure is found with the previous map. If you don't want rtabmap to start a new map when odometry

is reset and wait until a first loop closure is found, you can set Rtabmap/StartNewMapOnLoopClosure to true. If parameter Odom/ResetCountdown is set to 1 (default 0=disabled), odometry will automatically reset one frame after being lost, i.e., it has the same effect than calling reset_odom service. It is very good to apply for the Volksbot, because when robot navigates, the speed is high, the Kinect-camera therefore will be very easy to loose odometry.

```
<node pkg="rtabmap_ros" type="rgbd_odometry" name="rgbd_odometry">
        <param name="Odom/ResetCountdown"              value="1" />
<node/>
```

CHAPTER 5

RESULT AND EVALUATION

The upcoming chapter shows the results of the experiments described from the previous chapters. The implementation method will be evaluated on different criteria and some additional observation will be given. The same software system is used for both the simulated and live robot.

5.1 Test Plan

The tests listed in tables 5.1 will be carried out. They will mainly focus on the navigation stack and RTAB-Map.

Evaluate	Description
Map_drawing by	Volksbot_teleoperate used to control robot to create a map
Control robot from remote Desktop	Steer the robot from the Remote Desktop while monitoring the robot through Team Viewer.
Accurate of the Program	Compare robot position in real life and in Rviz.
Multi Session Mapping	Verify that the robot can rediscover areas which have been mapped in previous mapping session.
Loop Closure Detection	As a core functionality in RTAB-Map, it is critical to evaluate the loop closure mechanism.
Autonomous Navigation	Perform a set of tests on the navigation stack. The tests should evaluate path planning with static and moving obstacles.

TABLE 5.1. Core Functionality.

5.2 Brief Summary of All Results

Teleoperate with Volksbot_teleop The launch file works as expected. Users can control the Volksbot via keyboard, with "i","," is move forward and backward. "u","o","j","l" is turn left, turn right for the front wheels and behind wheels respectively. Speed of robot can be adjusted with keys "q/z", with "q" to increase robot velocity 10%, and velocity can increase/decrease maximum 10 times. Any other buttons will stop the robot. In figure 5.1, we show the screen instructions:

```
Reading from the keyboard  and Publishing to Twist!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
anything else : stop

CTRL-C to quit
```

FIGURE 5.1. Screen instructions Volksbot_teleop.

Remote Desktop Control Through the "Remmina Remote Desktop Client on Linux", an operator can connect to the robot via a VNC, stream video from an URL and control the robot by using the mouse. To connect two computers, first we must set parameters for "Desktop Sharing Preferences". The setting tutorial is in fig.5.1.b. The password is "123".

Then, on the remote screen, open Remmina Remote Desktop Client and type Ip of Robot's PC, here is "10.12.10.159", then click Connect! The program will ask for the password we set before, just type "123".

Autonomous Navigation Autonomous navigation was evaluated based on the ability to reach a feasible goal state, the ability to avoid static and dynamic obstacles. The global planner works quite well in both the simulator and in the real world, as expected. Volksbot successfully navigates static and dynamic obstacles course. However, robot usually gets error when it moves to the narrow place, like a door, because in some local cost map, the robot absolutely stuck in the inflated area, although in reality robot still has space to move.

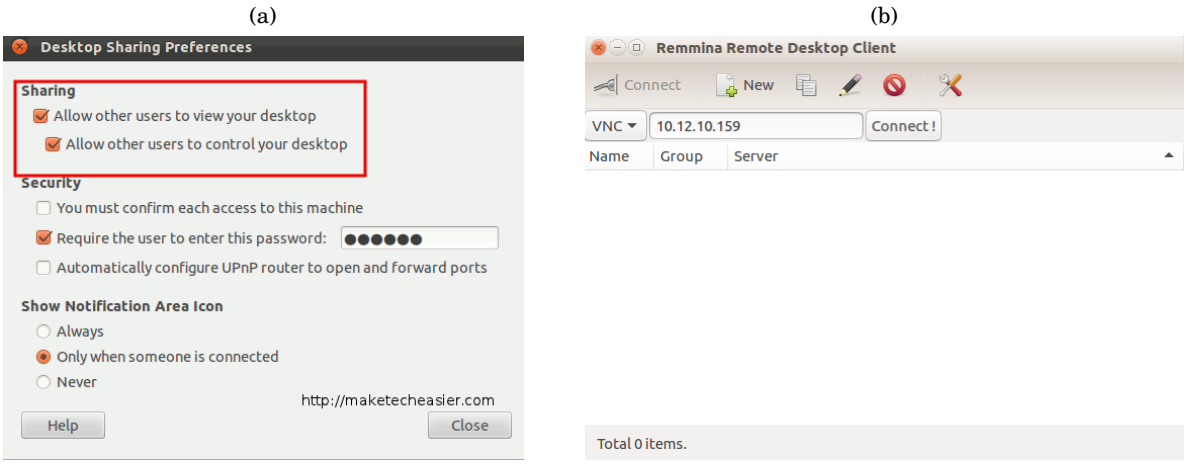


FIGURE 5.2. Setup connection between 2 robot processor and a remote Desktop.

5.3 Simulation Results

5.3.1 Mapping

Navigation stack on the robot in Gazebo allows for the controlled testing in a simulated environment. The simulated world proves to be a challenge for navigation at least with the parameters that were used during testing. However, the results got from the test reveal an optimistic view, the robot can navigate smoothly and create a 3D map eventually. Figure 5.3 shows an example of a result of the 3D map in Rviz and the simulate robot and obstacles in Gazebo.

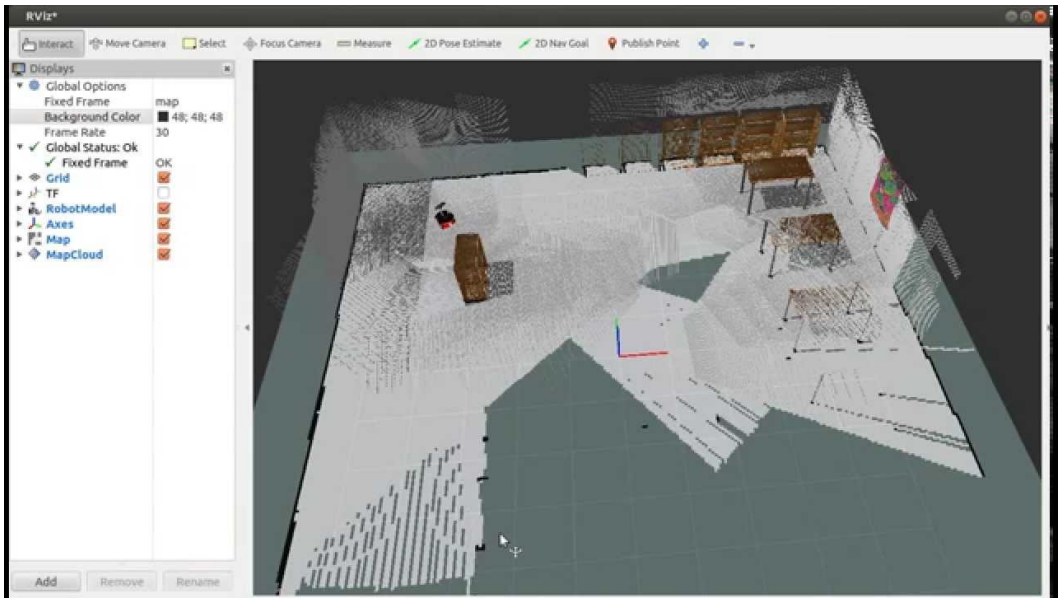


FIGURE 5.3. 3D map of draw in simulation world gazebo using tool Rviz

5.3.2 Autonomous Navigation

Testing the navigation stack in Gazebo fulfilled two goals. The first goal was to learn how the system behaved with different parameters and to uncover potential problems with the system, before attempting to test the live robot. The second goal was to evaluate the ability to relocate the robot and avoid obstacles.
For the first trial, the robot footprint was extended well beyond the physical mobile base. The purpose was to prevent the obstacles from getting too close to the Kinect and laser scanner. As the minimum detectable range for the Kinect is 0.8 meter (measured minimum depth), the footprint was extended 0.8 meter beyond the front of the mobile base. A second parameter to be tuned is the inflation radius, the area beyond each obstacles where it is expensive or impossible to traverse. During trials with the big footprint, the robot showed good collision avoidance capability but the ability to follow the planned path is reduced. Setting the obstacles inflation radius is also a dilemma in choosing between large margins for the global path or the ability to navigate through narrow passages.
In later navigation sessions, the robot footprint was adjusted slightly larger than the physical robot base. During some of the testing sessions, the robot would get stuck by near obstacles.

5.4 Live Robot Results

5.4.1 Mapping

In this test, a Wi-fi connection was established between a PC in Autonomous lab and Think-pad on-board of Volksbot. The maximum range to control the robot is up to 25 meters. A 3D map drawn by Volksbot RT3 using RTAB-Map is shown in figure 5.4 which captures the structure of building 8, floor 2 of Frankfurt University of Applied Science. The robot is required to create a map from the room 206a, then it moves to room 206b and along with the corridor before it returns to the starting point. The map and position of robot were update continuously while it was running. The 3D point cloud is create by Kinect Camera and the blue lines represent the robot's trajectory. In experiments, the real robot velocity was in general slower than its simulated version because of the tougher environment.

Sometimes, in real tests, the loop closure detection cannot recognize the visited position if the robot is moving in the environment where it lacks features. Particularly, when the robot navigates in corridor, the walls and the floor are similar to each other everywhere. The new captured image, therefore, cannot be compared with previous images stored in long term memory, hence, the loop closure detection will add the current image to its memory, an overlapped map is produced as the result. In this case,the screen will notify error.

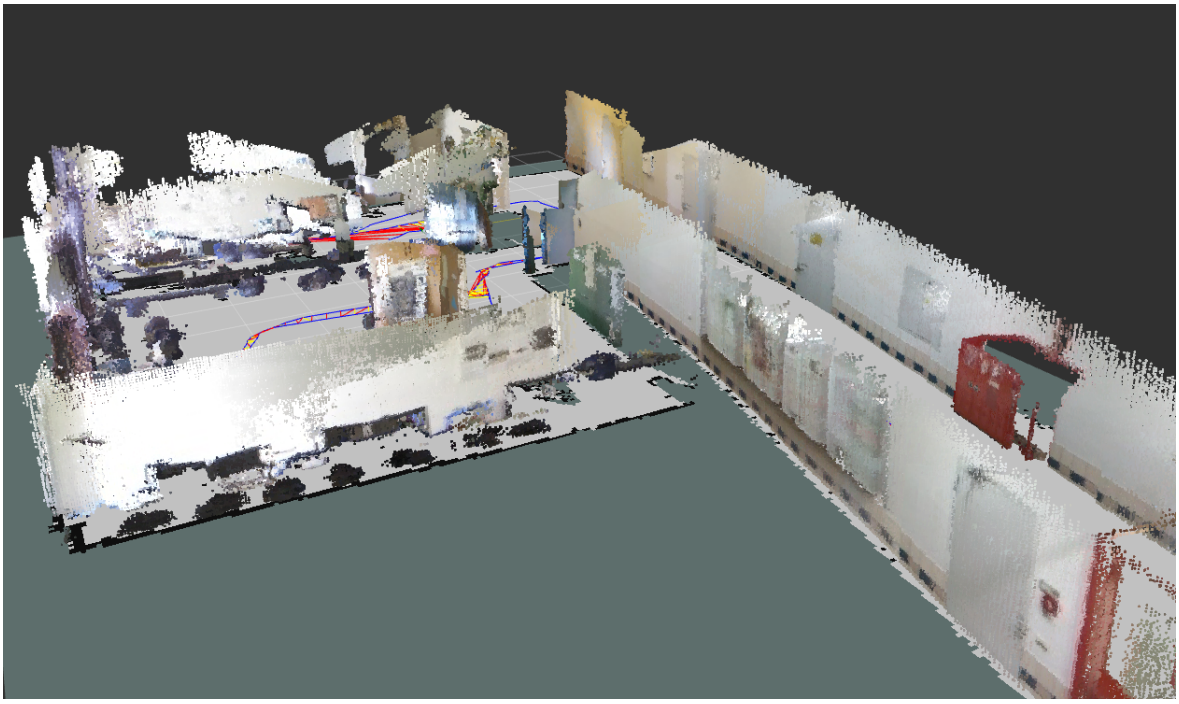


FIGURE 5.4. 3D map of building 8, floor 2, Frankfurt University of Applied Science include corridor and rooms 206.a,b, draw by Volksbot RT3 and RTAB-Map

5.4.2 Test accuracy of robot localization

One of the main concern in mapping is the accuracy. To meet this requirement, robot must define exactly its position. A small experiment has been set up to compare the position displayed in Rviz (ROS visualization) with the real environment. Note that the robot always sets the initial point(0,0) as the first position when program starts. The setup was shown in figure 5.5. In the 7 meters distance, robot moves from Autonomous Lab, crosses the door and runs along the corridor. The exact path is that firstly, robot moves forward 4 meters, then turns right, and move forward 3 meters. The first 4 meters are occurred inside the room, while the next path is planned to move outside the room. This special design serves the purpose to evaluate to what extent the localization program is stable when it navigates in strange environment.

Fig.5.5 shows the difference between real and visual robot pose when it uses the odometry from Kinect camera as an input. As indicated from the graph, in the case robot moves inside laboratory from the initial point (0,0) to (4,0),the difference between the robot position in Rviz and real world is very small, the tolerance is approximately 0.002m in both x and y direction. However, the errors are significantly increasing when robot moves outside, revealing that the inputs play a significant role for an accurate localization, especially in a new environment. Even though it has not been shown in this thesis, better results have been achieved if we use more

inputs for estimation, such as laser measurement and odometry from wheel encoder as mentioned in previous chapters.

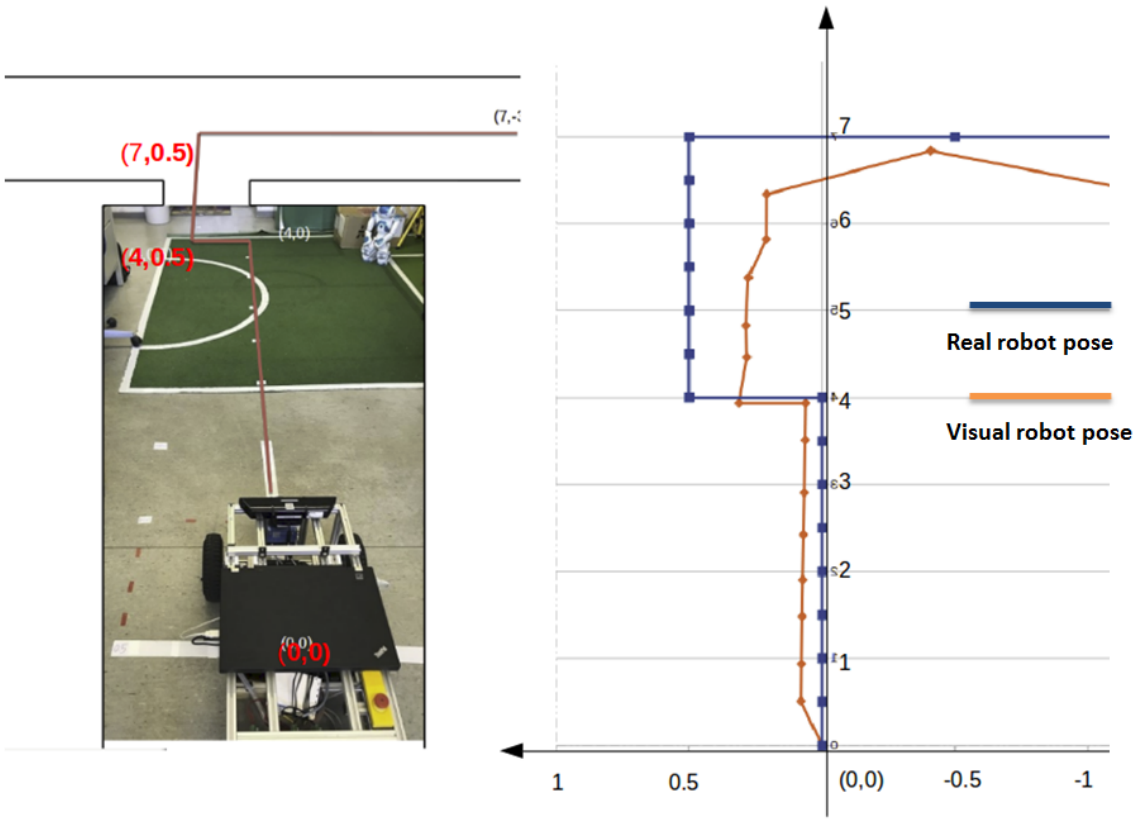


FIGURE 5.5. A comparison between real robot pose and the visual robot-pose display in ROS visualization Rviz

5.4.3 Loop Closure Detection

Loop closure detection test was carried out on the second floor of building 8, Frankfurt University of Applied Science. This environment provides a good mix of featureless and rich-featured surroundings. An additional challenge must be dealt, when many students are walking in the hallway. There are loops in the environment that allow testing of vision based loop closures and odometry error correction[9].

The first large-scale test run revealed two problems with the implementation, the first being odometry errors and the second being a failure to visualize detect loop closures. Odometry based on wheel encoder fails to indicate correct heading changes when rounding corners. From experiments, it is helpful to start a mapping session in an area that has rich and distinctive features. The map and floor plan comparison shown in figure 5.6 shows a map where a loop

closure was successfully detected. In this figure the Match ID is 26, which means there are 26 features matched between two pictures. The pink circles represent the matched features between new and previous patterns.

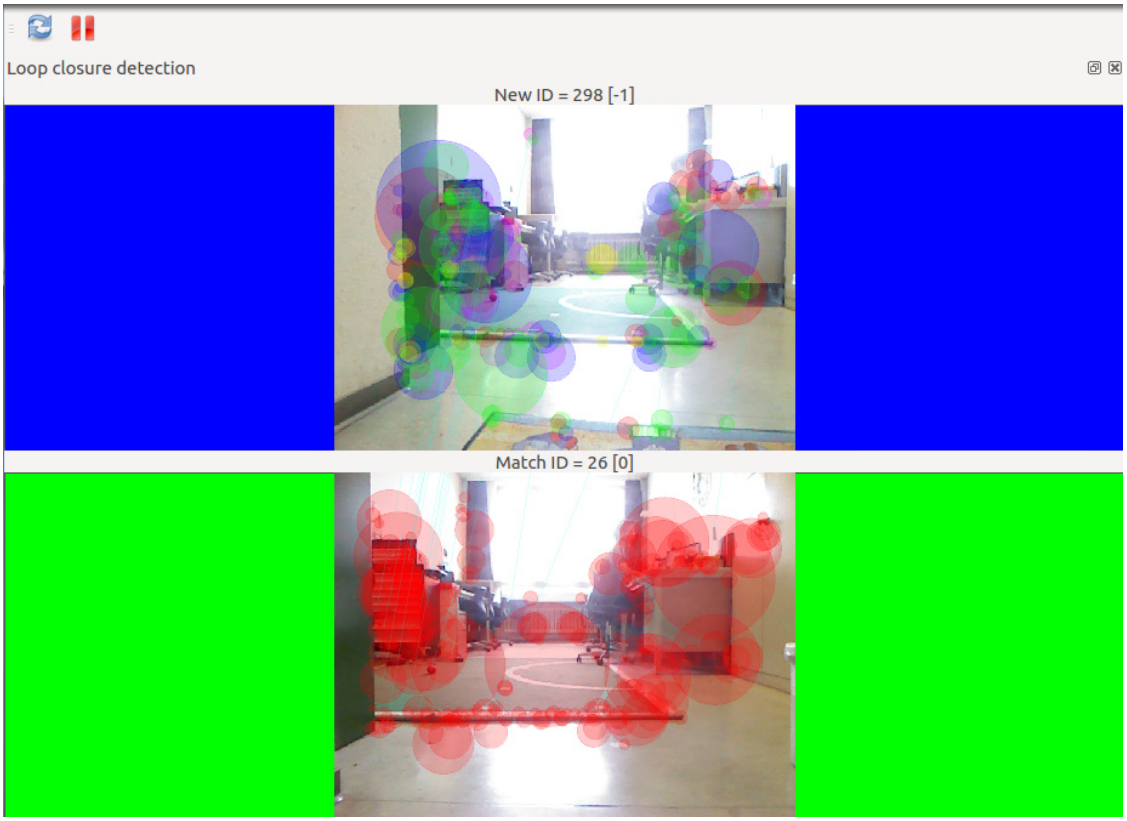


FIGURE 5.6. An example of an accepted loop closure hypothesis during a live mapping session. As before, the matched features are indicated by the pink circles.

5.4.4 Autonomous Navigation

Initial navigation session was dedicated to tune the navigation stack. Changes were made for the parameters of the local planner. The minimum speed was increased to overcome frictions. The goal tolerance was increased up to 30cm xy-tolerance and 0.25 radians yaw tolerance.

5.4.4.1 Navigating Among Static Obstacles

The first live navigation trials were performed by setting up a static obstacles course in a corridor. Most navigation trials were carried out in mapped areas. The robot performed well when navigating in known environment. In unknown environment, the robot will still plan an optimistic path, but if it is unable to map or detect new obstructions, it will collide.

5.4.4.2 Autonomous Navigation

In this section, the robot will make a challenge to pass twelve Hudora Pylons which is arranged like in the racing. The move_base packet and global cost map were used to detect those static obstacles. The picture of Hudora Pylons was shown in fig.5.7:

FIGURE 5.7. Hudora Pylons, static obstacles of the Volksbot and the trajectory the robot navigate in the test

The purpose of this test is to check the accuracy of global_cost_map of move_base package, also to check whether in this test, the width of inflated area of obstacles is safe enough or not. Pylons was put randomly with 1m distance between each other, Volskbot will run from initial position to the goal and need to pass through the obstacles without hitting them.

The test was divided in two cases:

- First case: the robot is allowed to move around the environment to draw a map of the whole area. Then this map will be used by navigation_stack package to make a path planning for the robot. In this case, robot makes the path planning and navigates very well, all the obstacles were avoided.

- Second case: the robot will be moving in unknown environment, map and location of robot will be updated by RTAB-Map provided to Navigation_Stack package. In this case, robot can avoid only three obstacles and hit the fourth one, the reason comes from the delay in transferring updated map from RTAB-Map to Navigation_Stack package.

Problems for the robot autonomous navigation of the robot are due to the fact that the environment is not fully observable by the Kinect camera or laser range scanner. The scanner is mounted on the fixed height above the floor and only scan a 1D line. As a result, some objects with minor height cannot be observed. This can lead to situations, where the execution of a navigation task has to be stopped immediately, to avoid for damaging the robot and its sensors. To solve this problem, the laser scanner is mounted in front-bellow the robot, so it can scan the obstacles with minor height.

Another problem with navigation_stack move_base is Volksbot stays in the range of laser beam, naturally the laser will detect the robot as an obstacle, and when the robot moves in 10 minutes, the map will be full of the obstacles with robot's footprint. There are two ways to solve that problem.

- **Limited the range of SICK laser scanner**: In this method, ROS node laser_filter was used to make a threshold for laser range, so the laser scanner only scans from 0.65m radius to infinite. This method is used to make the laser not detect robot as an obstacle. However, the problem of this method is robot also cannot detect obstacles in 0.65m radius.

- **Limited angle of SICK laser scanner**: Because the limited result in previous method, a new method was consider! It is a decrease in the scan angle of laser scanner from 270° to 180°. Although this method limits to observe the ability of the robot, however, it solves the current problem that robot not detect is as an obstacles while can avoid obstacles in 0.65m radius.

  In this test, robot got good result, it can avoid the obstacles smoothly when autonomous navigation. Nevertheless, in case the space is too narrow or people walk near the robot, it will stuck in the local map and start the recovery mode.

5.4.4.3 Compare the map created by robot with the real map

In this test, robot will move along corridor and three rooms in building 8, floor 2, to draw a map of the building. Robot is equipped a Kinect Xbox 360 camera to provide 3D point cloud, a SICK LMS100 laser scanner to provide 2D occupancy grid map and wheel encoder to provide odometry. It was controlled by a keyboard from a remote computer. Robot starts to move from room 206b , crosses the door which symboled on the real map as "RD-Tur", moves to room 206a and explores this room, then crosses the RD-Tur to create a map of the main stairs area before runs back to the initial point. While robot moves, the 3D-map updates continuously by RTAB-Map.

The real map of the main stair and the corridor of building 8, floor 2, Frankfurt am Main University of Applied science taken from University floor's map is shown in figure 5.8. The map includes a main stair area in the middle and two RD-Doors link at left and right corridors. At the left corridor there is a small corner at the right side linking to the seminar room.

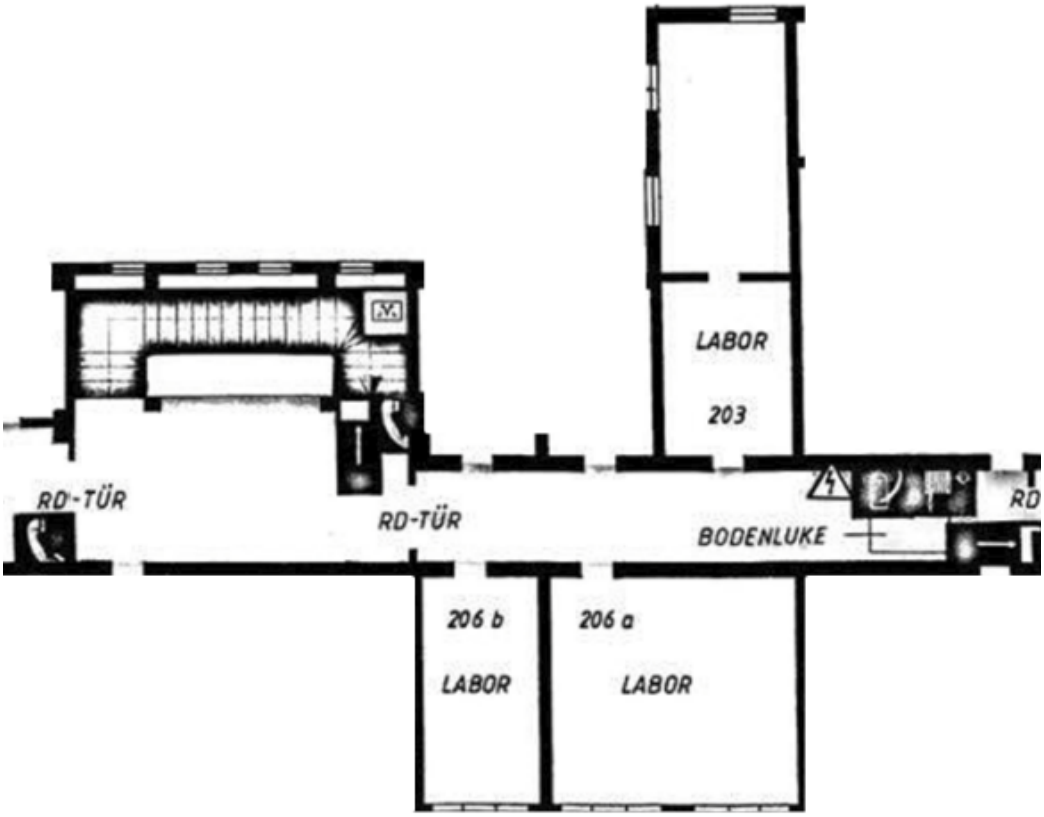


FIGURE 5.8. Real map of floor 2, building 8, Frankfurt University of Applied Science.

Fig.5.9 and 5.10 show 3D-map of similar corridor to fig.5.8 but the 3D. Map was created by RTAB-Map using Volksbot RT3, the 2D laser scan was used to provide ICP_odometry and Kinect Camera for display 3D part.

The blue line in fig 5.10 represents the trajectory of robot. Robot was started in room 206a, then it moved from room 206a to room 206b, moved a circle to creat a map of the room 206b before moving out to the corridor. Next, robot will run along the corridor to room 203 before it moved back to initial position. While it was moving, the 3D map updated continuously by RTAB-Map. The red and yellow line between the trajectory of robot represented when robot moved to areas which already visited, in this situation, the loop closure detection was triggered the graph- optimization to avoid robot graph overlapping.

The following command was used to launch RTAB-Map to create 3D-map in fig 5.12 and 5.13.

```
<Linux command to launch RTAB-Map>
```

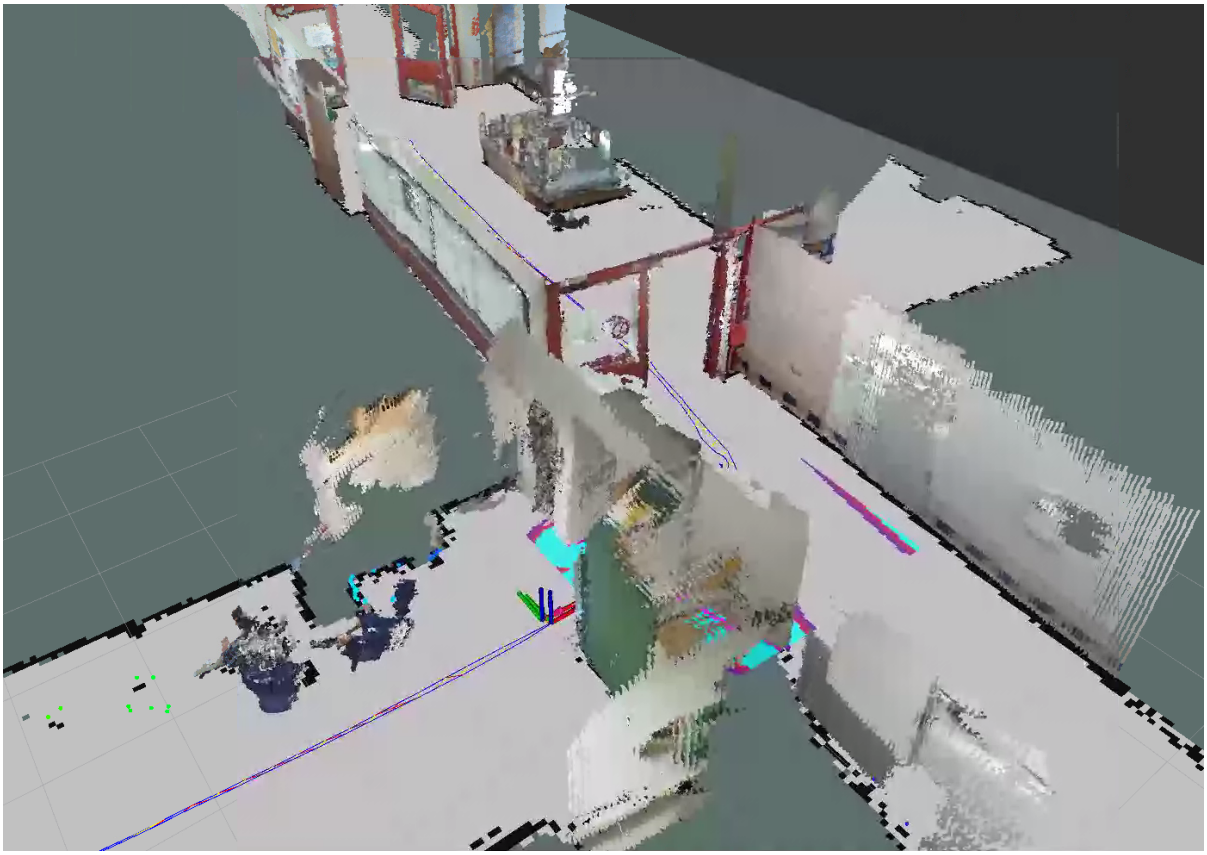


FIGURE 5.9. 3D- Map create by RTAB-Map using Volksbot platform

```
$roslaunch volksbot_bringup.launch
$ roslaunch volksbotdisplay.launch
$roslaunch rtabmap_ros navigation.launch map_topic:=/rtabmap/grid_map
$rosrun tf static_transform_publisher 0 0 0 0 0 0 /base_footprint /camera_link 100
$rosrun tf static_transform_publisher 0 0 0 0 0 0 /base_footprint /laser 100
```

There are some parts of 3D map not clear, specifically the objects on the wall which were too high and out of camera range. Remember that Kinect camera was mount on the top of robot, which only 1.5 meter from the ground.

The figure 5.11 is similar but now this map was created by Volksbot using SICK LMS 100 and Hector SLAM ros_package. The purpose of this test is to study the behavior of the SLAM package in the absence of perfect simulated conditions. The result is, in the real environment with noise odometry and range sensing information not 100% perfect, Hector SLAM still gives a good result, about 90 % information of the real map was created.The only problem is when robot moves in the corridor in a long time, the map border has a small error compared with the real map, this error may be caused from scan-matching error in some position while robot moves.

The following command describe how to launch to file Hector_map.launch in the to create

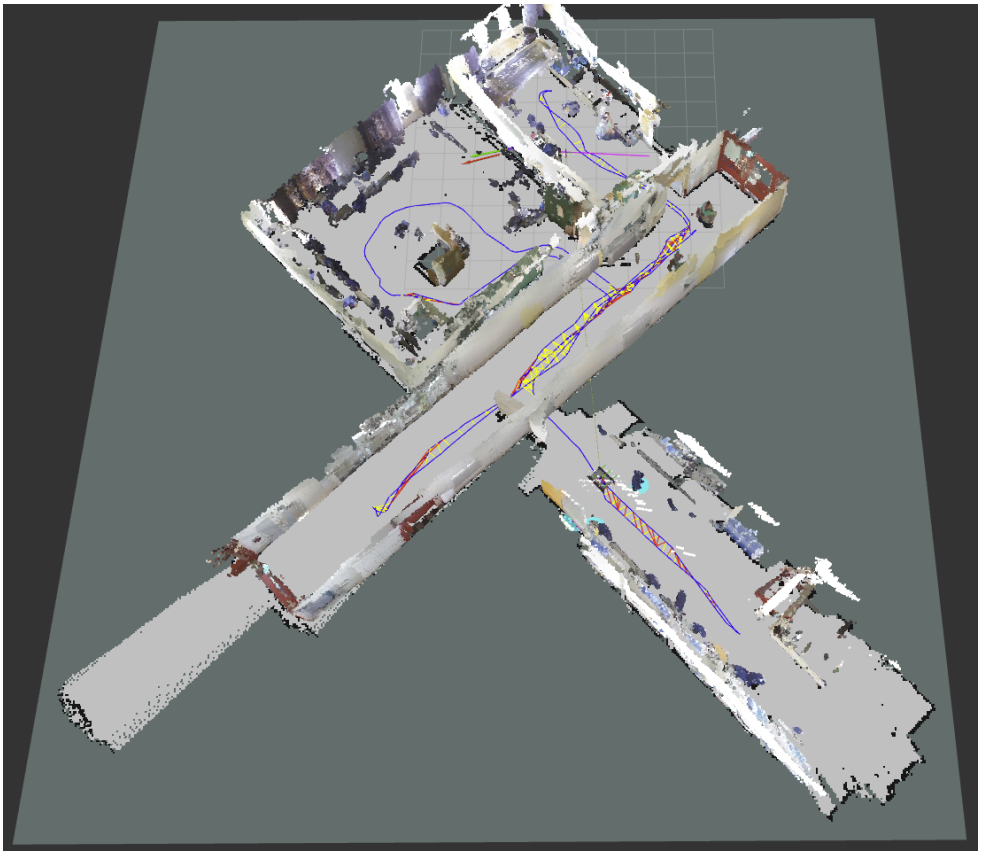


FIGURE 5.10. 3D- Map create by RTAB-Map using Volksbot platform

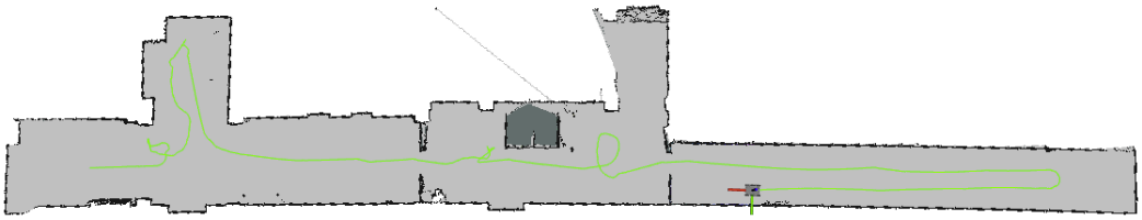


FIGURE 5.11. 2D-Map create by SICK laser scanner use Hector SLAM navigate by volksbot platform .

2D-map in fig 5.11.b.

```
$ roslaunch hector_map
$hector_mapping     hector_map_server  hector_map_tools
$ roslaunch hector_mapping
```

CHAPTER 6

DISCUSSION AND CONCLUSION

This chapter will assess the implementations presented in chapter 4 based on their performance which is detailed in chapter 5. The following discussions will qualitatively evaluate how can implement robot better in the future. The first section will discuss the performance of the system as a whole. Next follows a discussion on the success and weakness of the navigation and mapping systems. Finally, the author will generalize the most important achievement of this project, and suggest further research for future.

6.1 Overall Assessment

The overall system, as it is at the end of this master's project, is a functioning proof of concept for a mobile autonomous robot. All planned modules were implemented including Kinect Xbox360, power supply, upgrade Volksbot platform and ROS. However, some of them are only capable of demonstrating basic functionality and the possibility of more robust and complete implementations still in progress. The most important features, the combination of Hector-SLAM and RTAB-Map for mapping and navigation, were successfully implemented and configured. The current implementations are acceptable, but they still can improve in terms of robustness, parameter tuning and functionality. For instance, an application for robot should be better than run a list of command on Linux terminal to start the robot.

6.2 Choice of Development Tools

Using ROS as a development framework might have been the most important factor that contributed to a functional solution. In the end, ROS proved to be a flexible and rich tool, despite its novel structure and initial learning curve. Almost experienced users of ROS will be able to

rapidly implement and test robot concepts. The node structure in ROS is also a good way of structuring the entire system into self contained, manageable and reusable modules. This makes it easier to reuse parts of this implementation in later projects, and will hopefully benefit ensuing projects on this topic.
In addition, ROS also has a lot of projects about robots, which are developed by many universities in all over the world. Thanks to its **Open Source**, all of these projects are continuously updated on GitHub and can be shared between the universities. In future the Rowistha-arms system can transform from Labview to ROS to integrate with the base-link developed in this thesis. Two weeks were spent in upgrading the robot platform to house the various robot equipment, including the 12V Panasonic battery for Kinnect-cameras and on-board computer. For internal power, the robot uses a 24 V car battery as a power source. However, the battery only can use in 5 hours and then it needs to charge in about 1 hours. Therefore, it would be better if the Kinnect could be supplied directly from the robot power supply.

6.3 Strengths and weaknesses of RTAB-Map

The results of the mapping session demonstrate both strengths and weaknesses in the chosen mapping method, RTAB-Map. They show that multi session mapping works rather well in the environment with a sufficient amount of detectable visual features.
Strengths: RTAB- Map has many significant strengths. Firstly, it is packed with features, parameters and useful tools. It supports many sensor configurations, including stereo cameras. The ROS wrapper makes it easy to integrate the method into an existing robot system. The developer or user has an access to hundreds of parameters to tailor and fine-tune the mapping system. Object recognition and 3D obstacles detection is also useful features for supporting the assistant robot. Next, RTAB-Map works with any RGB-D sensor. It is possible to use both an active camera and a stereo camera, and switch between them, based on the conditions. A stereo camera is better suited for outdoor mapping than a Kinect, due to the presence of IR wavelengths in sunlight.
Weaknesses: Appearance based loop closure detection with RTAB-Map has many confirmed and potential weaknesses that must be addressed. The first problem of RTAB-Map is an incorrect loop closure detection followed by a subsequent incorrect odometry correction of the previously visited locations. To illustrate, if the robot moves in the featureless environment, or it turns too fast for Kinect camera to capture the frames clearly, the error in loop closure detection will happens.
Another weakness comes from the old legacy Kinnect Xbox 360. Because of its blind area up to 0.8m radius, a dynamic object which suddenly appears before the robot in 0.1m distance, cannot be detected by Kinect, so it can be hit. In case it is detectable, but the distance between Kinect and that object is still so small that the robot misunderstands it is a featureless image, the robot also cannot calculate its visual-odometry and the program will be interrupt.

6.4 Achievement

Even though it remains some weaknesses, RTAB-Map is still a reasonable choice for this thesis as it helps to reach a lot of significant achievement. First of all, robot can receive data from Kinect camera, Laser scanner and also from wheel encoder using their ROS_drivers to produce the most precise map. Besides, robot can receive control command via remote computer, the range to control the robot is up to 30 meters. Moreover, RTAB-Map and Hector-SLAM successfully implemented on robot so that it can create 3D and 2D map in large environment. Autonomous Navigation was also tested on both the simulator and the real robot, both got good results. Last but not least, robot can navigate to the goal and avoid the obstacles using global and local cost map.

6.5 Limitation

The problem that the loop closure detection does not work in case robot navigates in featureless environment still needs to be solved later. In autonomous navigation, robot cannot avoid the obstacle lower than the laser scanner. In a narrow area, robot is also usually stuck in its local cost map because the robot's path is blocked by the inflated area. In addition, the Kinect camera have the blind area up to 0.8 meter, therefore if an obstacle appear in this range, robot can collision with that obstacle.

6.6 Final Conclusion

Project Objective

The main objective of this project is to study SLAM algorithm and implement it as a solution for Volksbot autonomous navigation. Therefore, choosing an appropriate SLAM is the main mission because it is one of the fundamental requirements for a mobile robot. To meet the objective, the robot was configured to use ROS together with RTAB-Map. An additional major implementation goal is to achieve autonomous navigation. The third objective is to implement a control station where an operator can monitor and control the robot via a wireless connection.

Implementations

RTAB-Map is configured to use a Kinect for Xbox 360 in combination with a SICK laser to generate 2D occupancy grid maps and 3D point cloud maps of the environment. The current implementation is capable of building maps over multiple sessions. Scan matching from Hector SLAM provides the odometry to RTAB-Map. The robot has been configured to use the navigation stack in ROS. The navigation stack configuration enables the robot to plan and follow a path to a simple feasible goal. Additional capabilities include dynamic re-planning in case of existence of obstructions, and 3D obstruction detection based on point clouds. Two sources of control inputs, beside of the navigation stack, were implemented: keyboard control for simple testing

and command over team viewer from the remote computer.

Performance and Assessment

Testing sessions, which performed with both a simulator and a real mobile robot prototype, demonstrated RTAB-Map‚Äôs ability to build maps and localize the robot within these maps. Loop closure detection could work when a sufficient amount of visual features were available. Using laser scans as a source of odometry was susceptible to errors in featureless areas, when people walked by or towards the robot. Navigation tests on the real robot demonstrated that the robot can navigate successfully in known and structured environment. The performance of navigation performance get worse in cluttered environment, e.g. office environment with many tables placed closely together.

6.7 Recommendation and Future Work

For better performance, the mobile base requires an improvement, and a new base_link should be considered, because the indoor navigation robot should have a small base_link in square or circle, to easily move in a narrow space, and easier to escape when robot stucks somewhere. A new drive system should be dimensioned to support the robots weight and be more rugged, so it can handle uneven surfaces.

The Kinect and the LIDAR is currently placed in the front of the base. Since the Kinect is unable to reliably measure distance closer than 0.8m, a new sensor location should be considered to avoid this blind-zone. And the Kinect for Xbox-360 has created point-cloud for 3D map in bad quality. A new version Kinect like KinectV2 is recommended to replace Kinect Xbox 360. They will bring the better quality of the 3D map.

The implementation presented in this thesis is the first attempt for integrating the mobile robot prototype with ROS. It concludes that ROS is a good tool for prototyping, and it is recommended to continue using the framework in subsequent projects on robotic maintenance. The current software is functional, but not much more. Future projects should strive to increase the system's usability and robustness.

Continue work on This Project: The only hardware requirement to continue the work where this author left off, is a new on-board computer running ROS Kinetic, because after the thesis finishes, the on-board laptop will be return to the University. The mobile base will also require an upgrade to have more space for on-board laptop and battery for Kinect, and Kinect for Xbox 360 should be replaced by Kinect for Xbox One.

Further Testing and Assessment of RTAB-Map: Improve the wi-fi quality will help to create a good map in large scale environment. Until now the robot navigates to draw map inside building 8, include room 206a,b and the corridor because the wifi signal will disconnect if the robot moves out the 20 meter range. If it is possible, the next purpose is to draw a map of all campus Nibelungen Platz/FH. RTAB-Map can be used with stereo cameras and Kinect cameras,

which could be useful to compare passive and active depth sensors. The advantages of stereo camera is that robot can observe in underexposed environment. Therefore, the stereo camera is used instead of Kinect when mapping in outside environment.

APPENDIX A

DESCRIPTION AND CONFIGURATION OF MOVE_BASE

A.1 Description of the move base

The move base is a node of the ROS distribution, which provides interface for configuring, running and interacting with node *Navigation_Stack* for autonomous navigation in dynamic environments. Figure B.1 shows the different subtasks of the move base and how they interact with each other.

If the move base receives a new goal, it plans a path from the current pose of the robot to the goal pose. This global plan will be calculated using the Dijkstra algorithm. The planning of the global path is based on the information of the global costmap, which forces the planned path away from obstacles. The global plan will only change, if the local path planner notifies that an obstacle is in front of the robot or the global map changes. The local path planner determines the velocity commands to execute the navigation, based on its local cost map, which also detects obstacle in the environment.

For the case that the robot get stucked, the differrent recovery behaviours are executed. Figure A.2 show in which order the behaviour are executed.

At first, the local costmap will be cleared and built up new from this point. After that, the robot can navigate again. If not, the robot performs a 2π turn in place and build up the costmap new. The aggressive reset clears both costmaps. If the navigation fails after the second turn in place, the path planning will be aborted(something blocks the robot, or the target in within the inflated area of the cost map) and the robot waits for the next target.

A.1.0.1 Configuration of move base

The next sections explain how to configure the different components of the move base to use it in different global coordinate frames.

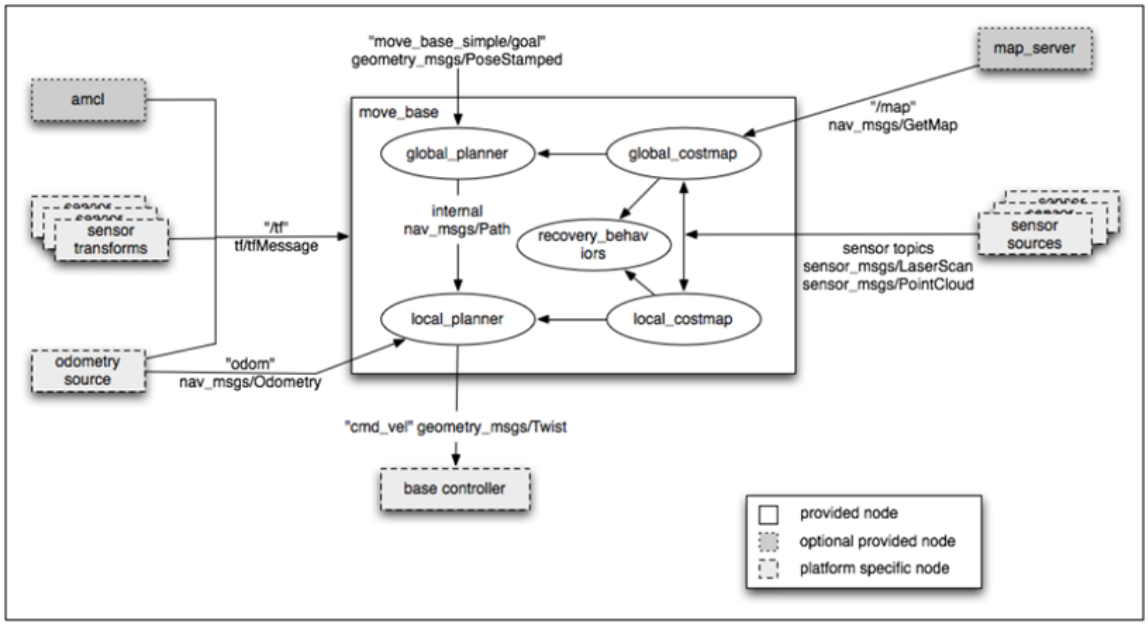


FIGURE A.1. Summary of the move base and how the different subtasks of the move base interact with each other. Source of figure: http://wiki.ros.org/move base

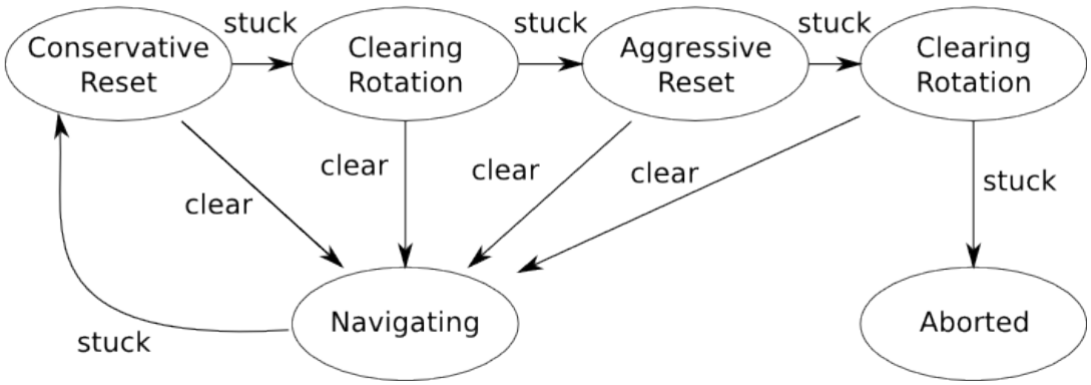


FIGURE A.2. Summary of the executed behaviors to unstuck the robot. Source of figure: http://wiki.ros.org/move base.

A.1.0.2 global-planner

This package provides libraries and nodes for planning the optimum path from the current position of the robot to the goal position, with respect to the robot map. This package has implementation of path finding algorithms such as A*, Dijkstra, and so on for finding the shortest path from the current robot position to the goal position.

local-planner: The main function of this package is to navigate the robot in a section of the global path planned using the global planner. The local planner will take the odometry and sensor reading, and send an appropriate velocity command to the robot controller for completing a segment of the global path plan. The base local planner package is the implementation of the trajectory rollout and dynamic window algorithms.

rotate-recovery: This package helps the robot to recover from a local obstacle by performing a 360 degree rotation.

A.1.0.3 base local planner

The base_local_planner is package of ROS, which provides a controller to move a robot in the plane. This controller severs to connect the path planer to the robot. Using map, the planner create a kinematic trajectory for the robot to get from a start to the goal location. Along the waym the planner create, at least locally around the robot, a value function, represented as a gird map. This value funtion encodes he costs if traversing through the grid cells. The controller's job is to used this value function to determine d_x, d_y, d_θ velocities to send to the robot.The important parameters are given in the upcoming listing:

- **holonomic robot**: Specifies, whether the robot can perform movements in the y-direction. In this thesis, the Volksbot RT3 is not able to perform movements in y-direction. Therefore, this parameter is set to false for every configuration.
- **yaw goal tolerance**: The tolerance for the heading, given in radians.
- **xy goal tolerance** : The tolerance for the position, given in meters for every component.
- **pdist scale**: Defines the weight how much the robot should follows the global path.
- **gdist scale**: Defines the weight how much the robot should reach its calculated control actions.
- **global frame id**: The frame to set the costs. Should be the same as for the localcostmap.

the *base_local_planner* is configured as shown in table A.1

Parameter	Value	Why chosen the specific value?
yaw goal tolerance	/0.2	To handle uncertainties during acquisition of the map.
xy goal tolerance	/0.3	Set to a big value, to handle uncertainties during acquisition of the map. If set to a smaller value, the robot can not turn in place.
pdist_scale	1.5	The value of this parameter is set bigger then the value of gdist scale to force the robot on the global planned path.
gdist_scale	/0.8	Could not be zero, to navigate the robot safely through doors.
global_frame_id	/odom	This frame is specified as global frame for the costmap.

TABLE A.1. Summary of the parameters for the local costmap in context of the active exploration task.

A.2 Costmap Parameters

The upcoming listing explains the important parameters for the costmaps:

- **global frame**: Defines the frame in which the costmap operates in global context.
- **robot base frame**: Defines the base frame of the robot.
- **transform tolerance** : The tolerance for the position, given in meters for every component.
- **update frequency**: The frequency for updating the map.
- **publish frequency**: The frequency for publishing the map for displaying it in rviz.
- **obstacle range**: The maximum distance from the robot to an obstacle, where the ob-stacle will be inserted into the map.
- **global frame**: Defines the frame in which the costmap operates in global context.
- **raytrace range**: The maximum distance to remove obstacles from the map.
- **cost_scaling_factor**: Factor for scaling costs during inflation.
- **inflation radius**: Radius around an obstacle for inflating it on the map with costs.
- **footprint**: Defines the principle physical properties of the robot.
- **robot radius**: Radius of the robot. Can be calculated as follows: $\sqrt{\Delta x^2 + \Delta y^2}$, where Δx is the length of the robot and Δy is the width of the robot.

- **observation sources**: Defines a list of sensor sources for observing the environ- ment.
- **observation source/topic** : The name of the topic, where the observation source publishes its data.
- **observation source/sensor frame**: The frame of the sensor data. Required for transformation.
- **observation source/expected update rate**: The expected update rate of the sensor. Best way to determine is that via the tf monitor and using the average time and additionally a buffer for it.
- **observation source/data type**: The type of the sensor data, e.g. laser for a laser scanner.
- **observation source/marking**: Specifies, whether the sensor can add obstacles to the costmap.
- **observation source/clearing**: Specifies, whether the sensor can clear obstacles from the costmap.
- **static map** : Specifies, whether a costmap is static or not. publishes its data.
- **rolling window**: If static map is set to true, this parameter has set false.
- **map topic**: The name of the topic, where the costmap can subscribe the static map.
- **width**: The width of the costmap, given in meters.
- **height**: The height of the costmap, given in meters.
- **resolution**: The resolution of the costmap, given in pixels per meters.
- **origin x and origin y**: The origin of costmap in x-direction and y-direction.
- **map type** : The type of the map. Set to costmap in every configuration of this thesis. publishes its data.
- **track unknown space**: Specifies, whether to track cells, which never observed yet.

A.2.0.1 Costmaps for Active Exploration

In the case of exploration, the occupancy grid map will be build up succesivly by the *GMapping* node. Therefore, the move base have to be configured in the way that it is able to run in conjunction with the *GMapping* node.

Local Costmap Global Costmap

Parameter	Value	Why chosen the specific value?
global frame	/odom_combined	Global frame for navigation in local context. Uses the odometry.
robot base frame	/base_footprint	Defined frame of the robot. See transformation tree in figure 4.9.
inflation radius	0.3	Has to set minor than the inflation radius of the global costmap that the robot is able to drive through doors, due to uncertainties during the process of mapping.
update frequency	/5.0	High update rate of the local costmap to be sure that dynamic obstacles are perceived as soon as possible and that the local costmap is updated as soon as possible, if the map provided by the GMapping node is updated.
publish frequency	/1.0	Publishes the local costmap every second for the scope of visualization. Will not be done more often to avoid overload of the system.
static map	/false	Non static map, due to the fact that this map is used for the local field of view of the robots sensor
width	/2.5	Using a minor value to avoid overload of the system and to handle uncertainties arose by using the move base in conjunction with GMapping.
height	/2.5	Same as before for the width of the costmap..

TABLE A.2. Summary of the parameters for the local costmap in context of the active exploration task.

Parameter	Value	Why chosen the specific value?
global frame	map	Global frame, by using GMapping and navigation in this frame.
robot base frame	base_footprint	Defined frame of the robot. See transformation tree in figure 4.9.
inflation radius	0.3	Has to set minor than the inflation radius of the global costmap that the robot is able to drive through doors, due to uncertainties during the process of mapping.
update frequency	/5.0	High update rate of the local costmap to be sure that dynamic obstacles are perceived as soon as possible and that the local costmap is updated as soon as possible, if the map provided by the GMapping node is updated.
publish frequency	/1.0	Publishes the local costmap every second for the scope of visualization. Will not be done more often to avoid overload of the system.
static map	/false	Non static map, due to the fact that this map is used for the local field of view of the robots sensor
rolling window	/true	Have to be set true, because the parameter before is set to false.
width	/50.0	Using big value, to be sure that the path planning is consistent over the whole map.
height	/50.0	Same as before for the width of the costmap

TABLE A.3. Summary of the parameters for the global costmap in context of the active exploration task.

APPENDIX B

CREATED ROS-LAUNCH FILES

B.1 Created ROS Nodes

This package contains different base classes, e.g. for any probabilistic filter. It also contains classes, which provide different operations and functions that all other nodes are use. Additionally, the different sensor and related operations are modeled by own classes and test classes are available.

The following launch files was create to turn on the Volksbot, Kinect, SICK laser scanner and link them together.

- **volksbot_bringup.launch**: Start the volksbot_nodem turn on the volksbot odometry extract from wheel encoder .
- **volksbot_bringup_onlylaser.launch**: Start the volksbot_nodem turn on the volksbot odometry extract form laser.
- **volksbot_bringup_onlykinect.launch**: Start the volksbot_nodem turn on the volksbot visual odometry extract form kinect.
- **SLAM.launch**: Start node rtab_map_node and rgbd_odometry_mode in the rtabmap_ros package.
- **navigation.launch**: Start node move_based in the navigation_stack package.

B.2 Communication of ROS Nodes

The communication between nodes in launch files

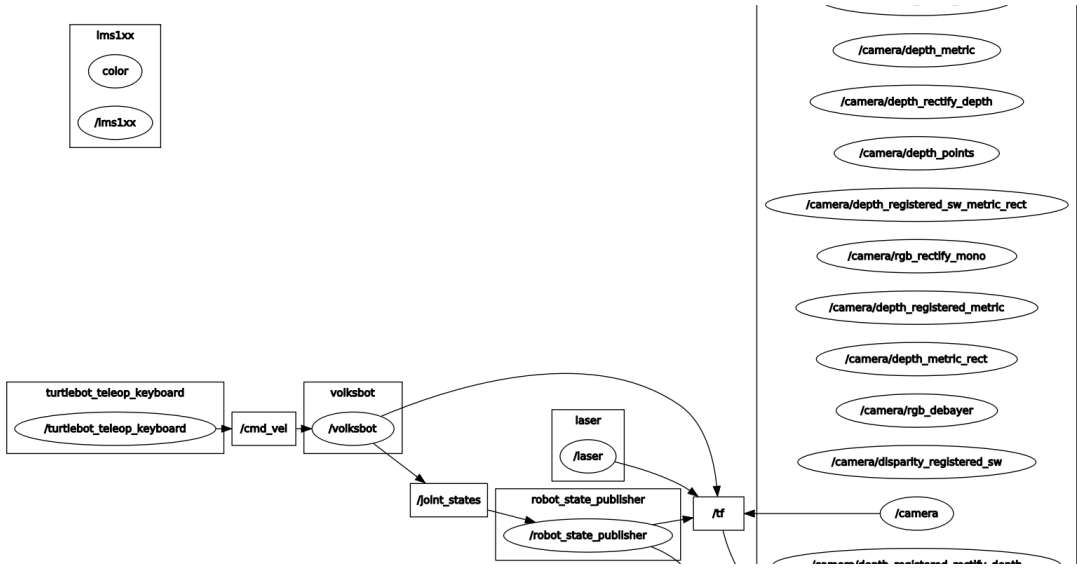


FIGURE B.1. Communication the ROS nodes for volksbot_bringup.

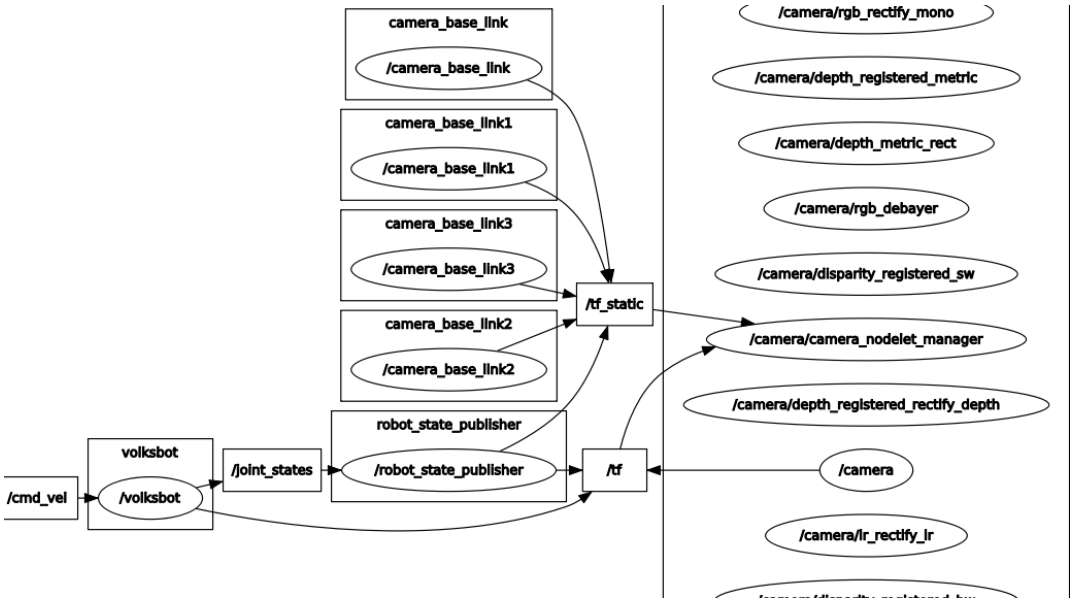


FIGURE B.2. Communication the ROS nodes for volksbot_bringup_Kinect

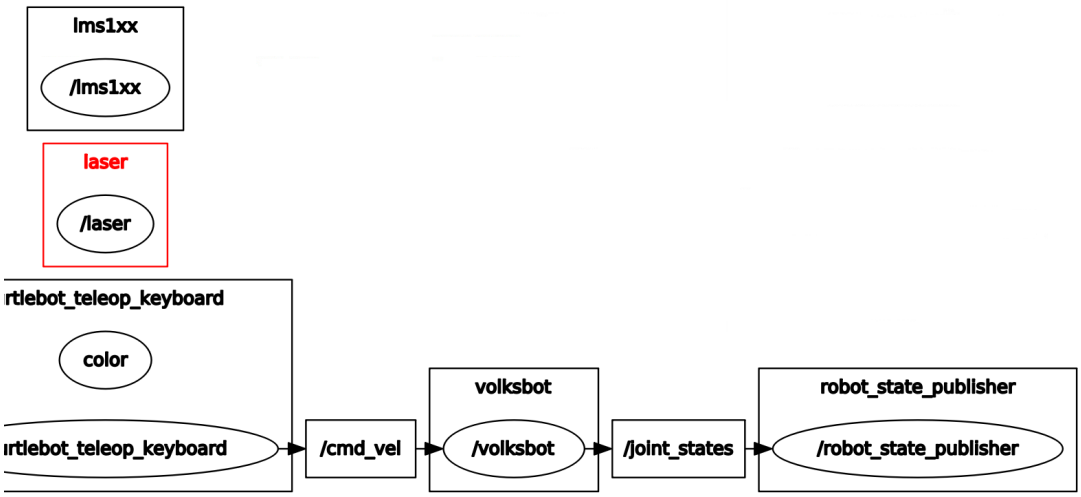


FIGURE B.3. Communication the ROS nodes for volksbot_bringup_Laser.

B.3 Start the different Tasks

The upcoming section describes which launch files have to be started in which order to execute the different algorithms with the Volksbot RT3 or as a simulation.

Active Kinect_Camera

This start the driver for the Kinect Xbox 360, as well as turn on the camera_depth_optical_frame, which can be estimate the 3D position of each point in the the 2D picture which capture by the camera.

Create a transform between base_link and camera_link

This command will cerate a static transform tf_link between the the based, the camera, and Laser SICK lms1xx of the volksbot.

```
$rosrun tf static_transform_publisher 0 0 0 0 0 0 /base_footprint

/camera_link 100
$rosrun tf static_transform_publisher 0 0 0 0 0 0 /base_footprint

/laser 100
```

Starting the volksbot_drviver, go to the folder launch in the stack volksbot_driver and execute the following command:

```
$roslaunch volksbot_driver volksbot.launch
```

After volskbot turn on, we need to check all the component include laser, kinnect and wheels odometry working or not. Type the following command to see the data come from components or not:

```
$rostopic hz /odom
$rostopic hz /scan
```

```
$rostopic hz /camera/rgb/image_rect_color
$rostopic hz /camera/depth_registered/image_raw
```

Then start the map program by:

```
$roslaunch rtabmap_ros rgbd_mapping.launch frame_id:=base_footprint
rtabmap_args:="--delete_db_on_start" rtabmapviz:=false subscribe_scan:=true
```

To turn on/of localization mode of rtabmap, we just add the command to rtabmap.launch

```
roslaunch rtabmap_ros rgbd_mapping.launch frame_id:=base_footprint
rtabmapviz:=false subscribe_scan:=true localization:=true
```

At this time, you can move the robot by teleoperate from key board to draw a map, robot also display it's localization in that map also. But robot can not move by itself, to make robot automonous, the package navigation stack must be run, and the following command will run the navigation.launch files which call the navigation stack.

```
roslaunch rtabmap_ros navigation.launch map_topic:=/rtabmap/grid_map
```

From this point, user can control the robot by just give the goal (x,y, theta) by button 2D Nav in Rviz. And robot will try to navigate this point by follow the path planning by Global planner. Like the figure B.4.

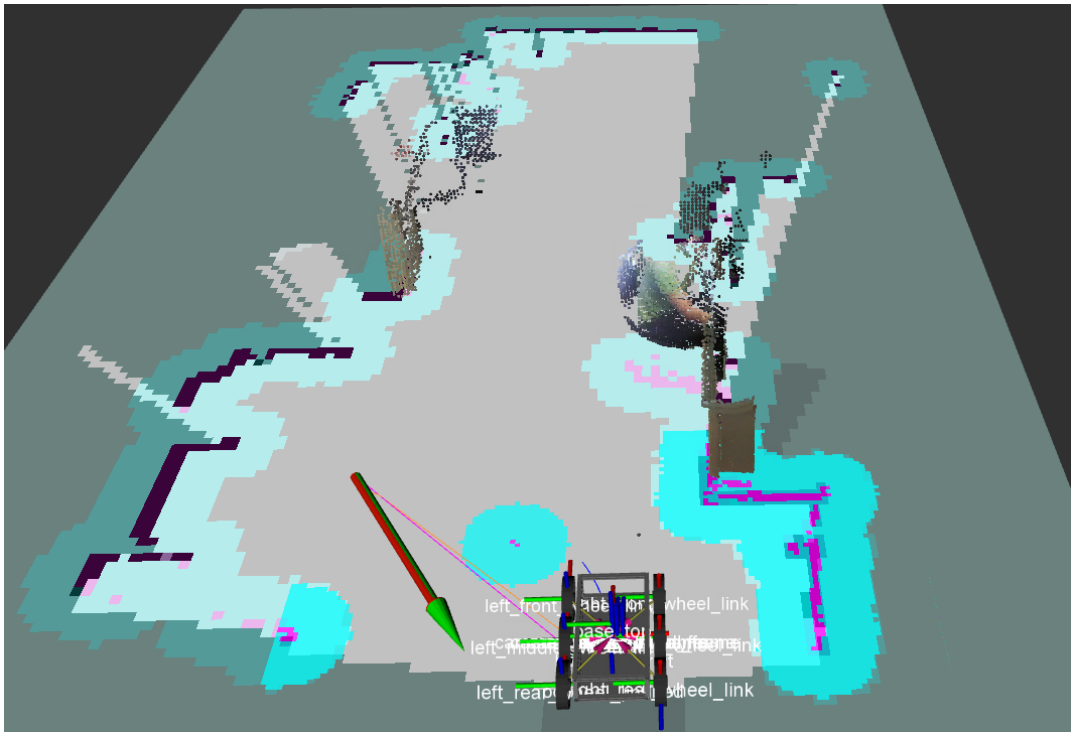


FIGURE B.4. Autonomous navigation with navigation stack.

APPENDIX C

FILTER USED IN THIS PROJECT

C.1 Probabilistic Filter - Bayes Filter

Probabilistic filters are filters, which represent the current true state at time t, using a belief $b(x_t)$ and can consist of any variable that has to be estimated.

All of those filters techniques have the Markov assumption first order as precondition. This assumption means that the current state x_t only depends on the previous state $x_{t,\text{àí}1}$, the current measurement z_t and the control data u_t. This is illustrated in figure 2.2.

In general, the Bayes filter consists of two steps:

Prediction step: During this step the control data, e.g. data of the odometry, will be used to predict the new belief. The equation for the prediction step is:

$$bel(x_t) = \int p(x_t|u_t, x_{t-1}).bel(x_t)dx_{t-1} = \sum_{x_{t-1}} (x_t|u_t, x_{t-1}).bel(x_t) \tag{C.1}$$

Correction step: After the prediction step, the current measurement zt will be used to correct the state. The equation for the correction step is

$$bel(x_t) = \eta.p(z_t|x_t.bel(x_t) \tag{C.2}$$

The correction step is also called as update step. η is a normalization value, to normalize $bel(x_t)$.

The discrete version of the Bayes filter is described as pseudo-code in algorithm 1. Accordingly to the Markov assumption, the input of the Bayes filter is the belief $b(x_{t,\text{àí}1})$, the current control data u_t and the current measurement z_t.

Algorithm 1 Recursive discrete Bayes filter

1: **procedure** BAYESFILTER($bel(x_{t-1}), u_t, z_t$)
2:
3: **for** all x_t **do**
4: $bel(x_t) = \sum_{x_{t-1}} (x_t|u_t, x_{t-1}).bel(x_t)$ ▷ Prediction Step
5: $bel(x_t) = \eta.p(z_t|x_t.bel(x_t)$ ▷ Correction Step
6: **return** $bel(x_t)$

C.2 Kalman Filter

The Kalman filter is derived from the Bayes filter and provides a concrete algorithm for estimating a state. The Kalman filter was published by R. E. Kalman in 1960 and solves the problem of prediction in state space for linear systems [13]. For the Kalman filter, an additional assumption, to the Markov assumption first order, will be made, namely that all believes will be represented by zero-centered Gaussian distributions. All equations in the following sections are given for the case that the state vector containing the pose of the robot and the measurement contains distance and bearing angle:

Linear and Time Discrete Kalman Filter:

Given is a linear and time discrete system with its system equation:

$$x_t = \mathbf{A}_t.x_{t-1} + \mathbf{B}_t.u_t + \sigma_t \tag{C.3}$$

x_t and x_{t-1} are the state vector at time t and $t-1$. u_t is a vector, obtainning the current control data. $\mathbf{A}_t$ describes the system dynamics and the matrix $\mathbf{B}_t$ adds the control data. The last term σ_t is the noise. The noise is a zero-centered Gausian with covariance matrix $\mathbf{R}_t$. This describe the uncertainties of the state transition. The prediction state vector given by:

$$\bar{x}_t = \mathbf{A}_t.x_{t-1} + \mathbf{B}_t.u_t \tag{C.4}$$

The uncertainies of the system have to predicted, too. The prediction of the uncertainties is given by:

$$\bar{\sum_t} = \mathbf{A}_t.\sum_{t-1}.A_t^T + R_t \tag{C.5}$$

Now, a measurement is available and the predicted state ca nbe updated. The measurement equation of the linear and time discrete system is give by:

$$z_t = \mathbf{C}_t.x + t + \theta_t \tag{C.6}$$

θ_t is the noise of the measurement and is a zero-centered. Gaussian with covariance matrix $\mathbf{Q}_t$. Using a recursive linear estimation, the state vector can be estimated. The result for x_t is:

$$x_t = \bar{x}_t + \mathbf{K}_t.(z_t - \mathbf{C}_t.\bar{x}_t) \tag{C.7}$$

The difference between z_t and $\mathbf{C}_t.\bar{x}_t$ is the innovation of the update step. The matrix K_t is a currently unknown matrix and is called Kalman gain and specifies how much the estimation of the state depends on the current measurement. The Kalman gain matrix is given by:

$$\mathbf{K}_t = \bar{\sum}_t.\mathbf{C}_t^T.(\mathbf{C}_t.\bar{\sum}_t.\mathbf{C}_t^T + \mathbf{Q}_t)^{-1} \tag{C.8}$$

Now, an algorithm for the Kalman filter can be defined. The algorithm requires the mean of the last state μ_{t-1} , the covariance matrix $\sum_{t-1}$, the control data u t and the measurement data z_t .

Algorithm 2 Algorithm of the linear and time discrete Kalman filter

1: **procedure** KALMANFILTER($bel(\mu_{t-1}), \sum_{t-1}, u_t, z_t$)
2:
3: **for** all x_t **do**
4: $\bar{\mu}_t = A_t.x_{t-1} + B_t.\mu_t$
5: $\bar{\sum}_t = A_t.\sum_{t-1}.A_t^T + R_{t-1}$
6: $K_t = \bar{\sum}_{t-1}.C_t^T.(C_t.\bar{\sum}_t.C_t^T + Q_t)$
7: $\mu_t = \bar{\mu}_t + K_t.(z_t - C_t.\bar{\mu}_t)$
8: $\sum_t = (I - K_t.C_t).\bar{\sum}_t$
9: **return** $\mu_t, \sum_t$)

The algorithim return the posterior belief, represented by the mean and the covariance of the Gaussian distribution.

C.2.0.1 Extended Kalman Filter(EKF)

In most cases, the system and measurement equations are nonlinear. The given system equation is denoted by $g(\mu_t, x_{t-1})$ and is nonlinear. The predicted system state is given by:

$$x_t = g(\mu_t, x_{t-1}) + \epsilon_t \tag{C.9}$$

The measurement equation is denoted by h_{x_t} and is nonlinear, too. Therefore, the measurement z_t is given by:

$$z_t = h(x_t + \rho_t) \tag{C.10}$$

ϵ_t and ϵ_t are the additive noise of the system and measurement process. To handle those nonlinear cases, the extended Kalman Filter 1. order linearizes the non-linear equation[9], using the Taylor series expansion and breaking up this expansion and omitting higher order terms, lead to:

$$g(\mu_t, x_{t-1}) \approx g(\mu_t, x_{t-1}) + \frac{\varphi_g(u_t, x_{t-1})}{\varphi.x_{t-1}}.\delta.x_{t-1} \tag{C.11}$$

The system equation depends on more than one variable that is part of the state vector x_{t-1} . Therefore, the matrix G_t will be introduced[19]. G_t is the Jacobian matrix of $g(u_t, x_{t-1})$ with

respect to the current state x_t . In particular, that means, $g(u_t, x_{t-1})$ will be approximated by its mean of the last known state μ_{t-1} and μ_t.

$$g(\mu_t, x_{t-1}) \approx g(\mu_t, x_{t-1}) + \mathbf{G}_t.(x_{t-1} - \mu_{t-1}) \tag{C.12}$$

The system function contains single equations for every component of the state vector in particular for the state vector containing x, y and θ:

$$g(\mu_t, x_{t-1}) = \begin{bmatrix} g_1(\mu_t, \mu_{t-1}) \\ g_2(\mu_t, \mu_{t-1}) \\ g_3(\mu_t, \mu_{t-1}) \end{bmatrix} \tag{C.13}$$

Therefore, the Jacobian matrix $\mathbf{G}_t$ is :

$$\mathbf{G}_t = \begin{bmatrix} \frac{\varphi.g_1(\mu_t, \mu_{t-1})}{\varphi.x} & \frac{\varphi.g_1(\mu_t, \mu_{t-1})}{\varphi.y} & \frac{\varphi.g_1(\mu_t, \mu_{t-1})}{\varphi.z} \\ \frac{\varphi.g_2(\mu_t, \mu_{t-1})}{\varphi.x} & \frac{\varphi.g_2(\mu_t, \mu_{t-1})}{\varphi.y} & \frac{\varphi.g_2(\mu_t, \mu_{t-1})}{\varphi.z} \\ \frac{\varphi.g_3(\mu_t, \mu_{t-1})}{\varphi.x} & \frac{\varphi.g_3(\mu_t, \mu_{t-1})}{\varphi.y} & \frac{\varphi.g_3(\mu_t, \mu_{t-1})}{\varphi.z}) \end{bmatrix} \tag{C.14}$$

The same will be done for the measurement equation, containing single equations for every measured variable - in particular for measuring a distance and a bearing angle, $h(x_t)$ is:

$$h(x_t) = \begin{bmatrix} h_1(x_t) h_2(x_t) \end{bmatrix} \tag{C.15}$$

The corresponding Jacobian matrix $\mathbf{H}_t$ is:

$$\mathbf{H}_t = \begin{bmatrix} \frac{h_1(x_t)}{\varphi.x} & \frac{h_1(x_t)}{\varphi.y} & \frac{h_1(x_t)}{\varphi.\theta} \\ \frac{h_2(x_t)}{\varphi.x} & \frac{h_2(x_t)}{\varphi.y} & \frac{h_2(x_t)}{\varphi.\theta} \end{bmatrix} \tag{C.16}$$

Therefore, the linearized measurement function is given by:

$$h(x_t) \approx h(\mu_t + \mathbf{H}_t.(x_t - \mu_t)) \tag{C.17}$$

Analogously to the linear and time discrete Kalman filter, the pseudocode for the EKF can be defined. The algorithm requires, like the Kalman filter before, the mean of the last state μ_{t-1} , the covariance matrix $\sum_{t-1}$, the control data u t and the measurement data z_t . The algorithm returns the posterior belief, represented by the mean and the covariance of the Gaussian distribution.

Algorithm 3 Algorithm of the linear and time discrete Kalman filter

1: **procedure** EXTENDEDKALMANFILTER($\mu_{t-1}), \sum_{t-1}, u_t, z_t$)
2:
3: **for** all x_t **do**
4: $\bar{\mu}_t = (\mu_t, \mu_{t-1})$
5: $\bar{\sum}_t = \mathbf{G}_t.\sum_{t-1}.\mathbf{G}_t^T + R_t$
6: $\mathbf{K}_t = \bar{\sum}_t.\mathbf{H}_t^T.(\mathbf{H}_t.\bar{\sum}_t.\mathbf{H}_t^T + Q_t)$
7: $\mu_t = \bar{\mu}_t + \mathbf{K}_t.(z_t - h(\bar{\mu}_t)$
8: $\sum_t = (\mathbf{I} - \mathbf{K}_t.\mathbf{H}_t).\bar{\sum}_t$
9: **return** $\mu_t, \sum_t$)

C.2.0.2 Kalman Gain

The Kalman gain K is computed to find out how much we will trust the observed landmarks and as such how much we want to gain from the new knowledge they provide. If we can see that the robot should be mov ed 10 cm to the right, according to the landmarks we use the Kalman Gain to find out how much we actually correct the position, this may only be 5 cm because we do not trust the landmarks completely, but rather find a compromise between the odometry and the landmark correction. This is done using the uncertainty of the observed landmarks along with a measure of the quality of the range measurement device and the odometry performance of the robot. If the range measurement device is really bad compared to the odometry performance of the robot, we of course do not trust it very much, so the Kalman gain will be low. On contrary, if the range measurement device is very good compared to odometry performance of the robot Kalman gain will be high.

C.2.0.3 Additional Kalman Filter Implementations

Also, an iterative version of the EKF first order exists, which produces more accurate results then the non-iterative version of the EKF. An introduction to the iterative version of the EKF first order is given in [15]. In addition, an EKF second order exists. The second derivative of the non-linear functional model will be calculated, using the Taylor series expansion and breaking at the second term. A description can be found in [15]. Last but not least, there is an additional Kalman filter approach for non-linear systems. This one is called Unscented Kalman filter (UKF). The UKF tries to linearize the non-linear functional model via an unscented transformation, instead of using the Taylor series expansion.

C.3 Particle Filter

The particle filter is an other derived filter of the Bayes filter and provides an algorithm for estimating the state, without the assumption of using a normal distribution for the beliefs. Every multi modal distribution can be handled by this nonparametric filter. The true state is estimated

by a set of particles χ_t . A particle χ_t represents one possible state of the true state. The index i addresses the particle of the particle set χ_t .

C.3.0.1 Algorithm

The principle algorithm of the particle filter is given as pseudo-code in algorithm 4. This algorithm requires the last known state, represented by a set of particles χ_t , the current control data μ_t and the current measurement z_t .

The particle filter iterates over the whole set of particles. At first, the prediction step will be applied to the data. Due to the sampling of a possible pose under the condition of the control data, uncertainties are handled too (approximates $\bar{bel}(x_t)$ from the equation 2.12). After that, the measurement will be used to proceed the correction step. The correction step returns a likelihood, which describes how well a measurement corresponds to the real world, according to the particle x_t. In the next step, the particles will be added with their corresponding weight to the set of particles. Finally, the particle set will be re-sample. In this step, particles with low weight will be likely to removed and particle with height weights will be likely to be duplicated. This approximate the $\bar{bel}(x_t)$ from equation 2.13[9].

The concrete implementation of the different steps depends on the used motion model, sensor model and the re sampling technique. Some possible models are discussed in the upcoming sections.

C.3.0.2 Re sampling Techniques

Re-sampling is the process of drawing M particles from t with replacement. Particles with high weights are more likely to be still part of the particle set and the low weighted particles are more unlikely to be part of the set. This is also known as importance sampling. After the re-sampling, all particles have the same weights and approximately distributed to $bel(xt)$ from equation 2.16 [18]. The re-sampling can be done, using different techniques. Two common techniques are described in the following.

C.3.0.3 Re-sampling Wheel

The re-sampling wheel is one possible technique for re-sampling and works as follows: At a random index and a random obset will be chosen. Up from the index, the oset will be added. If the oset passes one weight, the index will be increased. This will be done, until M particles are drawn.

C.3.0.4 Low Variance Sampler

The low variance sampler is another technique for re-sampling. The idea of the algorithm is to select the particles based a sequential stochastic process.

APPENDIX D

ADDITIONAL WORK

D.1 How RTAB-Map solve kidnapped Robot problem

One of the more difficult challenges in robotics is the so-called "kidnapped robot problem". Imagine you are blindfolded and taken by car to the home of one of your friends but you don't know which one. When the blindfold is removed, your challenge is to recognize where you are. Chances are you'll be able to determine your location, although you might have to look around a bit to get your bearings. How is it that you are able to recognize a familiar place so easily? It's not hard to imagine that your brain uses visual cues to recognize your surrounding. For example, you might recognize a particular painting on the wall, the sofa in front the TV, or simply the color of the walls. What's more, assuming you have some familiarity with the location, a few glances would generally be enough to conjure up a "metal map" of the entire house. You would then know how to get from one room to another or where the bath room is located.

Over the past few years, Mathieu Labbe from the University of Sherbrooke [12] Quebec has created a remarkable set of algorithms for automated place learning and SLAM that depend on visual cues similar to what might be used by humans and other animals.He also employs a memory management scheme inspired by concepts from the field of Psychology called short term and long term memory. His project is called RTAB-Map for "Real Time Appearance Based Mapping" and the result are very impressive.

In fig. D.1. the upper part is the color image seen through the camera. On the right is the same image where the key visual features are highlighted with overlapping red discs. The visual features used by RTAB-Map can be computed using a number of popular techniques from computer vision including SIFT, SURF, BRIEF, FAST, BRISK, ORB or FREAK. Most of these algorithms look for large changes in intensity in different directions around a point in the image. Notice therefore that there are no red discs centered on the homogeneous parts of the image such

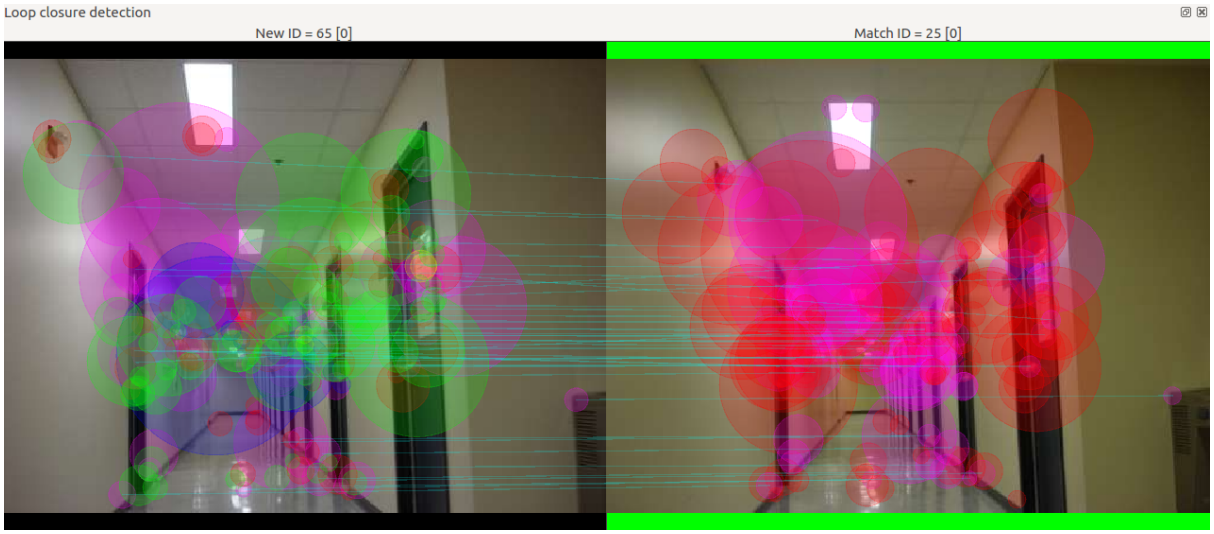


FIGURE D.1. Two images are taken from a typical mapping session using Kinect Xbox360 and RTAB-Map

as the walls, ceiling or floor. Instead, the discs overlap areas where there are abrupt changes in intensity such as the corners of the picture on the far wall. Corner-like features tend to be stable properties of a given location and can be easily detected even under different lighting conditions or when the robot's view is from a different angle or distance from an object.

RTAB-Map records these collections of visual features in memory as the robot roams about the area. At the same time, a machine learning technique known as the "bag of words model" looks for patterns in the features that can then be used to classify the various images as belonging to one location or another. For example, there may be a hundred different video frames like the one shown above but from slightly different viewpoints that all contain visual features similar enough to assign to the same location. The following image shows two such frames side by side:

Here we see two different views from essentially the same location. The pink discs indicate visual features that both images have in common and, as we would expect from these two views, there are quite a few shared features. Based on the number of shared features and their geometric relations to one another, we can determine if the two views should be assigned to the same location or not. In this way, only a subset of the visual features needs to be stored in long term memory while still being able to recognize a location from many different viewpoints. As a result, RTAB-Map can map out large areas such as an entire building or an outdoor campus without requiring an excessive amount of memory storage or processing power to create or use the map.

Note that even though RTAB-Map uses visual features to recognize a location, it is not storing representations of human-defined categories such as PAINTING, TV, SOFA, etc. The features we are discussing here are more like the receptive field responses found in lower levels of the

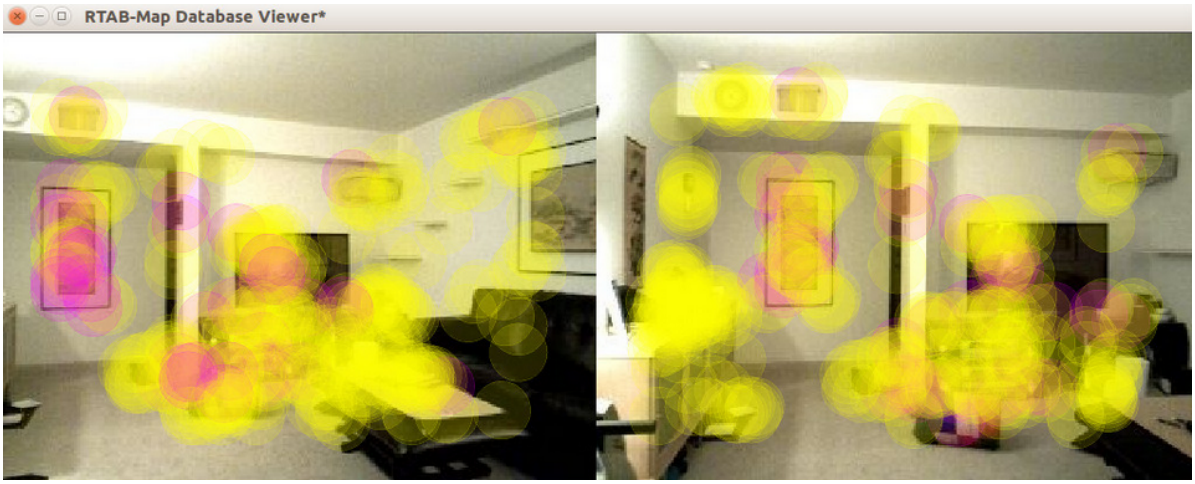


FIGURE D.2. Two different views from essentially the same location

visual cortex in the brain. Nonetheless, when enough of these features have been recorded from a particular view in the past, they can be matched with similar features in a slightly different view as shown above.

D.1.0.1 RGBD SLAM and Octomap

Octomap is another 3D mapping framework available for ROS. Similar to RTAB-Map, Octomap can also be used as a standalone version. Maps are represented by memory efficient Octo-trees where each leaf node represents a cube, or voxel, in the volumetric map. The voxel can be either occupied, free or unexplored. The volume of the cube is determined by how deep in the tree the leaf node is located. In a ROS graph, the Octomapping is performed by the node octomap_server. This node will subscribe to point cloud messages sensor_msgs/PointCloud2, and return volumetric occupancy maps, i.e. Octomaps. There are several approaches to SLAM which uses Octomaps. An example that stands out in the context of ROS is; a SLAM approach which depends on a RGB-D sensor, and relies on Octomap for efficient map storage. This mapping framework was not used in this project in order to limit the project scope, and because the alternative RTAB-Map was associated with less uncertainty.

D.1.0.2 Real Map Automotive Lab in comparison with Hector SLAM map

For experiments a map of the basic scenario is required. Firstly, a map of Automotive-lab and corridor was create by control Volksbot to navigate in the room by teleop_keyboard. The SICK-laser scanner and Hector SLAM ROS_package was use to create map. Map include room 206a, 206b, 203 and the corridor of floor 2, building 8, Frankfurt University of Applied Science. Because in part 4 of figure D3 there is a start and robot can not move to this area, therefore it' miss

on the map. Every other room are mapped by Hector SLAM was 90% same with the real map, the different reason is the object in room like table and computer. The green line in the map is trajectory of the robot, because in some area robot muss can two or three times to get better map resolution. Hector map give good result for a 2D map, but if we need 3D, RTAB-Map should be choose.

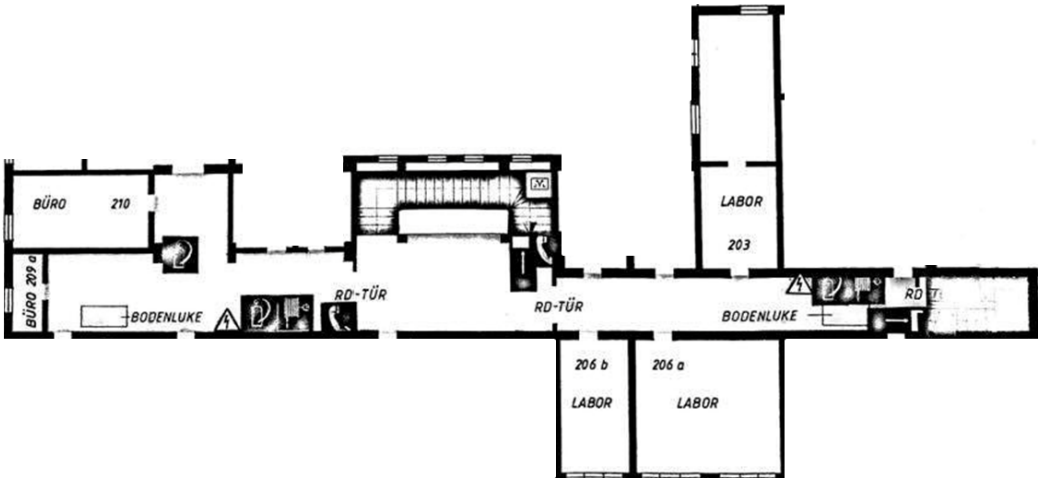


FIGURE D.3. Line map of the building 8 floor take from fire alarm map.

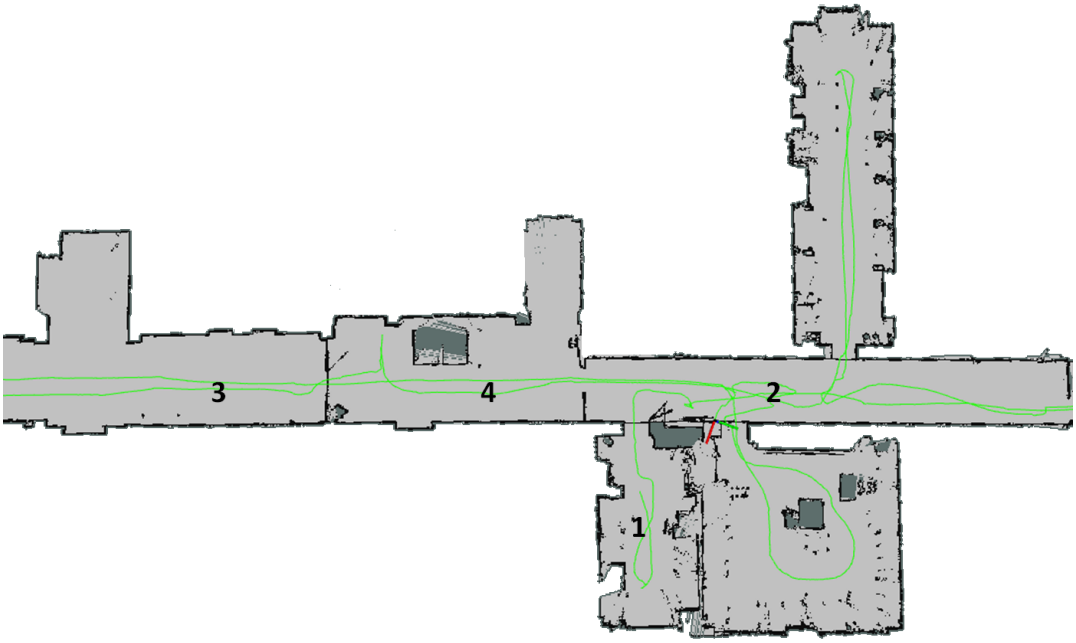


FIGURE D.4. Map create by Volksbot used Hector SLAM.

D.2 Wifi Signal Strength Mapping

Version rtabmap >0.11.14 required. This test will setup a map including information taken with any other sensor. For convenience, we will map Wifi signal strength of the environment. Fig.5.12 shows how the Wifi signal strength is visualized. It is just a level bar used (sensor_msgs/Pointcloud2), but more fancy output like Markers can be used to customize the look.

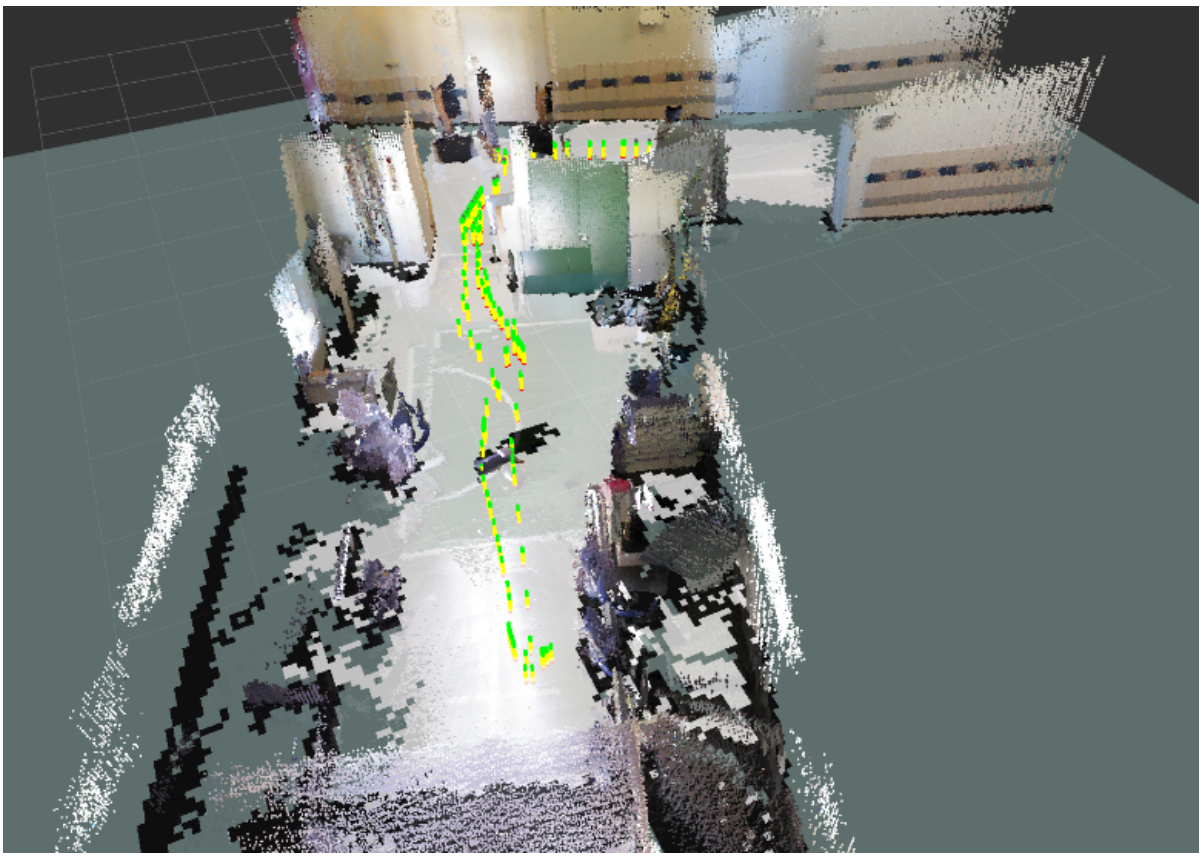


FIGURE D.5. Volksbot RT3 go and draw a map of wifi strength signal

To add user data to map, rtabmap node has topics called user_data and user_data_async. The first one is used when parameter subscribe_user_data:=true, and it will be synchronized with other sensor data. It means that the publishing rate of this topic should be high (at least as fast as the images received). The second topic user_data_async is asynchronous. It should be used only when user data is published slower than the detection rate of rtabmap (default 1 Hz). The received user data will be saved in the next node created in rtabmap (if user data is published faster than nodes are created, old user data will be overwritten)[5].

For this example, we use user_data_async with a Wifi signal data published each 2 seconds. Here is the basic information needed to publish a rtabmap_ros/UserData topic.

How to Retrieve User Data (wifi_signal_sub): rtabmap publishes rtabmap_ros/MapData topic after each update. In this topic, we can get the user data that we published to rtabmap earlier, which is now linked to graph.

Exmaple of usage:

```
thanh@thanh-ThinkPad-W530:~$
$ roslaunch freenect_launch freenect.launch depth_registration:=true
$ roslaunch rtabmap_ros rtabmap.launch rtabmap_args:="--delete_db_on_start"
user_data_async_topic:=/wifi_signal rtabmapviz:=false rviz:=true
$ rosrun rtabmap_ros wifi_signal_pub interface:="wlan0"
$ rosrun rtabmap_ros wifi_signal_sub
```

In RVIZ, add PointCloud2 topic called wifi_signals published from wifi_signal_sub node. Note that you may need to start wifi_signal_pub in sudo. To do so:

```
thanh@thanh-ThinkPad-W530:~$
$ sudo su
$ source /opt/ros/kinetic/setup.bash
$ source /home/{user_name}/catkin_ws/devel/setup.bash (if rtabmap is built from source)
$ rosrun rtabmap_ros wifi_signal_pub interface:="wlan0"
```

Example of Result: You can show the map of the screen-shots above using this database demo_wifi.db (taken with a stereo camera) with the lines below:

```
thanh@thanh-ThinkPad-W530:~$
$ roslaunch rtabmap_ros rtabmap.launch database_path:=~/Downloads/demo_wifi.db
rtabmapviz:=false rviz:=true
$ rosrun rtabmap_ros wifi_signal_sub
// In RVIZ, add `PointCloud2` topic called `wifi_signals`
```

D.3 DVD Contents

- Master Thesis (documentation)
- Project files: Robot (ROS)
- Project files: Installation Guide
- Project files: User Guide
- Videos:
  - live_mapping
  - live_navigation
  - sim_mapping
  - sim_navigation
- Images

BIBLIOGRAPHY

[1] M. J. M. A. J. GLOVER, W. P. MADDERN AND G. F. WYETH, *Fabmap + ratslam: Appearance-based slam for multiple times of day*, Proceedings of the IEEE International Conference on Robotics and Automation„ vol. 24, no. 5, pp. 1107-1120, 2008., pp. 139–151.

[2] M. CUMMIMS AND P. NEWMAN, *FAB-MAP: probabilistic localization and mapping in the space of appearance,*, The Int. J. of Robotics Research, 2008.

[3] M. CUMMINS AND P. NEWMAN, *Accelerated appearance-only slam*, Proceedings of the IEEE International Conference on Robotics and Automation, vol. 24, no. 5, pp. 1107-1120, 2008., p. 140151.

[4] T. F. B. G. EITAN MARDER-EPPSTEIN, ERIC BERGER AND K. KONOLIGE, *The office marathon: Robust navigation in an indoor office environment*, International Conference on Robotics and Automation, 13 (2010), pp. 139–151.

[5] F.MICHAUD, *Appearance-based loop closure detection for online large-scale and long-term operation*, IEEE Transactions on Robotics, vol. 29, no. 3, pp. 734-745, 2013. (IEEE Xplore), vol. 110, no.3,pp.346-359 (10-02-2016).

[6] W. GARAGE, *Ros-industrial supported hardware*, http://wiki.ros.org/Industrial/supported-hardware, vol. 110, no.3,pp.346-359 (2009).

[7] T. T. H. BAY, A. ESS AND L. V. GOOL, *Speeded up robust features (surf)*, Computer Vision and Image Understanding, vol. 110, no.3,pp.346-359 (2009), pp. 139–151.

[8] A. KOUBAA, *Robot operating system (ros)*, The Complete Reference, volume 1. (Springer, 2016.).

[9] J. D. T. L. M. PAZ AND J. NEIRA, *Divide and conquer: Ekf slam in o*, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems„ vol. 24, no. 5, pp. 1107-1120, 2008.

[10] D. G. LOWE, *Instinctive image features from scale-invariant key-points,*, Int. J. of Computer Vision „ 60 (January 5, 2004), pp. 91–110.

[11] M.LABBE, *On-line global loop closure detection for large-scale multi-session graph-based slam*, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2014, vol. 110, no.3,pp.346-359 (10-02-2016).

[12] M.LABBE AND F.MICHAUD, *Memory management for real-time appearance-based loop closure detection*, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems„ vol. 110, no.3,pp.346-359 (10-02-2016), pp. 1139–1521.

[13] M. MUJA AND D. G. LOWE, *Fast approximate nearest neighbors with automatic algorithm configuration*, Proceedings of the International Conference on Computer Vision Theory and Application, vol. 24, no. 5, pp., 2008., pp. 139–151.

[14] E. H. X. R. D. F. PETER HENRY, MICHAEL KRAININ, *Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments*, https://rse-lab.cs.washington.edu/papers, (April 2009).

[15] ROS-INDUSTRIAL, *http:/ /rosindustrial.org /the-challenge /*, The Plant Cell, 13 (2016-04-05), pp. 139–151.

[16] ROS-QUESTION, *Kinect for windows does in work with ros?*, http://answers.ros.org/question/12876/kinect-for-windows/, vol. 110, no.3,pp.346-359 (10-02-2016).

[17] D. F. SEBASTIAN THRUN, WOLFRAM BURGARD, *Probabilistic robotics*, Book, volume 1, 7, 12, 14 (1999-2000).

[18] W. B. SEBASTIAN THRUN AND D. FOX, *Probabilistic robotics*, MIT Press, Cambridge and Mass, 2005, vol. 110, no.3,pp.346-359 (10-02-2016).

[19] J. SIVIC, *Efficient visual search of videos cast as text retrieval*, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 31, NO. 4. IEEE. pp. 591-605 (April 2009).

[20] T. G. K. P. O. V. S. STEFAN KOHLBRECHER, JOHANNES MEYER AND U. KLINGAUF, *Hector open source modules for autonomous mapping and navigation with rescue robots*, Department of Computer Science, TU Darmstadt, Germany. http://www.gkmm.tu-darmstadt.de/rescue, vol. 24, no. 5, pp. 907-1420, 2008 (2008, p. 8.), pp. 139–151.

[21] P. WEBB AND J. ASHLEY, *Beginning kinect programming with the microsoft kinect sdk*.

[22] WIKI.ROS.ORG/HECTOR_MAPPING, *Tutorial: Using the hector slam*, Proceedings of the Australasian Conference on Robotics and Automation, vol. 24, no. 5, pp. 1107-1120, 2008 (2008, p. 8.).