NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

PROGRAM OF POSTGRADUATE STUDIES

MASTER'S THESIS

Integration of ROS, Turtlebot, RPLIDAR, RFID technologies and algorithm implementation for navigation and RFID tag detection in a warehouse.

Paris A. Ioakeimidis

Supervisor **Hadjiefthymiades Stathes,** Associate Professor

ATHENS

SEPTEMBER 2018

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ενσωμάτωση ROS, Turtlebot, RPLIDAR, RFID τεχνολογιών και υλοποίηση αλγορίθμου για την πλοήγηση και ανίχνευση ετικετών RFID σε αποθήκη.

Πάρης Α. Ιωακειμίδης

Επιβλέπων: **Ευστάθιος Χατζηευθυμιάδης,** Αναπληρωτής Καθηγητής

ΑΘΗΝΑ

ΣΕΠΤΈΜΒΡΙΟΣ 2018

MASTER'S THESIS

Integration of ROS, Turtlebot, RPLIDAR, RFID technologies and algorithm implementation for navigation and RFID tag detection in a warehouse.

Paris A. Ioakeimidis

A.M.: M1397

Supervisor: **Hadjiefthymiades Stathes,** Associate Professor

September 2018

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ενσωμάτωση ROS, Turtlebot, RPLIDAR, RFID τεχνολογιών και υλοποίηση αλγορίθμου για την πλοήγηση και ανίχνευση ετικετών RFID σε αποθήκη.

Πάρης Α. Ιωακειμίδης

Α.Μ.: Μ1397

ΕΠΙΒΛΕΠΩΝ: **Ευστάθιος Χατζηευθυμιάδης,** Αναπληρωτής Καθηγητής

Σεπτέμβριος 2018

ABSTRACT

During recent years, we have seen a rapid development in robotic automatic systems which without the human intervention, operate in different environments and perform specific tasks according to their specifications and the missions assigned to them. This thesis attempts to use such a robotic system, namely Turtlebot 2 which uses ROS, in combination with two other technologies. The two technologies used and integrated are LIDAR which is a laser pulse emission technology for mapping the environment which is surrounding the robot and RFID technology for radio frequency identification to identify different objects around the robot. Furthermore an implemented algorithm in Python alongside ROS is presented for assigning a mission to the Turtlebot 2 robot for the autonomous navigation indoors. During the Turtlebot's 2 robot path, data is collected using RFID antennas from the RFID tags which represent the objects on the shelves of a warehouse. Finally the collected data can either be watched live during the collection procedure or stored for further future processing.

SUBJECT AREA: Robotics

KEYWORDS: Robotics, Turtlebot, Navigation, ROS, LIDAR, RFID Technology, Warehouse.

ΠΕΡΙΛΗΨΗ

Τα τελευταία χρόνια έχει παρατηρηθεί ραγδαία εξέλιξη στα ρομποτικά αυτόματα συστήματα, τα οποία χωρίς την παρεμβολή του ανθρωπίνου παράγοντα ενεργούν σε διάφορα περιβάλλοντα και εκτελούν συγκριμένα καθήκοντα ανάλογα με τις προδιαγραφές και τις αποστολές που τους έχουν ανατεθεί. Σε αυτήν την διπλωματική εργασία γίνεται προσπάθεια χρήσης ενός τέτοιου ρομποτικού συστήματος και συγκεκριμένα του Turtlebot 2 που χρησιμοποιεί το ROS σε συνδυασμό με ακόμα δύο τεχνολογίες. Οι τεχνολογίες αυτές που χρησιμοποιούνται και ενσωματώνονται στο ρομποτικό σύστημα είναι η τεχνική LIDAR εκπομπής παλμικής ακτινοβολίας λέιζερ για την χαρτογράφηση του περιβάλλοντος που περιβάλλει το ρομπότ και η τεχνολογία RFID για την ταυτοποίηση μέσω ραδιοσυχνοτήτων και για την αναγνώριση διαφόρων αντικειμένων που βρίσκονται στο κοντινό περιβάλλον όπου δρα το ρομπότ. Στην συνέχεια παρουσιάζεται ένας αλγόριθμος σε Python και ROS που έχει υλοποιηθεί για την εκτέλεση της αποστολής που έχει ανατεθεί στο Turtlebot 2 ρομπότ για την αυτόνομη πλοήγηση του σε εσωτερικό χώρο. Κατά την διάρκεια της διαδρομής που ακολουθεί το Turtlebot 2 ρομπότ συλλέγονται δεδομένα με την βοήθεια των RFID κεραιών από τις RFID ετικέτες που αντιπροσωπεύουν τα αντικείμενα στα ράφια μιας αποθήκης. Τέλος, τα δεδομένα που συλλέγονται μπορούν είτε να παρακολουθούνται ζωντανά κατά την διάρκεια της συλλογής τους είτε να αποθηκεύονται για περαιτέρω επεξεργασία τους στο μέλλον.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Ρομποτική

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Ρομποτική, Turtlebot, ROS, Πλοήγηση, LIDAR, RFID Technology, Warehouse.

DEDICATIONS

Dedicated to my family, my friends and all who supported me all these months during my master studies at the National and Kapodistrian University of Athens at the Department of Informatics and Telecommunication.

THANKS

I would like to express my particular gratitude to supervising professor Stathes Hadjiefthymiades and his whole team at his lab at Department of Informatics and Telecommunication, at the National and Kapodistrian University of Athens, who supported, advised and tolerated me during implementation and writing of this master thesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PROLOGUE – FOREWORD

This thesis as already described in the abstract is introducing an automated system created from separate technological parts for navigation and identification of objects. The challenges that this master thesis tries to solve belong to the field of robotics, automation, IoT and AI. The final solution is a robotic system consisted from a robotic system hardware/software, sensors, and technologies for scanning and collecting the desired data. This system is meant to be used in a warehouse environment where it will enable the robot to navigate and move in a predefined manner with the help of a implemented algorithm and other technologies and detect goods lying on the shelves.

1. INTRODUCTION

As technology is developing robots are becoming part of our everyday life especially in the industrial world. The last years many robotic systems are developed for performing different tasks autonomously without human intervention in different environments and sectors. Systems based on IoT – Internet of Things, AI – Artificial Intelligence and automation solutions are popping up more and more to solve everyday and industrial problems as technology in this fields evolve. Robot come in different sizes, and shapes they can vary from very huge robots to very tiny operating on human bodies. Robots cannot act autonomously without sensors which provide them with data from the external environment and give the ability to navigate and perform tasks and goals set by humans. Specifically one of the most well known systems uses is open-source ROS - Robotic Operating System a flexible framework for developing software for robots and Turtlebot 2 an collection of open-source software personal robot which get benefited from ROS.

Figure 1: Turtlebot 2
https://www.turtlebot.com/turtlebot2/

For the navigation of the robot a sensor for the mapping of the surrounding is needed. The technology used in this master thesis is called LIDAR which stands for Light Detection And Ranging and specifically a sensor called RPLIDAR A2. LIDARs with the help of a laser measure the reflected pulses with the help of the sensor which first produce a precise map representation of the its surrounding and finally acts as the eyes of the robot for the navigation and preventing it from colliding with the objects. LIDARs today are used in many aspects from mapping and creating a 3D image of an area from an airplane to autonomously driving cars for constructing a map and avoiding unexpected obstacles.

Figure 2: RPLIDAR A2

For the scanning of the object located around the robot and specifically around Turtlebot 2 another technology is integrated on top of Turtlebot 2 platform. This technology is called RFID and it is well known for a long time now but the last year it has become part of our everyday life and affect it positively. In this work RFID technology is used for the detection of the objects around our robot Turtlebot 2. The environment where Turtlebot acts could be a warehouse which hosts thousands of objects/products with RFID tags on top of each with information about the each object, next the RFID antennas located on top of Turtlebot scan the RFID tags and locate the objects recording the position of where the object was located and the RFID tag of the object.

Figure 3: Kathrein Wide Range 70 degrees Antennas https://www.kathrein-solutions.com/products/hardware/rfid-antennas/wide-range-70-antennas

Figure 4: MTI RFID RF Module

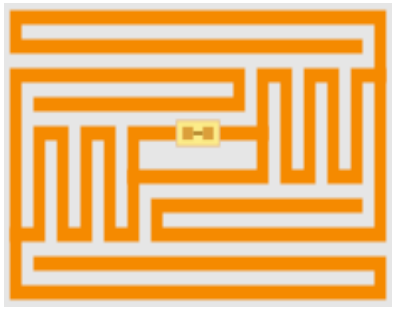

Figure 5: RFID tag inner view, source: https://simple.wikipedia.org/wiki/File:EPC-RFID-TAG.svg

As described previously in this master thesis this three technologies described previously alongside with other technologies are integrated in a such way to create a final solution for collecting data and locate the object/products hosted in a warehouse automatically with the help o a robot called Turtlebot 2, lidar called RPLIDAR, RFID Antennas and RFID Tags.

1.1. Current Situation

Nowdays there are many examples where robots are used in the warehouse sector one example is Amazon that uses more than 80 thousands robots in its distribution centers to

perform different tasks like moving goods inside of their warehouses. Also at Alibaba's smart warehouse robots perform 70 percent of the work and carry huge amounts of goods up to five hundred kilograms on top of them moving around the Alibaba's warehouse. The reason this robot systems where introduced to warehouses was mostly due to e-commerce continues grow, these way the speed and the accuracy of the services provided was increased. These way better efficiency and accuracy was succeeded. Many problems could be managed successfully with the help of such robotic systems which offer automated solutions.

1.2. Why this work is important

This work is not only important because it solves the set problem but because it brings different technologies together to solve the desired problem. And to give ideas for future industrial solutions.

1.3. Questions answered in this master thesis

This master thesis tries to answer multiple question that have a great impact on how future warehouse could be organized and generally how automation, IoT – Internet of Things and other technologies can all come together to constitute if not a complete solution, for sure a very attractive proposal for a possible complete solution. Questions like the following are answered:

(a) What could be a solution for a automatic goods or objects detection and registration system in a warehouse?

(b) What technologies are available today to solve this problem?

(c) How to combine all the different technologies and integrate them together to get to the final result?

(d) How each separate technology is important for solving the set problem? Etc.

1.4. Hypothesis

We make a hypothesis that the proposed, combined and used technologies will all work as a complete solution in this project.

1.5. Methodology algorithm implemented for the navigation and RFID data collection.

There are several parts of this project where special algorithm and software are built. Probably the most important is the navigation in the warehouse of the Turtlebot. The general idea of the proposed algorithm and implemented program is done with the help of a special framework and collection of tools so an algorithm is implemented for setting goals and a passing a mission to our robot for the navigation inside of a warehouse in a

predefined route. Also several structures are implemented for the collection of the RFID tags with the help of an another algorithm for RFID collection.

2. PROBLEM DESCRIPTION

Lets assume that we all agree that robots are important to us but the use of the robots came at a cost. Many problems arise since robotic systems need configurations, adjustment, development of the necessary algorithms, integration of sensors on top of the robotic system, study of the surrounding environment, setting the right goal to the robot for proper mission completion and much more.

In this master thesis we are looking for a solution to the problem of use of a robot in a warehouse for detection of objects which lie on the shelves. This robot should have the ability to navigate between the corridors of the warehouse and detect the objects that it meets on the predefined path.

The first thing we are called to do is to choose an appropriate robotic system to rely on for the implementation of this master thesis. The chosen robotic system has to be set up and tunned properly to be fully functional. The robotic system and robot selected in this work are ROS – Robotic Operating System and Turtlebot 2 robot which are presented in the next chapters more precisely.

Lets suppose that we are in a warehouse and there are thousands of objects on the shelves. This objects have to be tracked and recorded efficiently. Until this moment this was done by humans but this could soon change, this task can be perform much faster and efficiently by a robot. The usage of robots for this task create several major problems which we are called to solve and will be described in the next paragraphs. The main problem are robot navigation, mission/goal assignment, and object detection.

First major problem appears is the autonomous navigation of the robot and the the goal assignment. This will provide the robot with the ability of moving though the warehouse routes autonomously and without human interaction, after the goal assignment and until its completion. A robot has to navigate in warehouse corridors and avoid collisions with obstacles which could potential harm both the robot and the obstacles that appear on its way. A technology has to be integrated on top of the robot for surrounding area mapping, scanning and navigating. For the navigation in the warehouse Turtlebot 2 robot is used on top of which RPLIDAR A2 was used.

Although robots can be programmed and operated easily from an experienced user who has all the necessary knowledge about a specific robotic system, it is nearly impossible for a non experienced user to program a robot without any previous knowledge and experience. So it is important to introduce or locate a user friendly approach of providing missions and goals to the robot even if the this user is coming from a complete different scientific field. An algorithm has to be implemented for the mission/goal assignment alongside with a technology like DSL - Domain Specific Language or more precisely EDL – Experiment Descriptive Language could be used.

Another problem arose is the ability of the robot to scan and record the objected located on the shelves of the warehouse. A technology has to be integrated on top of the robot and provide the robot with the ability of object detection. Robot has to move in the predefined path and scan for the objects located around. After the data collected it has to be stored and forwarded and presented to the user. Luckily there are many technologies that can help us accomplish this task one of them is a technology widely used and known called RFID or Radio Frequency Identification.

In the next two chapters previous work is presented and the solutions proposed in this master thesis to settle down the presented problem.

3. PREVIOUS WORK

In this chapter of the master thesis are presented previous works already done according both the general subject of this work and the problems described in previous chapter.

3.1. Work 1: EDL

[1]"Kostas Kolomvatsos, Michael Tsiroukis, Stathes Hadjiefthymiades. 2017. An Experiment Description Language for Supporting Mobile Iot Applications. National And Kapodistrian University of Athens. Department of Informatics and Telecomunications"[1]

First is presented a work done by a team in National Kapodistrian University of Athens in Computer Science Department. The Title of this work is "An Experiment Description Language for Supporting Mobile Iot Applications", by Kostas Kolomvatsos, Michael Tsiroukis, Stathes Hadjiefthymiades. This work is presented because it concerns the issue of remote experimentation with a Domain Specific Language called EDL – Experiment Descriptive Language together with a developed platform for experimentation which gives an easy and accessible approach for scientists from any scientific domain.

In this work authors cover the subject of management remote experimentation in the field of mobile IoT for experimenters with or without the knowledge of the underlying technologies. They present a collation of different innovative technologies they developed. They call their proposal RAWFIE from the following words: Road-, Air- and Water-based Future Internet Experimentation. The main things they present in this paper are the experimentation Domain Specific language (DSL) they developed and called Experiment Description Language (EDL), web editors they developed and other functionalities available in their innovative creation.

RAWFIE platform offers remote experimentation functionalities to the researchers and professionals. RAWFIE contains a framework for interconnecting numerous test-beds over which remote experimentation on top of real devices will be realized. The devices manufacturers and characteristics can be totally different from device to device.

Since experimenters(not experienced) have no knowledge about the characteristics of the devices RAWFIE covers this gap by offering an abstraction of the underlying functionalities by developing a Domain Specific Language (DSL) of their own. According the paper this will offer the opportunity to non-experienced users to write more easily domain specific programs not dependent on the underlying platform. Their DSL is called Experiment Description Language (EDL) which can be edited in visual and textual editors with assistance. Code generation component automatically create information transferred to mobile nodes. RAWFIE efficiently interconnect experimenters coming from various domains with the nodes present in numerous test-beds.

1 http://eprints.gla.ac.uk/163502/

The problem authors are dealing with the lack of knowledge of experimenters for the low level commands to handle the heterogeneity of the devices. Researchers who perform testbed to handle and manage different devices they don't have the knowledge to use all these technologies and commands for the heterogeneous devices. To overcome this problem they want to create an abstract way for defining commands for UVs that will give the chance to experimenters of any field to handle the devices in their experiments. They want to use DSLs which follow the principles of Model Driven Engineering (MDE) development. RAWFIE offers EDL which provides a terminology for defining experiments for mobile IoT.

The purpose of the RAWFIE initiative is to create a federation of different test-beds that will be combined to make their resources available under a common framework. RAWFIE will integrate numerous testbeds for experimenting in vehicular(road), aerial and maritime environments. These UVs will be expose to the experimenters a vast test infrastructure. The vision of Experimentation-as-a-Service (EaaS) is promoted though RAWFIE.

RAWFIE architecture consists of tree tier design patterns. Each tier is separated in different software elements, each one providing a different functionality. i) the front-end tier (includes : The RAWFIE Web portal, The Testbed and Resource Discovery, and The Experimentation Suite which include five tools), ii)the middle tier(includes: the Experiment Validator, the Experiment Controller, the Visualization Engine, the Testbed directory, the Data Collection and Analysis module, the Launching Service, the Booking Service, and the System Monitoring Service.) and iii)the data tier(insures the data persistence, stores everything in Data Storage and Code Repositories and serves the Cloud).

The main component is RAWFIE EDL – Experiment Description Language. EDL is used to create simple and complex experimental scenarios for the IoT domain. By providing high level of abstraction it shields the experimenters from the complexities of the underlying implementation of the RAWFIE platform and the available devices. EDL is simple and similar to XML or legacy programming languages. It is built with Xtext framework. Main parts of EDL are Metadata, Requirements, Declarations and Execution sections. 'Typical' commands originated in legacy programming languages are also included.

Another important components of RAWFIE is EDL Textual Editor and Visual Editor which are synchronized and both are provided as Web application and support functionalities for creation, update, compilation and validation of the experiments. Also editors provide rich editing facilities and advanced content assistance with checking mechanism at syntax time.

The EDL Visual Editor is created for creating experiments in the RAWFIE authoring tool and it provides a user friendly environment that simplifies the creation of an experiment. Both editors are synchronized but the error and warnings appear on the textual editor area.

Next component is the EDL Validator and the Generator. Validator performs the syntactic analysis on the provided EDL scripts on top of the proposed EDL model based on the EDL grammar. Responsibilities of validator are: (i) syntactic and semantic validation, (ii) application of semantic checking for nodes communication, spatio-temporal management, sensing and data management. The code generation component generates files part of which is uploaded in the Uvs.

The EDL grammar and the editors are created by adopting the Xtext framework. In Ecore model produced the structure of EDL's abstract syntax tree (AST) is described. The Xtext framework offers a set of automatic validation functionalities. Parser to the first validation step and the linker heck for broken cross-references between EDL concepts. Also other validation custom tools are available which are written in Xtend language. This custom validation is for additional constraints for the defined experiments. TH Xtend language is also adopted for the creation of the EDL generator. When it come to the editors the backend Xtent functionalities are invoked with HTTP requests to the server-side component. The text content is either loaded from the Xtext server or provided though Javascript. The Web integration of Xtext supports two operation modes: (i) stateful mode and (ii) stateless mode. The client side of both editors is built with Javascript and the map of the client side is OpenLayers.

Important part of the paper is the demonstration of case study where the authors show how RAWFIE platform could be used. They show all the parts presented in the paper in practice. And by presenting the platform with pictures they explain all the different parts of RAWFIE.

3.2. Work 2: E-Pres

[2]"Michail Chatzidakis, Michail Loukeris, Kostis Gerakos, Stathes Hadjiefthymiades. 2016. E-Pres: Monitoring and Evaluation of Natural Hazard Preparedness At School Community. Pervasive Computing Research Group. National And Kapodistrian University of Athens. Department of Informatics and Telecomunications"[2]

Next is presented a work which is also very important and could give a lot of potential knowledge to this master thesis which is called "E-Pres: Monitoring and Evaluation of Natural Hazard Preparedness At School Community", by Michail Chatzidakis, Michail Loukeris, Kostis Gerakos, Stathes Hadjiefthymiades. This work was done by the Pervasive Computing Research Group at Department of Informatics and Telecommunications of National and Kapodistrian University of Athens. One of the main parts of this work is a system implemented with RFID Antennas and Tags for performing evacuation drills at school community.

Authors in this paper present a unique project which implements a system for performing evacuation drills their main target is school community. They use sensor network, and they implement a front and back-end systems. They obtain live stream of the evacuation

2 https://ieeexplore.ieee.org/document/7857218/authors

procedure and they store in database and make prepossessing of the data with relevant statistical tools. The main component their system is bases is RFIDS and RF antennas. Authors describe their system as innovative, robust, secure, real-time, salable, and fault-tolerant through the paper.

The main purpose of the E-PreS project is to prevent the natural hazards, the main goal is to design and evaluate the drills and exercises performed to help the staff and students to be prepared to react appropriately. E-PreS can be used inn any situation where location analytics is needed, to highlight congested junction points and perform bottleneck detection.

In the paper they present several other works done in the field of natural hazards. E-PreS fills the gap by providing a trustworthy means to evaluate any evacuation drill and thus any evacuation scenario focusing on earthquake, food, volcanic eruption and also focusing to schools.

Authors have developed a methodology for the real-time evaluation of prevention measures involving different categories of actors, districts, steps and metrics. To the drills or training new steps or individuals could be added if there are available spatial and sensor information.

The E-PreS system consists of a number of checkpoints on each of them RFID antennas are set up for monitoring connected to RFID readers connected to a board computer. Each person of the drill has an RFID tag on their shoes for better signal. Data is transferred though WIFI from board computers to back-end server which runs web services and databases. Through this web services user can perform different types of functionalities (Building insertion, Upload floor blueprints, Define acceptable evacuation metrics, Fill questionnaires).

The architecture of E-PreS is consisted of the following five parts : Sensor network, Data stream components, Data processing components, Service components, and Web application components.

Sensor network is the backbone of E-PreS system. Sensors gather information and transmit it though the internal network of the mobile server for processing. They want the sensors to follow the following guidelines: to be uniquely identifiable, available to the server on demand, and readings and messages to be stored and retrieved. To create this capabilities they use embedded systems of sensors with software implemented by the authors to exploit the capabilities aforementioned. Sensor network is consisted of 1)Reader interface: which is responsible for data collection and transmission) and 2)Sensor Message Queue: which is each sensor’s internal message queue which acts as unique message storage. Responsibilities of sensor message queue are: Maintaining the messages, Keeping the data safe, and Enabling fast and reliable message delivery.

Data stream components is the group between the level of sensors and data processing components. These components are responsible for handling the data flow between the to levels by distributing data in parallel with assurance for no data to be lost. Data stream components group is consisted of 1)Server Message Queue – It is implemented on the server and receives data from sensor units Message Queues (Receives, Maintains the messages, and Keep safe the data), and 2)Message Broker – It is the software component that handles the message queue and delivers a feed of messages to different types of receivers (Handling the messages, Maintaining feed, Providing interfaces, Delivering the messages). The Message Broker is in charge of transmitting the stream of data to other layers.

Service components provide functionality for the web application component. Their responsibilities are data analysis, registration services etc. They run in web applications. 1)Real Time Service: provides data analysis to the web application controller. 2)Data Analysis Service: Data analysis on off-line data. 3)Registration Service: It's responsibility is the registration of procedures. 4)Security layer: Security functionality to any component in the system.

Web Application components: provides interaction to the user with the E-PreS system though the web application. It is consisted of the following three components: Drill Registration/Modification, Drill Review, Real Time Monitoring.

Data Processing components processes the data produced bu the sensor component, run analytical tests and present the results to the service components. It follows a modified lambda architecture(scalability, and fault-tolerance). Lambda architecture defines three layers: A)Batch layer: stores the data to a database for persistence and consistency, it can handle and store efficiently massive amounts of data, B)Serving layer: It is important for loading the batch views, and automatically updates the outdated views, C)Speed layer: It is processing the data received and creates views and stores them in specialized memory module for compensating the latency of the Batch layer – It is a real time storage and analytical component. Speed layer utilities storm server and bolt architecture to achieve real-time views of the ongoing drill.

3.3. Work 3: Robotic Navigation

[3]"Kshitija Deshmukh, Ashitha Ann Santhosh, Yogesh Mane, Saurabh Verma, Sdhana Pai. Nov 2015. Robotic navigation and inventory management in warehouses. International Journal of Soft Computing and Artificial Intelligence. ISSN. 3(2): 75-79"[3]

In this paper they discuss the introduction of robots for navigating and moving cartons around the warehouse and an approach for automated inventory management procedure. They propose an automated system composed of a robot, RFID technology and other parts for replacing humans. They present briefly several already introduced approaches

3 http://www.iraj.in/journal/journal_file/journal_pdf/4-204-145127922775-79.pdf

like Amazon's and Swisslog warehouse robots. In their approach they use a robot which will navigate in a warehouse by using a line follower technique with IR sensor. For the identification and the inventory management they choose passive RFID tags with reader alongside with an Arduino controller and a wireless module. Finally the loading and the unloading should be performed with combination servo motors or a forklift. Finally they mention a central unit for the control of the overall system.

The system presented in this paper looks very attractive but it has several flows like luck of experiments. It is not completely obvious how several parts of their proposed system function. Also there is not enough comparison or reference of similar technologies that could solve the same technological challenges. Finally the distance of RFID reader is to short only 6 cm. The overall idea of this paper is very interesting and useful.

3.4. Work 4 Navigation Tuning

[4]"Kaiyu Zheng. Sep 2016, ROS Navigation Tuning Guide"[4]

In this work are described the main parts of the ROS navigation stack, this work can be used as a reference and it guides the user to perform tuning navigation parameters of ROS. Most importan navigation stack parameters are presented and discussed in this work. At the end of this work the main problems of the navigation stack are presented.

3.5. Work 5: RFID Technology

With a fast Google search for keywords “RFID usage in warehouses papers” someone can find tons of papers describing RFID Technology being used in warehouses. This proofs that there are many proposed solutions for RFID technology to be applied in warehouses. RFID technology is applied in warehouses to improve the logistics part. This works were not studied in the context of this thesis but are mentioned as a proof that RFID technology is studied for future use in warehouses. Mostly these works and papers are presenting RFID as a technology for bringing optimization to the warehouse logistics chain through tracking, identifying, and detecting objects, such works are: (a)*Wamba and Chatfield(2010)*[5], (b)Hassan et al.*(2012)*[6], (c)*Rodrigues(2009)*[7], (d)*Pacciarelli et al.(2011)*[8],

4 http://kaiyuzheng.me/documents/papers/ros_navguide.pdf

5 https://ro.uow.edu.au/infopapers/1827/

6 https://ieeexplore.ieee.org/document/6468853/

7 https://www.semanticscholar.org/paper/1-of-10-INTEGRATION-OF-RFID-TECHNOLOGY-IN-A-CHAIN-Rodrigues/9882462c9beaf14f50f41ec3f5e6e84dcbc0a9f1

8 https://link.springer.com/article/10.1007/s11066-011-9059-4

(e)*Yan et al.(2008)*[9], and (f)*Wang et al.(2015)*[10]. In contrast to this papers we present RFID technology in combination with several other technologies in a more practical form. Others present RFID technology for indoor location sensing which is a completely different approach not being investigated in this work.

9 https://ieeexplore.ieee.org/document/4609858/authors

10 http://www.ijmmm.org/index.php?m=content&c=index&a=show&catid=40&id=258

4. PROPOSED SOLUTION

The main choice performed in this thesis is the selection of the robot for the implementation of this work. The robot chosen for this master thesis was an accessible robot called Turtlebot. Turtlebot is an open-source affordable low cost personal robot for experimentation and implementation of different applications. Turtlebot is a kit that provides all the basic necessary parts to start a robotic project. Turtlebot is a kit that includes a mobile base, a 3D Kinect Sensor and a netbook alongside with the Turtlebot hardware mounting kit and ROS SDK accessible publicly on-line. Turtlebot gives the opportunity to mount hardware and sensors on top of it. Specifically Turtlebot model 2 was chosen for the implementation of this master thesis. Turtlebot is a good choice not only because it is open source an low-cost but also because it contains Turtlebot SDK and it it runs ROS – Robotic Operating System, which gives a lot of functionalities and capabilities to the robot directly out of the box similar to larger robotics platforms. The main functionality we are interested in this master thesis is navigating in interior and mounting extra hardware on top of the robot. In the next chapters Turtlebot is explained and presented more precisely.

For the navigation, mapping, obstacle avoidance and RPLIDAR model A2 is being used. RPLIDAR is a 360 degree laser scanner and it is intended for both indoor and outdoor application. Its characteristics are much superior than Kinect for indoor navigation this is the reason why a choice was made to use it as a replacement to Turtlebot default Kinect 3D laser. RPLIDAR can be physically mounted on top of Turtlebot and scan 360 degrees the surrounding area. The fact that RPLIDAR spins 360 degrees(with all the other superior technical characterists) compared to the small scanning angle of Kinect which scans only the area in front of the our robots is another reason RPLIDAR preferred as a mapping and navigation sensor. Finally LIDARs is the solution chosen and mounted on top of the autonomous vehicles and self driving cars for moving autonomously and avoiding obstacles.

For the scanning and detection of the objects on the shelves RFID – Radio-frequency Identification technology is chosen which uses electromagnetic fields. Specifically by saying RFID technology we mean RFID Antennas, passive RFID tags, RFID reader and an algorithm running in the background tuning the proper functionality of the hardware. The RFID reader used is an MTI RU 861-010 reader/module with USB connectivity. By mounting RFID tags on top of each object in warehouse it is possible to locate, track and identify them with the help of RFID antennas and RFID reader both mounted on top of Turtlebot and scanning the area around TurtleBot for RFID tags. The RFID tag technology is preferred to other possible technologies solutions like Barcodes, because in contrast to barcodes, RFID tags don't need to be within the direct sight of the RFID antennas/reader and as positive result can be traced from longer distance. There are also other technologies that could be possible solutions like NFC(RFID like for small distances),

barcode technology, and bluetooth technology for detecting good on the shelves of a warehouse although there are not so

For creating the mission/goal for the Turtlebot to accomplish though the algorithm running on top of Turtlebot EDL – Experiment Descriptive Language is used. This approach was selected because the external user has just to select the points where the Turtlebot should move towards to complete the goal set no further knowledge about internals of ROS – Robotic Operating System or Turtlebot is required.

Furthermore the data collected from the RFID tags is saved both in a structure and a file locally, it is printed as logs and finally published on a topic though a distributed streaming platform called Apache Kafka. As described in the documentation on the official website Apache Kafka has the capabilities of publishing and subscribing to streams of records, it is fault-tolerant durable solution to store streams of records and because it gives as the ability to precess the streams of records that are appearing. With the help of Kafka our data can be reliably transferred between systems and applications and in real-time transform or react on the data that is coming in streams in our case from RFID antennas/reader. Finally a hashing structure is used for filtering the RFID tags data collected. This structure helps us keep only the tags that had the highest signal to each position our robot passed eliminating duplicate tags with lower signal quality.

Last but not least one of the most important parts of this work is the implemented algorithm that proposes the solution for our Turtlebot robot navigation by setting goals. The implemented algorithm get benefited from the ROS and specifically navigation move_base package that is provided by ROS for setting goals to our Turtlebot robot. For the development software in ROS Python, C++ and ROS API are available. For the development of the specific algorithm Python was used but it was partially not fully implemented in C++ to check the outcome of the result comparing to the Python implementation. Generally a lot of ROS packages are written in C++ but in the context of this master thesis Python was chosen.

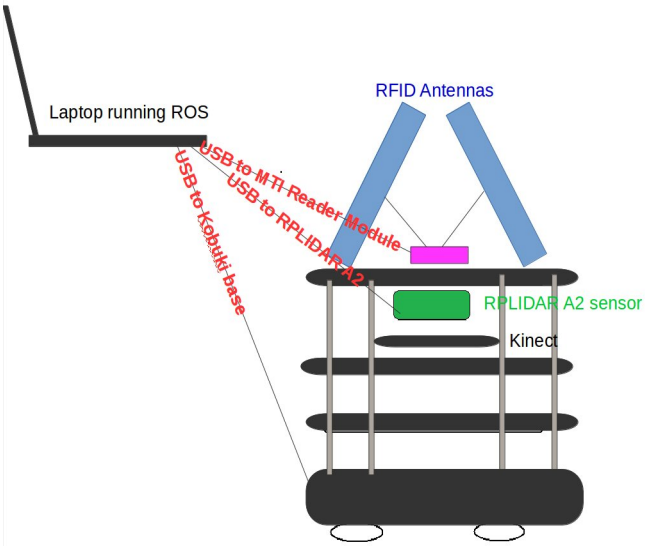

Figure 6: Representation of the Turtlebot with RFID technology, Lidar, and Laptop running ROS.

5. CONCEPTS AND TECHNOLOGIES DESCRIPTION

5.1. Robotics

Term robotics derives from the word robot and stands for a scientific field which study the machines that have the ability to perform different tasks, like replicating human behavior in a more efficient, faster and affordable way or performing task that human never could achieve before. Robotics solve issues concerning all aspects of robots industry several of them are the design, the building, the operation, the control and the use of robots. The technological and the scientific advancements contribute to the field of robotics and robots are introduced to even more sectors of our society. Generally speaking robots doesn't have a specific appearance defining them, but their design choices are adapted according to the requirements of the task they are meant to perform. On the other hand there are robots that are designed to replicate the exact appearance and behavior of humans.

The idea of robotic systems is not something new but it is coming from many centuries ago humans always had a desire to create and engineer automatic machine performing tasks. First references exist about automatic system in the Lie Zi text in Third century B.C. More than hundred machines where described in Pneumatica and Automata by Heron of Alexandria in First century A.D. and earlier. Today Robotic Systems exist everywhere space exploration, in medicine, in agriculture, in industries and factories, in different research areas, in rescue and much more.

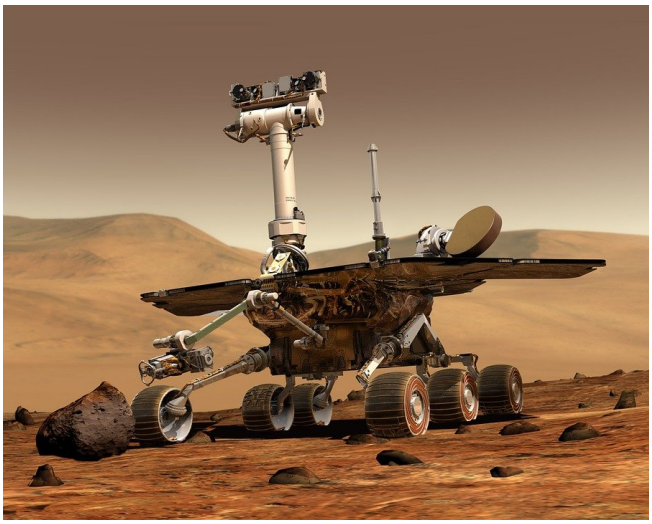

Figure 7: Autonomous vehicle in space, source: https://pixabay.com/en/mars-mars-rover-space-travel-robot-67522/

5.1.1 Robots[11][12]

First reports about creation similar to robots appear in ancient Greek mythology. Several such mythical creation are discussed in literature like the Talos giant automaton self operating machine made of bronze. Talos was Hephaestus creation and was given to king of Crete Minos, its task to protect Europa at the island of Crete by destroying the foreign invaders and moving around the island three times per day.

Generally speaking a robot can be defined as a programmable mechanical device by a computer that can substitute a human being by performing a series of actions automatically. Robots are machines that are programmed in a such way that they can perform different complex task automatically with an internal control device or under the human guidance.

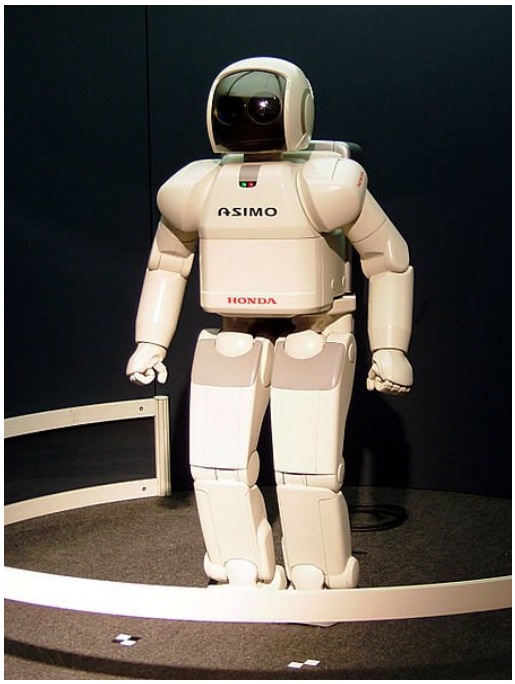

Figure 8: ASIMO robot, source: https://en.wikipedia.org/wiki/File:HONDA_ASIMO.jpg

5.2. Robotic Frameworks

There are many ways and approaches for programming robots out there. In this chapter are briefly presented several approaches about robotic frameworks and generally programming robots. The most well known and accepted is **Robot Operating System ROS** which is the choice used in this work. It was chosen for this work because ROS is

11 More about Robots: https://en.wikipedia.org/wiki/Robotics

12 More about Robots: https://en.wikipedia.org/wiki/Robot

free, ROS runs on top of Turtlebot, it is open source, there are many examples of real robots out there running ROS, it contains all the necessary libraries and tools for robotics development, it contains all the necessary packages already implemented waiting for being used, ROS has an distributed internal approach of software that makes it more stable and hard to fail, any component can be integrated easily into ROS due to it messaging system, sensors can be mounted on top of ROS robots, it is modular. The fact that ROS is open source that makes it accessible, and maintained by many people, this accessibility gives to ROS all the positive feedback it can receive from the accumulated community knowledge. Main source where you can find information about ROS is the official ROS website [13] and the paper presenting ROS [14].

However, ROS is not the only system out there, for many years numerous systems, libraries and frameworks were built and used for programming robots. In the next chapters some other frameworks are presented briefly, the aim of this chapter is just to enumerate several other frameworks and not analyze each one of them. The main frameworks similar to ROS[15]:

1. **Player**(http://playerstage.sourceforge.net/), Player Project is a Free Software tools for robot and sensor applications. It purpose is to create Free Software for research in robot and sensor systems. Player robot server is wildly used. The simulators used in ROS like Stage and Gazebo are part of Player Project.

2. **YARP**(http://www.yarp.it/): YARP stands for Yet Another Robot Platform. YARP is free and open, licensed as LGPL. YARP is ment for building a control system for a robot, organized as a collection of programs communicating in a p2p way alongside with a extensible group of connection types.

3. **Orocos**(http://www.orocos.org/): Orocos stands for Open Robot Control System. Orocos is free and open, licensed as LGPL. It started as an idea for a Free Software project for robot control in December 2000.

4. **Carmen**(http://carmen.sourceforge.net/): Carmen stands for Carnegie Mellon Robot Navigation Toolkit. It is an open-source collection of software meant for the control of mobile robot. It is modular and it provides all the necessary tools for basic navigation.

5. **Orca**(http://orca-robotics.sourceforge.net/orca_doc_overview.html): Orca is a project with software reusability in mind to create progress in the fields of robotic research and industry. It is licensed as LGPL and GPL.

13 http://www.ros.org/

14 http://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf

15 http://wiki.ros.org/ROS/Introduction

6. **MOOS**(http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php/Main/HomePage): MOOS is a Cross Platform Software for Robotics Reasearch.

7. **Microsoft Robotics Studio** (https://www.microsoft.com/en-us/download/details.aspx?id=29081): Microsoft Robotics Studio is Windows-based and .NET-based programming environment freely available for robot control and simulation.. It is intended for building robotics application by both professionals, non-professionals as well as hobbyists.

Tekkotsu[16] is a robotics framework thats name means 'iron bones in Japanese'. The way Tekkotsu is organized it gives you the chance to program in higher levels. It is written in C++ and it provides an API for development. Tekkotsu provides several services out of the box for helping you get started like visual processing, localization, kinematics solvers, remote monitoring and teleoperation tools etc. Finally Tekkotsu is open source and free software that also is built on top of other third party libraries and packages. There are several papers about Tekkotsu [17] and [18].

OpenRDK modular framework is another robotic framework which is open-source as well, on the official sources it is presented as a modular software for rapid development of distribute mobile robotic systems. It is important to mention that OpenRD is written in C++ and it runs on Unix based systems. More information about this framework could be found on the official website of openRDK here [19]. Finally OpenRDK can be tracked back to 2008 and more information about OpenRDK and OpenRDK was described in the following paper [20] and in this paper [21] they present several other robotic frameworks that existed until that moment compared to OpenRDK. Robotic frameworks listed in OpenRDK paper are OROCOS, Orca, CARMEN, OpenRTM-aist, Microsoft Robotics Studio, Player, MOOS, CLARAty, MARIE, MOAST, MIRO, SPQR-RDK.

Last but not least a paper presenting robotic architecture frameworks is [22] they present several robotic frameworks, perform a case study. They speak about different types of frameworks and standards. Several are open source frameworks like ROS – Robotic

16 http://tekkotsu.org/about.html

17 https://ieeexplore.ieee.org/abstract/document/5980533/

18 https://www.ri.cmu.edu/pub_files/pub4/tira_thompson_ethan_2004_1/tira_thompson_ethan_2004_1.pdf

19 http://openrdk.sourceforge.net/

20 https://pdfs.semanticscholar.org/e981/25fdb4947f4ccce67fc61ee363cb61745809.pdf

21 https://www.dis.uniroma1.it/~iocchi/publications/iocchi-iros08.pdf

22 https://www.diva-portal.org/smash/get/diva2:623989/FULLTEXT01.pdf

Operating System and OROCOS mostly written in C++ and run on Windows and Linux OS and some commercial frameworks like MRDS – Microsoft Robotics Development Studio. On the other hand they speak about frameworks which combine Ada, VHDL and FPGA. They also present a standard framework called Jaus Joint Architecture for Unmanned Systems, Open Jaus SDK and MDE – Model Driven Engineering.

ROS is the most convenient solution among all described systems because it is the only one that combines all the necessary tools, packages and solutions for robotics software development and at the same time it is open source and accepted and supported by the robotics community.

5.3. Warehouses

[23] The system studied in this work is meant mostly for warehouse use it could be useful describing the warehouse term. Warehouse is a place (a building) mostly used for commercial purposes where goods are stored. From the ancient times people used different storage technics for storing goods meant for protecting the goods from the natural environment. Early examples of such warehouses were existed from earlier civilization in form of storage pits for protecting seeds and surplus food. During the period of Roman empire existed building called horrea which were public warehouses used during ancient Rome. Later medieval merchants across Europe used to keep goods in household storerooms on ground or underground level. More warehouses existed in Venice and UK. Nowadays warehouses still exist for different types of goods. Warehouses today are modernized and have created a whole system which is meant for effective, secure, convenient storage of goods. A big part of todays economy is affected from how goods are stored and moved from place to place. Also different innovations are introduced to modern warehouses thanks to technological progress like a forkit truck, pallet racks and autonomous robotic systems.

5.4. Robots in Warehouses

Nowadays most of the tasks and operations performed in warehouses are done mostly by humans. Although with the technological progress this obsolete habit is going to change. Already several warehouses introduce revolutionary technologies which introduce faster, easier, safer, more secure and precise execution of operations previously performed by humans. Tasks like loading and unloading goods, moving objects around the warehouse, performing inventory tasks all can be handled by autonomous robotic systems.

In this master thesis we focus on the exploitation of a robots and specifically a robot called Turtlebot in a warehouse environment, its main task is to navigate in a warehouse and perform inventory process. Turtlebot navigates around the warehouse and detect objects placed on the shelves of the warehouse by scanning RFID tags mounted on each object,

23 https://en.wikipedia.org/wiki/Warehouse

today this work is done by humans in most warehouses.

As described previously there are several companies introducing robots to the modern era warehouses with pretty promising results for the future of this industry.

[24]A great example is **Amazon's**, they introduced a huge fleet of robots to their distribution centers, the main task of these robots is to move 24 hours per day a great amount of goods around the warehouse. Amazon robots move whole block of shelves to a person who's task is to pick the product and put it into the cart. This procedure eliminates the need of a person to perform all this steps of searching for the product, scanning, picking and then putting this product into the cart.

[25] [26]**Alibaba** is another company that introduced robots through their smart warehouse robots that perform approximately 70 percent of the work by carrying goods that weigh more than hundred kilograms.

[27]Also Alibaba is performing test on a robot purposed for delivery with a revolutionary technology of pair of eyes.

[28]Furthermore **Ocado** a United Kingdom online grocer they use automatic arm to store and retrieve products. These are only few companies known for using robots in their warehouses, out there many other companies exit that create design and use robots inside of their warehouses.

[29]Last but not least **Swisslog** is a company that designs, develops and delivers automation solutions in domains like health, warehouses and distribution centers. Swisslog together with KUKA have introduced CarryPick, which is an automated storage and order fulfillment system. CarryPick is a storage and picking system which exploit mobile vehicles (KMP600). This robots navigate though a grid to deliver racks with boxes to workstations for picking. This approach reduces the path workers had normally to traverse.

24 https://en.wikipedia.org/wiki/Amazon_Robotics

25 https://www.youtube.com/watch?v=FBl4Y55V2Z4

26 http://uk.businessinsider.com/inside-alibaba-smart-warehouse-robots-70-per-cent-work-technology-logistics-2017-9

27 https://www.technologyreview.com/the-download/611286/alibaba-is-testing-a-delivery-robot-with-a-revolutionary-pair-of-eyes/

28 https://www.youtube.com/watch?v=V5TegyxJY3I & https://www.youtube.com/watch?v=XO7fvrdTCgs

29 https://www.swisslog.com/en-us/warehouse-logistics-distribution-center-automation/products-systems-solutions/asrs-automated-storage-,-a-,-retrieval-systems/boxes-cartons-small-parts-items/carrypick-storage-and-picking-system

Illustration 9: Swisslog Robot, source:
https://www.youtube.com/watch?v=Z-n942tutXY

Generally as it is obvious the main target is to reduce the involvement of humans and introduction of autonomous systems. These technologies eventually will totally integrate in warehouse sector because in contrast with humans they never get tired, they remain operation through the whole day, they minimize the number of mistakes, they have higher productivity and they are more secure. Also multiple tasks that were to dangerous and harmful for humans for example like handling toxic materials now can be handle with zero harm by machines. As technology evolves year by year the warehouses will become even more automated. These changes will transform totally how warehouses and logistics are organized and managed today.

5.5. Sensors

[30]In Wikipedia a sensor is defined as "A ***sensor*** *is a device, module, or subsystem whose purpose is to detect events or changes in its environment and send the information to other electronics, frequently a* computer processor*. A sensor is always used with other electronics, whether as simple as a light or as complex as a computer.*"

Generally speaking a sensor has an input, which it detects data from the environment and an output which redirects the data collected to the next device. This collected data can be modified, processed, studied and used for making a decision from the recipient.

There are thousands of sensors and each is able to collect a specific type of environmental changes or events. There are sensor that measure temperature, light, speed, acceleration, etc... This ability of sensors to get environmental information and provide it to other devices make them useful for multiple applications in medicine, aerospace, agriculture, manufacturing, hazard detection, entertainment, robotics, etc.

Sensors are really important for the robotics field. By collecting the data from the environment, sensors make the robot able to perceive the surrounding world and perform tasks autonomously, make decisions and generally perform work.

30 https://en.wikipedia.org/wiki/Sensor

In this work several sensors are being used for the implementation of the project. The first sensor is measuring a distance of surfaces from with the help of light laser, LIDAR – Light Detection and Ranging, it used for scanning of the surrounding environment, map creation and navigation by the Turtlebot robot, the second sensor which is composed of two RFID antennas are scanning for RFID tags by producing radio frequency waves. Turtlebot hosts more sensors already embedded in it like the sensors for the cliff detection on the left, center and right sides, wheel drop sensor on the left and right side and 3d sensor for navigation.

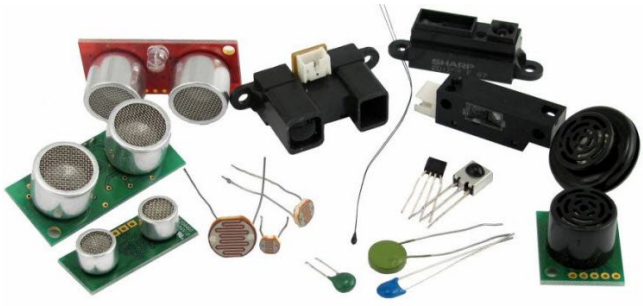

Figure 10: Sensors, source: http://www.efxkits.com/blog/various-types-of-sensors-applications/

5.6. ROS – Robotic Operating System

[5]ROS stands for Robotic Operating System[31] but its name could be misleading because ROS isn't really an operating system it is a framework that provides everything needed for robotics development, and thats why ROS needs a real operating system Linux Ubuntu. ROS is an open-source, meta-operating system that provides a collection of all necessary tools, packages, and services, needed for robotics project implementation. The main standard operating system services provided by ROS are: a) hardware abstraction, b)low-level device control, c)implementation of commonly-used functionality, d) message passing between processes, and finally e) package management.

One of ROS main characteristic is its adaptability, it is accepted and widely used today by researchers and companies. The fact that ROS is free and open source makes it even more attractive. It is accessible for performing experiments for students, amateur, and professionals software developers. Robots running ROS are affordable like Turtlebot, easy to assemble and operate directly out of the box. It is even possible to create and work with ROS robots in a simulated world, with tools like Gazebo and RVIZ coming with ROS installation avoiding the purchasing of a real robot. ROS accommodates a collection of tools, libraries and conventions making really simple the procedure of creation of complicated and robust robot behavior.

Development of ROS started back in 2007 by the Standford Artificial Intelligence

31 http://www.ros.org/

Laboratory (SAIL) to provide support to Stanford AI Robot project. Since 2008 Willow Garage took over and continued ROS development. Finally Open Source Robotics Foundation (OSRF) looks after the maintenance of ROS as well as the project connected to ROS like Gazebo, etc.

ROS makes the development of multiple components of any robotic system accessible and easy. Developers has the chance to share the implemented code and benefit from the large accumulated knowledge from the previous works. The modularity of ROS makes it possible for different parties collaboration for implementation of robotic project by combining their works. Finally already implemented components build up a strong foundation for someone to use in his work.

The main core components provided by ROS are a) Communications infrastructure (message passing, recording and playback of messages, remote procedure calls, distributed parameter system), b) Robot specific features (standard robot messages, root geometry library, robot description language, preemtable remote procedure calls, diagnostics, pose estimation, localization and navigation), and c) Tools (command line tools – rviz – rqt).

Online you can find many ROS powered robots [32]

The official source of unlimited ROS information is the official website [33] contained unlimited resources like wiki pages - documentation, questions and answers, blog and forums.

5.6.1 Why ROS

Several reasons why ROS is used and accepted by the community and more importantly why it is chosen as a tool in this master thesis:

- Turtlebot the robot used in this work runs ROS,
- ROS is free and open source,
- All necessary tools, libraries, and packages are provided by ROS,
- Reusability of already implemented code, take away the need for reinventing the wheel,
- Continuously growing community with great amount of accumulated knowledge,
    - Researchers, developers, students share their work building a big heritage of work,

32 https://robots.ros.org/

33 http://www.ros.org/

- Distributed approach makes ROS stable and hard to fail.
- ROS can be distributed across multiple machines thanks to its peer-to-peer network of processes
- ROS follows a modular approach,
- Components are easily integrated into ROS thanks to ROS messaging system,
- ROS is maintained by many people.
- ROS is accepted by the community.
- Fast way for building, maintaining and improving robots features,
- Control of underlying low level hardware is possible with ROS.
- Increasing number of companies use ROS for their robots
- Available for wide range of applications.
- Theoretically ROS is language independent. It is implemented in Python, C++ and Lisp and there are experimental libraries in Java and Lua.
- Many ROS powered robots already exist [34]
- ROS can scale.
- No need for a robot experimentation can be performed in a simulation.
- Test can be performed though integrated unit/integration test framework “rostest”.
- Code developed in ROS could be integrated with other robot software frameworks.

Maybe a negative aspect of ROS is the learning curve which is pretty steep especially at the beginning, Also ROS is tasted only with Unix based Operating Systems like Ubuntu and Mac OS X. But the previous positive facts outweigh this negative aspect.

5.6.2 ROS Distributions

[35]ROS is out there for several years now starting from 2008 when it first appeared. ROS distribution is how ROS separated in different versions that are composed of different ROS packages. ROS distributions are similar to Linux distribution. The aim of these distribution is to provide to the developer a foundation of code. After the distribution release changes

34 https://robots.ros.org/category/ground/

35 http://wiki.ros.org/Distributions

are limited to bug fixes and improvements that doesn't break the core packages under the ros-desktop-full, this applies for the whole community except of the higher level packages.

The distributions are separated two types the ones that are LTS this are with the Long Term Support and other that are not LTS. The latest ROS LTS release is ROS Kinetic Kame (released date: May 23rd 2015) which is the recommended distribution from ROS official website. Although in this master thesis we used ROS Indigo Igloo[36] (released date: July 22ng, 2014) which is an LTS ROS distribution supported until April, 2019. This distribution was chosen because of its compatibility with Turtlebot 2 robot. Also it is possible to create your own ROS distributions in case an already released distribution doesn't cover your needs for the type of robot you are creating.

Finally it worths mentioning that not all ROS distributions are compatible with all Linux Ubuntu versions, sometimes there is one to one compatibility for example ROS Indigo Igloo is compatible with Linux Ubuntu 14.04 LTS Trusty. Also not all ROS distributions are backwards compatible with each other. This happens because of several changes introduces between the version in message definitions. This leads to disability of ROS nodes to communicate with each other. So it is not recommended to combine nodes implemented on different ROS distributions.

Figure 11: Indigo ROS logo, source: http://wiki.ros.org/Distributions

5.7. Turtlebot

[37] [38]Turtlebot as described on the official website is a low-cost, personal robot kit build on top open-source robotic software. Turtlebot was created at Willow Garage by Melonee Wise and Tully Foote in November 2010. Turtlebot uses ROS and it has several integrated

36 http://wiki.ros.org/indig & http://wiki.ros.org/Distributions

37 http://www.turtlebot.com/, https://en.wikipedia.org/wiki/TurtleBot, http://wiki.ros.org/Robots/TurtleBot, http://wiki.ros.org/turtlebot/Tutorials/indigo

38 https://spectrum.ieee.org/automaton/robotics/diy/interview-turtlebot-inventors-tell-us-everything-about-the-robot

functionalities and it also is available for navigating around, see in 3D and build useful and exciting applications. Turtlebot is a solid base for developing many practical, research, and scientific projects alongside with ROS. During the time this work was written there were three main versions of the Turtlebot in the market : Original Turtlebot 1(discontinued), Turtlebot 2 Family, and latest Turtlebot 3 Family. Nowdays Turtlebot can be found anywhere and it can be purchased from numerous distributors and partners around the world accessible from the official website[39].

The main parts of any TurtleBot are the following:

- All Turtlebots use ROS - Robotic Operating System
- Although Turtlebot is a specific product it is released under the FreeBSD Documentation License. In a nutshell it is open source [40].
- Turtlebot main hardware includes:
  - Kobuki Base,
  - Asus Xion Pro Live,
  - Netbook (ROS Compatible),
  - Kinect Mounting Hardware,
  - TurtleBot Structure,
  - TurtleBot Module Plate with 1 inch Spacing Hole Pattern.
- The robotic software development environment includes:
  - An SDK for the TurtleBot,
  - A development environment for the desktop
  - Libraries for the visualization, planning, and perception, control and error handling,
  - Demo applications.

39 https://www.turtlebot.com/

40 http://www.turtlebot.com/opensource/

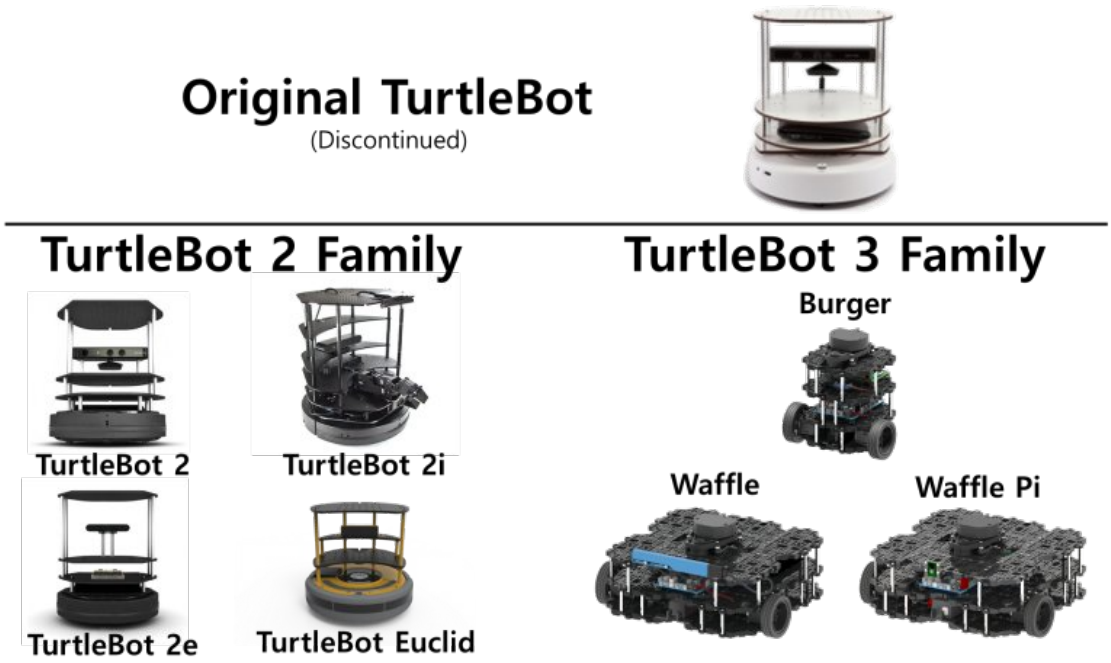

Figure 12: Turtlebot Robots of different types. Turtlebot 2 is used in this work, source: http://wiki.ros.org/Robots/TurtleBot

5.7.1 Turtlebot 2

[41] This chapter is dedicated to a brief presentation of Turtlebot 2 which is a 2nd generation Turtlebot robot being compatible with "Specification for TurtleBot Compatible Platforms" described on the ROS official website [42].

As you can observe on the next picture are presented the main hardware, software included of Turtlebot 2 robot [43].

41 https://robots.ros.org/turtlebot/, https://www.turtlebot.com/turtlebot2/

42 http://www.ros.org/reps/rep-0119.html

43 http://www.turtlebot.com/turtlebot2/

Table 1: Turtlebot 2 Hardware, Software, License, source - https://www.turtlebot.com/turtlebot2/

Hardware	Software	Open Source
The main hardware includes: • Kobuki Base • Asus Xion Pro Live • Netbook (ROS Compatible) • Kinect Mounting Hardware • TurtleBot Structure • TurtleBot Module Plate with 1 inch Spacing Hole Pattern	The robotic software development environment includes: • An SDK for the TurtleBot • A development environment for the desktop • Libraries for visualization, planning, and perception, control and error handling. • Demo applications	TurtleBot is an open source hardware project as described by the **Open Source Hardware Statement of Principles and Definition v1.0.** It is released under the **FreeBSD Documentation License**. See the **documentation** page to download the designs. open hardware

Kobuki base is the main part that composes Turtlebot 2 is that part on top of which the rest parts of Turtlebot are built. Netbook is that part of Turtlebot that hosts ROS and provides all the necessary abilities to Turtlebot.

There are several differentiated variations of turtlebot the one used in this master thesis is **TurtleBot 2 with a netbook**. Other variations are :

- **TurtleBot 2e**: which is a revision of the TurtleBot but with netbook being replaced with a single board computer such as the 96 Boards CE computer, the DB410c.
- **TurtleBot 2i**: it has some extensions added to the previous Turtlebot which include modular chassis and a native support of robotic arms. Specifically it includes Pincher MK3 4 DOF Robotic Arm which adds functionality to Turtlebot to interact with small objects, transforming the robot to a mobile manipulator.
- **TurtleBot Euclid**: It is a variation of TurtleBot 2 with main aim to make the out of the box experience as easy as possible. It comes preassigned with Intel Euclid Development Kit reinstalled with a web-interface to access the development environment.

It is even possible to build a Turtlebot from the scratch by purchasing all the necessary

parts separately and using the documentation provided by the official ROS creators[44].

In this work Turtlebot 2 as already described in previous chapters is being used for navigation in a warehouse environment and detection of objects through RFID tags. It obvious that there are hundreds of other implemented projects on top of Turtlebot and even more other possible applications Turtlebot is waiting to be created.

5.8. Unix – Linux – Ubuntu Operating System

The reason this chapter is introduced to this master thesis is the fact that at this moment ROS is mostly compatible with Ubuntu operating system. Ubuntu is an open-source operating system. On the official website it is presented as “Ubuntu is an open source software operating system that runs from the desktop, to the cloud, to all your Internet connected things” [45]. It is a Linux based operating system produced by Canonical. It is build on top of Debian's architecture and infrastructure. It provides numerous features and security directly “out-of-the box”. There are many Ubuntu releases not all of them are compatible with every ROS distribution so ROS - Ubuntu distributions compatibility should inspected.

5.9. ROS Basics

The main tool used in this thesis is ROS which controls the behavior of the robot. Although presenting and describing ROS is not the purpose of this work. It could be beneficial to present the main parts of it.

5.9.1 ROS filesystem

[46]ROS filesystem is build on top of Linux filesystem. The main parts of ROS file system are:

- Packages: which contain libraries, tools, executable, etc. (lowest level)
- Package Manifest files: contain the description of the ROS packages and the dependencies of the packages.
- Stacks: Collection of packages (higher level library)
- Stack Manifest files: contain the description of the Stacks and the dependecies.

ROS provides basic command line tools for the fast navigation between ROS packages. The main commands are

- **rospack**: information about packages

44 Documentation provided by the official ROS creators http://www.turtlebot.com/learn/ and here http://www.turtlebot.com/build/

45 https://www.ubuntu.com/

46 http://wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem

- **rostack**: information about stacks
- **roscd** : moving fast between ROS packages
- **rosls** : ls for ROS

ROS provides a set of shell commands called **rosbash** which contains useful bash commands: roscd, rospd, rosd, rosls, rosed, roscp, and rosrun.

5.9.2 Building packages - Catkin

[47]To build ROS packages, ROS provides its is possible to use **catkin** or **rosbuild**. In this work for the developed code were used Catkin. It provides a fast, reliable and easy way to organize and build your ROS packages.The command line tool **catkin_make** provides the calls to cmake and make in the standard Cmake workflow.

5.9.3 ROS packages

[48]The reason why this chapter is added is because packages is one of the most important concepts. ROS software is organized in packages, everything in ROS is contained in packages. A package can contain different type of files which can be ROS nodes, libraries, datasets, configurations, etc. The concept of packages provides ROS with reusability. Packages is the smallest part of ROS that can be build and released. Every ROS package follow common structure the main directories and files are:

(a) include/package_name

(b) msg/

(c) src/package_name

(d) srv/

(e) CMakeList.txt

(f) package.xml

(g) CHANGELOG.rst

Also ROS packages provide as with several command line tools which are rospack, catkin_create_pkg, catkin_make, rosdep, and rqt. To create ROS package **catkin_create_pkg** is available and **catkin_make** command is used to build a ROS package. To build successfully a ROS package the appropriate changes have to be applied in CMakeList.txt and in package.xml files.

47 http://wiki.ros.org/ROS/Tutorials/BuildingPackages

48 http://wiki.ros.org/Packages

5.9.4 ROS Graph concepts

[49]The main ROS graph concepts are:

[50]**Nodes:** It is a simple executable that uses ROS to exchange messages between each other. Ros node provides a command called **rosnode**. Nodes can be written in python (rospy), in c++ (roscpp), and other languages.

Messages: The communication between nodes is performed by publishing messages to topics. Messages are simple data structures and supports standard primitive types and more complex data types of structures and arrays. For the messages there are msg files that describe the fields of the messages. Through this files source code for different programming languages is generated.

Topics: ROS nodes publish or subscribe to topics to send or receive messages and achieve communication.

Master: It is used as a name service for ROS. With its helpt ROS nodes locate each other.

rosout: Similar to stdout or stderr.

roscore: A combination of ROS Master, ROS Parameter Server and rosout logging node. Roslaunch starts automatically roscore.

5.9.5 ROS Services and Parameters

Service provide another way of communication for the nodes with each other. Through services nodes send requests and receive responses. ROS provides a command line tool **rosservice.** It is possible to create srv files for describing services. Which is made of two parts a request and a response.

ROS provides a Parameter Server stores parameters which are accessible through network APIs and nodes use it to store and retrieve parameters at runtime. ROS provides **rosparam** command.

5.9.6 ROS publisher/subscriber

For both publisher and subscriber a different ROS node is created. It is possible to create such nodes in c++ and python. To create a ROS node you need an already created catkin workspace. ROS publisher publishes messages to a topic and at the same time on the other side a ROS subscriber subscribes and consumes messages from that topic. In python the abstract code would look like this [51]:

Publisher

49 http://wiki.ros.org/ROS/Concepts & http://wiki.ros.org/ROS/Patterns/Communication

50 http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

51 http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

```
pub = rospy.Publisher('chatter', String, queue_size=10)
...

hello_str = "hello world %s" % rospy.get_time()
pub.publish(hello_str)
```

Subscriber

```
def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)


…

rospy.Subscriber("chatter", String, callback)
rospy.spin()
```

To execute ROS publisher/subscriber first run **roscore** command and after **rosrun <package> <node>** for each node.

5.9.7 ROS service/client

For both service and client executables different ROS nodes are created. Services receives requests from clients and publish responses to clients. Services and clients could be written in both python and c++. The code in python would look something like this [52]:

Service

```
 def handle_add_two_ints(req):

      print "Returning [%s + %s = %s]"%(req.a, req.b, (req.a + req.b))

      return AddTwoIntsResponse(req.a + req.b)

...

s = rospy.Service('add_two_ints', AddTwoInts, handle_add_two_ints)
rospy.spin()
```

Client

```
rospy.wait_for_service('add_two_ints')
try:
    add_two_ints = rospy.ServiceProxy('add_two_ints', AddTwoInts)
    resp1 = add_two_ints(x, y)
    return resp1.sum
except rospy.ServiceException, e:
    print "Service call failed: %s"%e
```

To execute ROS service/client first run **roscore** command and after **rosrun <package> <node>** for each node.

52 http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28python%29

5.9.8 ROS dependencies

ROS packages sometimes need external libraries and tools called system dependencies to function properly. This dependencies are provided by the operating system and could be installed with a ROS tool called **rosdep** which download and install this dependencies. The dependencies needed by each ROS package are declared in the manifest files of these packages. By typing the following **rosdep install [package]** you can download and install the required dependencies.

5.9.9 ROS basic commands

Ros provides different command line tools[53] several; of these tools are presented here:

- **roscd**: cd directly to the desired location by name without the need of full path.
- **roscore**: runs the ROS core stack.
- **rosdep**: downloads and installs system dependencies of a package
- **rqt-dep**: present system dependencies in a graph form
- **rosed**: used for editing ROS files
- **rosd**: lists directories inside of a package
- **rosls**: lists files inside of a package
- **roscp**: copies files
- **roscreate-pkg**: creates the common Manifest, CMakeIlsts, and other files necessery for a ROS package.
- **rosrun**: runs executables
- **roslaunch**: launches a set of nodes on local and remote machines defined in an XML configuration file.
- **rosmake**: builds all dependencies in ROS packages.
- **rosmsg**: command line tool for displaying information about ROS Message types.
- **rospack**: command line tools for retrieving and displaying information about available ROS packages.
- **catikin_create_pkg**: creates new package
- **catking_make**: compiles created package

53 http://wiki.ros.org/ROS/CommandLineTools

- **rosparam**: command line tool for handling ROS parameters.
- **rossrv**: command line tool for displaying information about ROS Service types.
- **rosstack**: command line tool for getting information about ROS stacks.
- **rosservice**: command line tool for listing and querying ROS Services.
- **rostopic**: command line tool for displaying information about ROS Topics
- **rosnode**: command line tool for displaying information about ROS Nodes.
- **rqt-(tools)**: graphical tools provided by ROS.

5.9.10 ROS actionlib

[54]Actionib stack is a standardized interface for interfacing with presentable tasks. It is used in the context of the project for moving the base of the robot to a specific predefined location on the map. This is done though a spinning thread which waits to receive a goals with the position. This functionality provides the user with the ability to send a request to a node to perform a specific task, and later receives a response to the previously send request.

Through actionlib package it is possible to create long running goals with preemption functionality. And also clients can send requests to the server though a client interface. The communication of the ActionClient and ActionServer is performed though ROS Action Protocol. Simple API is provided for client and server to request and execute(though function calls and callbacks) goals.

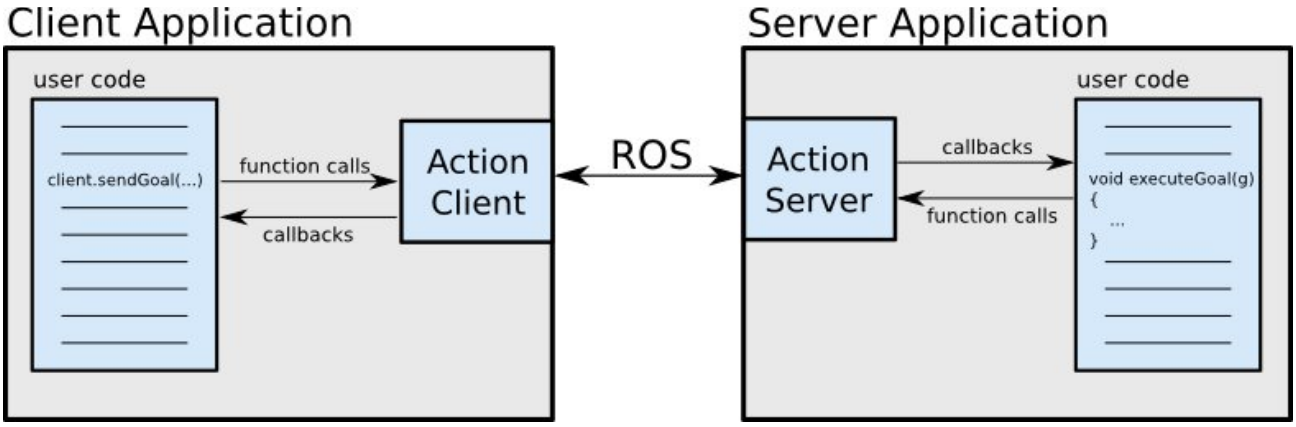

Figure 13: This pictures shows how the communications is performed between the client and server application, source: http://wiki.ros.org/actionlib

The communication is performed though Goal, Feedback and Result messages. In the context of this project Python API for SimpleActionClient was used.

54 http://wiki.ros.org/actionlib

5.9.11 ROS parameters

[55]ROS provides as with a parameter server which is a shared, multivariable dictionary that is traversable through API accessible via network provided by ROS. Through the parameter server nodes can modify and retrieve data from parameters. Parameter server is implemented in XMLRPC and runs in ROS Master. The usage of XMLRPC makes the API accessible though normal XMLRPC libraries. ROS paramteres support specific data types like 32-bit integers, Booleans, Strings, Doubles, etc. As already presented previously in ROS basic commands rosparam tools is used for working with ROS parameters[56]. The YAML syntax is used by rosparam command line tool to get and set parameters on the Parmeter Server. The <rosparam> let us use the rosparam YAML files. The <rosparam> tag is put in the .launch file. It is preferable to use parameters for static data because of luck of high performance.

The reason why ROS parameters are presented here is because in the context of this project's ROS parameters where used and appeared to be useful. Specifically custom parameters where used to store information like duration for the robot to complete the goal set or the radius around the goal for considering a goal set to be successful. The data stored in parameters is for configuration which means it is static.

5.9.12 Slam

[57]Slam stands for Simultaneous Localization and Mapping. Slam is a technique for constructing and updateing a map. Slam technique provides us with the ability to keep track of the current position of the robot and at the same time build a map of an unknown environment. ROS slam node is called slam_gmapping. In our project our robot uses RPLidar and Kinect to create a 2-D occupancy grid from laser and pose data collected by the robot Turtlebot.

5.9.13 ROS Simulations

The implementation of this master thesis started in a simulated world and later it was transferred to the physical turtlebot in the lab.

There are two reasons for someone to use ROS in a simulated environment. First reason is that maybe a physical robot is not available at the moment of development and second reason is that maybe we want to try and test everything in a perfectly simulated environment before transferring the application into the physical robot. It is important to mention that no dramatic changes are required for the transferred application.

Next are discussed some of the advantages using Turtlebot in simulation:

55 http://wiki.ros.org/Parameter%20Server

56 http://wiki.ros.org/roslaunch/XML/rosparam

57 https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping & http://wiki.ros.org/Build%20a%20map%20with%20SLAM

1. Since the hardware is not required in the simulation and everything is done by software, we avoid the possibility of damaging the expensive hardware.
2. It is less time consuming in contrast of using a physical robot.
    1. no configuration is needed,
    2. no battery charging is required,
    3. no hardware repairing,
3. Within the simulated environment it is possible to run experiments in different conditions with different types robots, in different conditions, with multiple different sensors etc. It is possible to save and reproduce the exact conditions to repeat certain behaviors of the robot or perform multiple tests on the system with different parameters set.

After using the perfect world of simulation it is always important to try the final result on the physical robot. Because simulation it is not capable of reproducing all the parameters that can appear in real world.

5.9.14 ROS Gazebo

ROS provides us a package called **turtlebot_simulator** that contain launchers for three different simulation environment for TurtleBot. The positive fact about ROS simulation API is almost unchanged compared to the physical Turtlebot. In this master thesis for performing simulations only the Gazebo simulator was used.

- First of all Gazebo is an open-source simulator.
- It is a multi-robot simulator which allows us to perform both indoor and outdoor simulations.
- Through Gazebo Tutrlebot can interact physically interact with objects.
- The feedback produced and collected from sensors is realistic.
- It very simple to use, intuitive and well-designed simulator which gives as the possibility to run, test algorithms and design robots, etc.
- Gazebo uses URDF 3D model.
- It is possible to design custom world in Gazebo.
- It is possible to add sensors to the robot in Gazebo.
- Etc.

5.9.15 ROS RVIZ

Rviz is a tool for ROS 3D visualization, data collected from sensors is visualized with RVIZ as well as the information about the states. Data produced from sensors like rplidar, kinect, infrared, camera etc. can be visualized with RVIZ. It is one of the most important and useful tools while working with ROS and Turtlebot and it is used both in simulation and physical world experiments. RVIZ tool comes with ROS installation. It gives us the possibility to visualize all of the data from navigation stack. It visualizes the world how the robot sees it, for example in our case it is useful because it is possible to see the point cloud around the Turtlebot, the created map from the RPLIDAR and the path planned by the global and local planner. Also through RVIZ it is possible to specify Robots location on the map though “2D Pose Estimate” set a goal to the robot through the “2D Nav Goal”, measure distance on the map through “Measure” and get the relative coordinates through “Publish Point”.

5.10. ROS Navigation Stack

[6]ROS navigation stack[58] [59] provides us with a set of algorithms that take information from sensors, odometry an a goal position and produce velocity commands to move the mobile base of the robot. The user through standard messages can control the robot. ROS navigation stack can be applied and used with any robot that runs ROS with only several configurations performed.

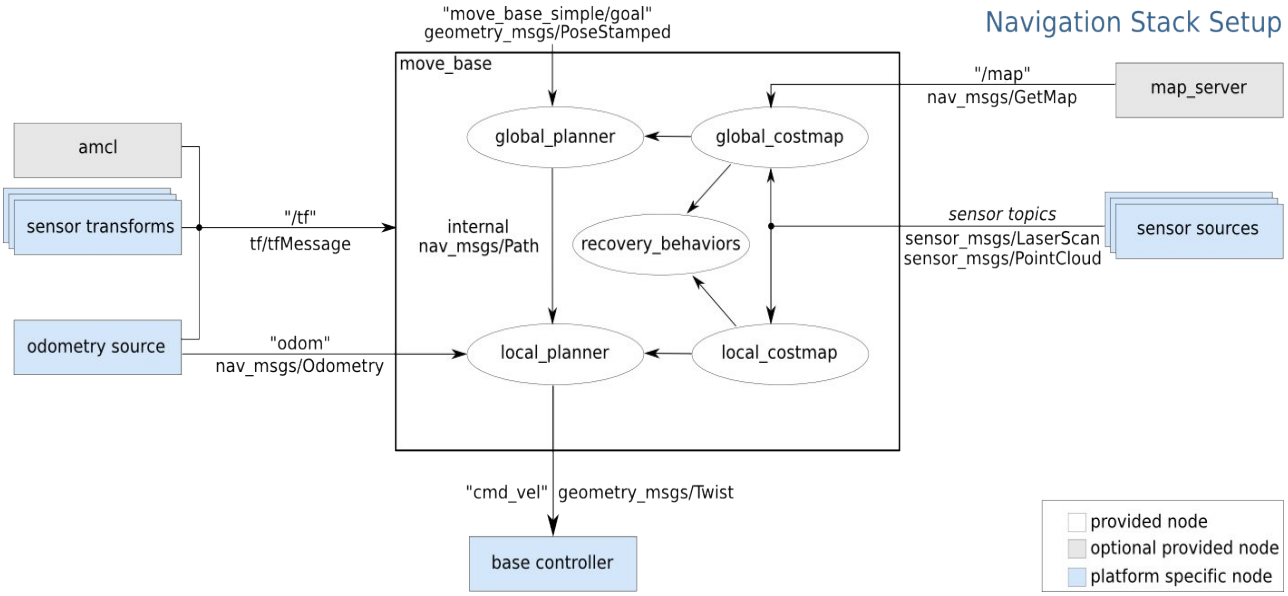

Figure 14: In this diagram is presented the organization of the ROS Navigation Stack.

58 http://wiki.ros.org/navigation

59 For more information about ROS navigation stack and other ROS parts refer to the book Effective Robotics Programming with ROS Third Edition by Anil Mahtani, Luis Sanchez, Enrique Fernandez, Aaron Martinez, Publisher Packt, 2016.

ROS Navigation Stack is organized in a specific way which is presented in the following diagram.

The main parts from ROS Navigation Stack affecting our project are the 1)Gmapping for the creation of the map from the data collected from RPLidar 2)amcl for the localization of the robot on the map, and 3)move_base for moving the robot pro grammatically through Pyhon code by setting a goal.

ROS Navigation stack is very importan for this project because it is used in the following parts:

1. Map creation : before navigating in an unkown environment a maps has to be created. So later it would be easier to perform movement, navigation and path planning for the robot.

2. Localization : by using ROS navigation it is possible to perform localization of the robot by estimation the relative position of the robot on the map. To perform this functionality ROS navigation stack provides as with amcl package.

3. Path Planning : the path is planned for the robot to follow. In this project RVIZ and move_base packages are used to send goals (pose) to the robot's mobile base.

5.10.1 Transform Frames(tf) software library

[60]TF manages the transform tree, lets us to keep track of multiple coordinate frames over time. Tf defines offsets for translation and rotation between different coordinate frames. tf maintains the relationship between coordinate frames in a tree structure buffers in time, and lets the user transforms points, vectors, etc between any two coordinate frames at any point in time. The robotic system is composed of 3D coordinate frames that change over time, for examples there is world frame, base frame, gripper frame, head frame, etc. tf keep track of all these frames over time, and allows you to ask questions about position of each part of the robot, about the pose of the objects relative to other frames, about the pose of the base frame. For example if we have a mobile base “base_link” on top of which we have attached a laser “base_lase” with 10cm backwards and 20 cm above, the tf will give us a translational offset that relates the “base_link” frame to the “base_laser” frame, so by calling the tf library we will get the transformation. In the following picture is represented a simple example for better understanding of the transformation explained previously.

60 http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF

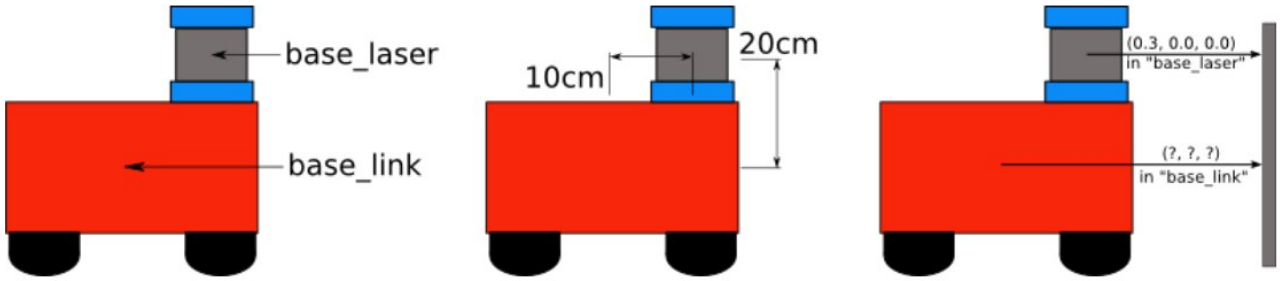

Figure 15: This picture presents an example where a laser was installed on top of a robot so a tf library was used to get the translational offset that creates a relation between the base_link and the base_laser frame. Source: http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF

We can create if needed tf broadcaster and then a listener. It is important to remember that the geometry of the robot is specified in the URDF file where it is configured. If we have our tf broadcaster running we can observe the transformation tree by running the following command : ros tf view_frames.

In this project tf transform was used when RPLidar sensor was installed on top of the Turtlebot. It was important to use the tf transformation of the RPLidar sensor for the creation of the map and later for accurate navigation in the created map.

5.10.2 Sensor Sources

[61]On a robot there could be different types of sensors for scanning the area and avoid obstacles in the world. The information from the sensors is consumed by the navigation stack. For the navigation stack Move Baseto be able to get this sensor information sensors must publish to : sensor_msgs/LaserScan or sensor_msgs/PointCloud messages over ROS. Also it is not always necessary to implement this part because several sensors have already covered this part by default.

5.10.3 Odometry Source

[62]As it is obvious from the Navigation Stack picture, navigation stack gets the robot's Odometry information. Odometry should be published and consumed by the navigation stack. Odometry information is published using **tf** and **nav_msgs/Odometry message**. tf first detects robot's location in the world and relate sensor data to a static map. The odometry source is responsible for publishing the transform about the coordinate frame that it is responsible and manages. The nav_msgs/Odometry message holds an approximation of the position(in odometric frame) with the geometry_msgs/Pose message and velocity(in child frame – mobile base) of a robot with geometry_msgs/Twist message.

61 http://wiki.ros.org/navigation/Tutorials/RobotSetup/Sensors

62 http://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom

5.10.4 Base Controller

A robot is controlled though the Base Controller. As shown in the previous Figure it is platform specific so it is not provided by ROS. Base controller node must subscribe to the topic called “cmd_vel”, and by generating the proper motor commands it should move the platform with velocities (angular and linear).

5.10.5 Map Server

[63]Map Server is an optional provided node and it is not required by the navigation stack. Map server gives all the necessary functionality for map handling. It is possible to create a map of our environment dynamically and later save them in a custom file with .yaml format which contains the map meta-data and an image file (for example could be .png format) which contains the picture of the occupancy data. The occupancy grid is presenting the state of the wold in which the robot navigates, with a different color depending on the occupancy of each pixel(white:free, darker-blacker:occupied, in between:unknown) this image is in gray scale(although colored images are allowed). The .yaml format has to be organized in a specific manner, example is presented in the following picture:

```
image: testmap.png
resolution: 0.1
origin: [0.0, 0.0, 0.0]
occupied_thresh: 0.65
free_thresh: 0.196
negate: 0
```

Figure 16: Example of how .yaml format should be organized, source: http://wiki.ros.org/map_server

required fields: image, resolution, origin, occupied_thresh, free_thresh, negate

optional parameter: mode

Map_server and **map_saver** command line tools are provided by ROS. Map_server is a ROS node that reads a map from a disk and offers it through a ROS service. Map_saver is a ROS tool that save a created map to the disk.

5.10.6 AMCL (Adaptive Monte Carlo Localization approach)

[64]AMCL is an optional provided node. AMCL is a probabilistic localization system that is responsible for the robot movement in 2D. The Adaptive Monte Carlo Localization approach helps the robot to detect its pose in a known map that is provided or created from laser scan. Amcl uses as input the maps created from laser scan, laser scans, and transform messages, amcl produces an estimation of the position(pose) of the robot. It is important to note that AMCL is configurable algorithm and in some circumstances it is very useful for optimization reasons.

63 http://wiki.ros.org/navigation/MapBuilding & http://wiki.ros.org/map_server

64 http://wiki.ros.org/amcl

5.10.7 Move Base

[65]Move_base package is one of the most important parts of ROS navigation stack used in this master thesis because it used for the navigation of the Turtlebot and specifically for sending goals. The move_base package is provided by ROS directly and it contains all the necessary navigation tools for the Turtlebot navigation pro grammatically.

Move base_base package provides us with the implementation an action and specifically the actionlib package presented previously, that why though move_base we send the goals to our robot. This is done by specifying the target positions and orientation so the robot moves towards these goals and the mobile base of the robot attempts to reach the destination.

Move_base node links together both a global and a local planner which are attached to interfaces of nav_core. To complete the navigation tasks set move_base keeps two cost maps, one for global and on for local planner specified in costmap_2d.

Move_base is one of the most important parts of ROS navigation stack. Through the move-base node the configuration and the interaction with the navigation stack of the robot is succeeded. The picture of the Navigation Stack Setup depicts the abstract view of the move_base node and its interaction with other components of the navigation stack. As shown in following picture move_base provided default recovery behaviors, with the help of these behaviors it can achieve the goal that is set.

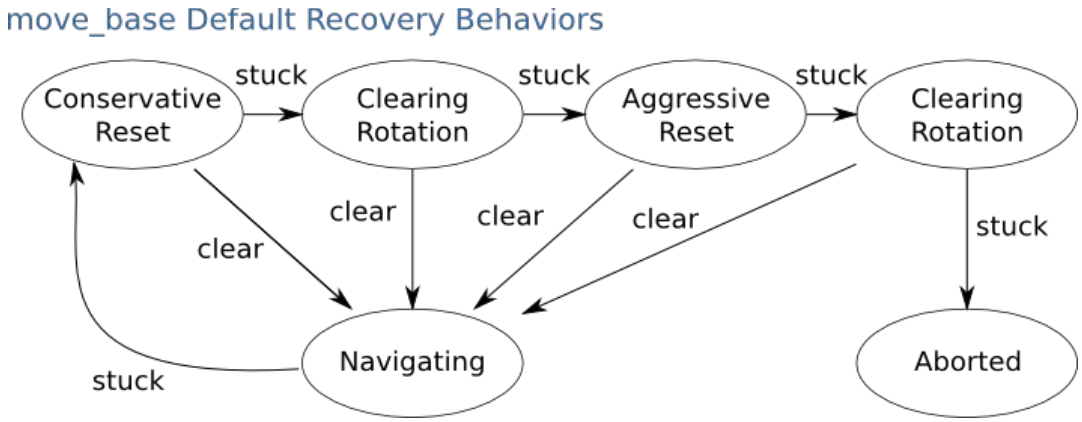

Figure 17: This picture shows the default recovery behaviors that move_base apply, source: http://wiki.ros.org/move_base

Before running the move_base node on a robot it is important to configure this robot properly. A properly configured robot will make an effort to accomplish a goal pose with its base within user specified tolerance. Move_base node will reach the goal within the tolerance set by the user or signal failure back to the user. As presented in the above picture move_base node may optionally perform recovery behaviors when the robot perceives itself as stuck. Next the recovery actions are presented to attempt to clear out

65 http://wiki.ros.org/move_base

space:

1. Clear obstacles outside of user-specified region on the robot's map.
2. Perform if possible an in-place rotation to clear out space.
3. If step 2 fails the robot aggressively clears its map (removing all obstacles outside rectangular region in which it can rotate in place)
4. Following to step 3 the robot will perform in-place rotation.
5. If all previous steps fail the goal is considerd as infeasible by the robot and notify the user that it has aborted.

The configuration of this recovery behaviors is done through the **recovery_behaviors** parameter and disables through **recovery_behavior_enabled** parameter.

The move_base node is implementing a **SimpleActionServer**. The goals passed to move_base node are containing messages of **geometry_msgs/PoseStamped** type. Finally it is important to mention that move_base node contains also other ROS navigation components that has their own ROS APIs : **costmap_2d**, **nav_core**, **base_local_planner**, **navfn**, **clear_costmap_recovery**, and **rotate_recovery**. This packages are used as behavior plugins for the move_base node.

5.10.8 Base Local Planner

[66]The base_local_planner package given a plan to follow and a costmap provides a controller that produces velocity commands that make the mobile base in the plane. The base_local_planner package implements both Trajectory Rollout and Dynamic Window approaches to local robot navigation on a plane. The purpose of the controller is to connect the path planner with the robot. With the help of a map the planer produces the movement trajectory so the robot can move from a starting position to a goal position. A value function represented as a grid map is created locally around the robot. This value function matches traversing costs to the grid cells. Finally the controller produces and sends dx, dy and dtheta velocities to the robot.

66 http://wiki.ros.org/base_local_planner

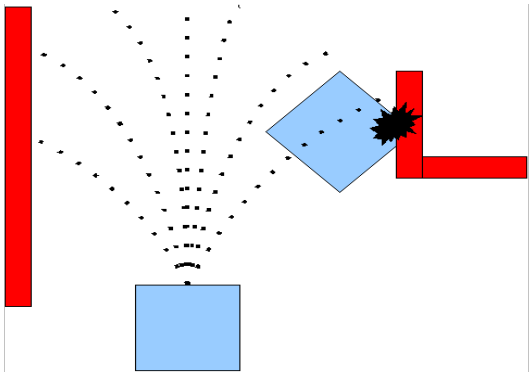

Figure 18: Representation of trajectory planning made by the base local planner, source:http://wiki.ros.org/base_local_planner

5.10.9 Global Planner

[67]Global_planner package implements a path planner library and a node for the robot navigation. Global planner package provides different parameterizations. Global planner uses the static global cost map and depending on the costs it creates a Gloabl Plan from a start point to and end point.

5.10.10 Clear Costmap Recovery

[68]Clear costmap recovery provides a simple recovery behavior for the navigation stack that simply clears the space in the navigation stack's costmaps returning to the static map beyond a given radius around the robot.

5.10.11 Rotate Recovery

[69]Rotate recovery package implements a recovery behavior for the ROS navigation stack. Through rotate recovery navigation stack rotates the robots 360 degrees in an attempt to clear space in the costmaps that surrounds the robot. The rotation is performed if possible and no local object prevents the 360 degree rotation.

5.10.12 Costmap_2D

[70] Costmaps represent the data collected by the sensors from the surrounding environment. Costmap_2D package implements the functionality of data collection from the sensor and representation in a 2D and 3D occupancy grids. Also in the 2D costmap inflated costs are represented based on the occupancy grid and the user specified inflation radius.

67 http://wiki.ros.org/global_planner

68 http://wiki.ros.org/clear_costmap_recovery

69 http://wiki.ros.org/rotate_recovery

70 http://wiki.ros.org/costmap_2d

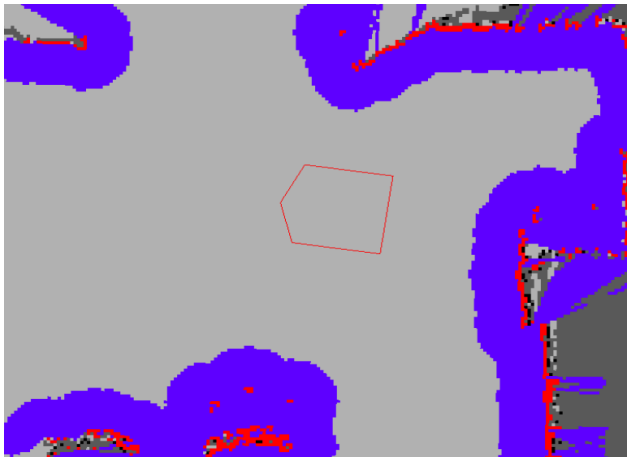

Figure 19: In this picture are observable different types of cells in a costmap. red: obstacles, blue:inflation around the obstacle, hexagon:robot footprint. source: http://wiki.ros.org/costmap_2d

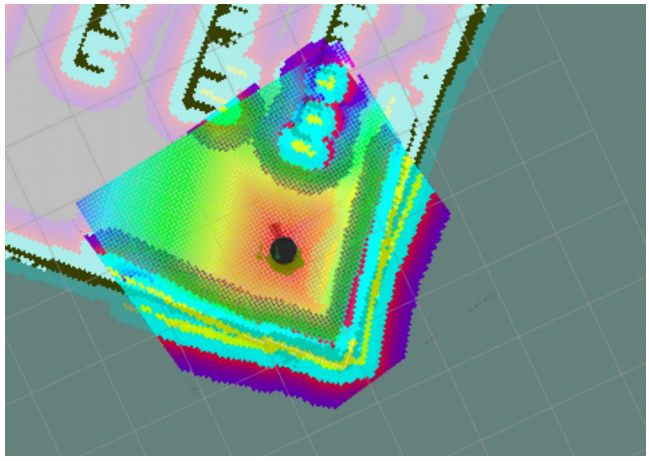

Figure 20: In this picture is presented a map in RVIZ. Black dot: Turtlebot, black lines: obstacles (static map), light blue inflation, etc.

Costmaps update themselves by automatically keeping a subscription to sensors topics over ROS. The map updates are performed in a rate specified in the update_frequency parameter. The updates on the costmap could be either a mark, a clear or both according to the sensor data. Each cell in a costmap can be Occupied, Free or Unknown space. For the proper navigation, obstacle detection and obstacle representation in costmap it is assumed that tf transforms are properly configured between the coordinate frames, gloabl_frame parameter, the robot_base_frame parameter, and the sensor sources.

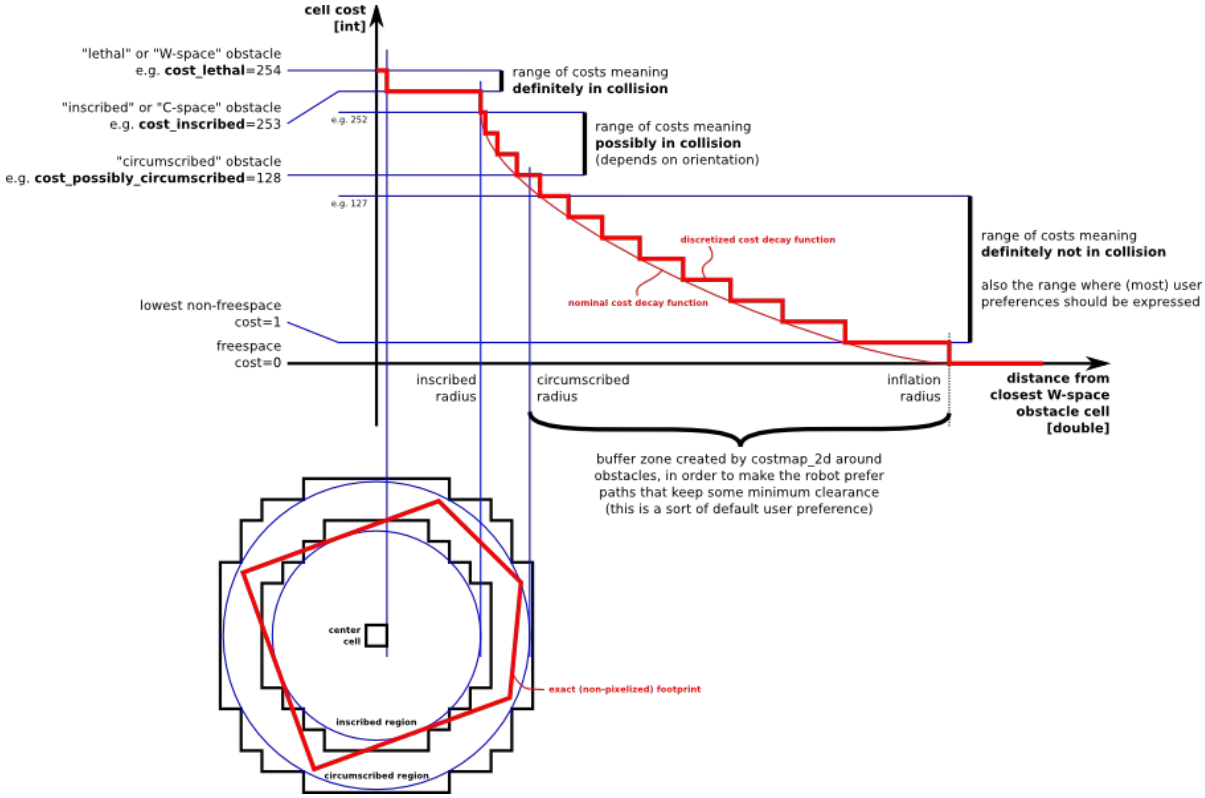

Figure 21: Five different symbols for the costmap values are defined. Lethal, Inscribed, Possibly circumscribed, Freespace, and Unknown costs. Depending of the distance from the object and the decay value which is used difined,all values are assigned between the Freespace and Possibly Circumscribed cell. Source: http://wiki.ros.org/costmap_2d?distro=indigo

The values in each cel of the costmap range from 0 to 255. The five different sybols each have a different value Lethal(255), Inscribed(253), Possibly circumscribed(128-252), Freespace(0), and Unknown(255).

5.10.13 nav_core

[71]Nav_core implements common interfaces for performing navigation actions for robots. Previously described parts of navigation stack like BaseGlobalPlanner, BaseLocalPlanner, and RecoveryBehavior interfaces are provided by nav_core package. Through this interfaces new versions of this planners, controllers, or recovery behaviors can be implemented. So as obvious from the ROS Navigation stack nav_core interfaces are the base interfaces for the navigation stack. The planners that are used as plugins in the move_base node must implement the interfaces provided by nav_core package.

71 http://wiki.ros.org/nav_core

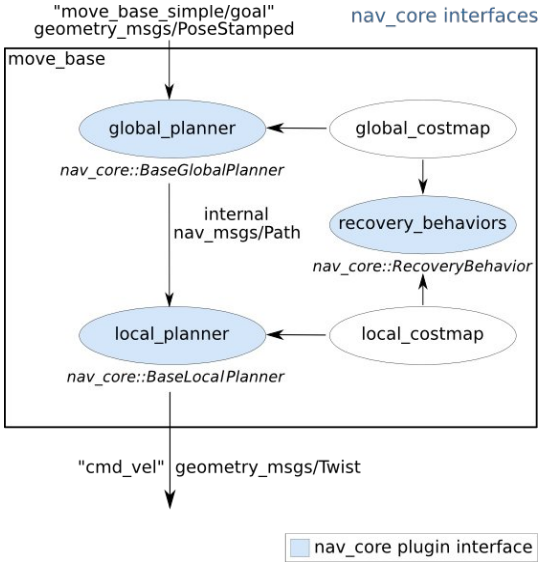

Figure 22: This picture presents the parts of ROS Navigation stack that adhere from nav_core interface, source: http://wiki.ros.org/nav_core?distro=indigo

5.10.14 Navfn

[72]Navfn package implements a fast and interpolated navigation function that creates plans for a mobile base. Navfn package produce a plan of minumum cost from a start point to an end point for a circular robot that operates on a costmap described previously.

5.10.15 Gmapping

[73]Slam is used in this project for Simultaneous Localization and Mapping. For this functionality ROS provides a package called Gmapping that implements a node called slam_gmapping. From the data collected from lasers and odometry coming from mobile base, gmapping node builds a 2D occupancy grid map.

5.10.16 ROS Sending Simple Goals

[74]To send simple goals to our robot Turltbot 2 we need to use several packages from ROS Navigation Stack. We want to send a goal with the desired location to the robot and it should complete this task by reaching the set location. To complete the goal the robot needs (a) previously created static map, (b) rplidar for scanning the surrounding environment, (c) amcl for localizing, (d) Local and global planners, and (b) generally ros navigation stack. To send Simple Goals to the robot we use **actionlib**[75] library and **move_base** package.

72 http://wiki.ros.org/navfn

73 http://wiki.ros.org/gmapping

74 http://wiki.ros.org/move_base & http://www.hotblackrobotics.com/en/blog/2018/01/29/action-client-py/

75 http://wiki.ros.org/actionlib

Through actions (actionlib) it is possible to send requests and receive responses. Actionlib provides as with feedback functionality which is useful because it provides ActionCLient with information about the progress of a goal like current position. Client and Server communicate through **Goal, Feedback and Result** messages. Actionlib is a ActionClient and ActionServer who communicate through a “ROS Action Protocol”. Actionlib provides as with the API for setting an Action Server and an Action Client, so client can request goals, and server can execute these goals through function calls and callbacks.

Move_base package provides as with the implementation of and an action (actionlib). We give a goal in the robot's world and the move_base will perform all the necessary steps to complete the goal with the mobile base of the robot. Move_base node implements a SimpleActionServer which implements a single goal policy on top of the AcitonServer class. Goals sent are **geometry_msgs/PoseStamped** message type. The communication with the move_base node is performed through the implemented SimpleActionCLient interface. Move_base combines global planner, local planner and it holds two costmaps to accomplish the navigation tasks. Finally we send a goal to our robot, while all the rest is performed by the configured navigation stack.

Next are presented several parts from the implemented code for sending Goals.

```
# Create the Action Client called 'move_base' with action type MoveBaseAction,
make the thread spin

self.__action_client = actionlib.SimpleActionClient('move_base', MoveBaseAction)

...

# Wait for the Action Server to start before it is able to receive goals

wait = self.__action_client.wait_for_server(rospy.Duration(5))

...

# Send the created Goal to Action Server

self.__action_client.send_goal(goal, self.done_cb, self.active_cb,
self.feedback_cb)
```

5.11. Turtlebot Basics

In the Appendix of this thesis are described the following issues

1. Turtlebot – Map Creation and Navigation

On top of Turtlebot is mounted Kinect sensor which is responsible of the scanning process of the surrounding area, also as we do it is possible to mount another sensor like a lidar for this purpose. After that ROS has all the necessary tools for the creation of the map. The

steps are simple and are enumerated in the appendix. It is possible to create a map of a physical world and a simulation as well for performing experimentation.

2. Turtlebot – Autonomous Navigation

Turtlebot has several ways to move around.

- By sending **velocity** commands on "/cmd_vel" topic to the robot this approach is not used in this work.
- By running **turtlebot_teleop** package so the turtlebot can be navigated from the keyboard, ps3 joystick or other input source with implemented launch file. This approach is used for the creation of the map.
- Using the **move_base** package alongside with several other packages for autonomous navigation by sending positions/goal of the desired position. This approach is used in the implemented algorithm for the navigation.

3. Turtlebot – Obstacle Avoidance - Localization

Obstacle avoidance and localization are implemented in ROS by default. The robot localize itself by using the collected data from the laser and the odometry data that are available to the robot. Turtlebot detects with the help of packages like amcl, etc its position on the map. Data from kinect or lidar is published on a topic, this sensor produced data of the surrounding environment is used by other ROS nodes like amcl for the transformation from map coordinates to the move_base coordinates. The main task of amcl is with the help of the odometry and sensor data to compute the position of the robot on the map. Finally the position of the robot is represented on the map.

5.11.1 TurtleBot Navigation Stack

[76]According TurtleBot navigation stack we must consider 4 important things which are main features:

- **Key files** governing Turtlebot navigation are launch and yaml files contained on turlebot_navigation package on launch and param directories respectively.
- Move base provides Tutlebot with navigation motion. Move base contains cost maps which give it the opportunity to create global and local plans.
    - Planner
        - Change speed limits
        - Goal tolerance

76 http://wiki.ros.org/turtlebot_navigation/Tutorials/indigo/Setup%20the%20Navigation%20Stack%20for%20TurtleBot

        - Cost computing biases

- Amcl provides localization for TurtleBot
- Gmaping provides TurtleBot with map building capabilities

6. Turtlebot – ROS Implemented Algorithm for Navigation

One of the main parts in this master thesis is the implementation of algorithm for the navigation in an indoor environment (which could be used although for outdoor controlled environment as well). The main purpose of this algorithm is to set missions (goals) to the Turtlebot through ROS. Turtlebot or any other robot that is supporting ROS can be controlled through the program implemented. In the next paragraphs the main parts of this algorithm are described and the logic behind it.

6.1. Explanation of the desired movement Turtlebot should perform

Lets assume that we have a warehouse with multiple corridors in such case the robot can navigate inside of the warehouse in a specific manner. In the next pictures two possible routes are presented.

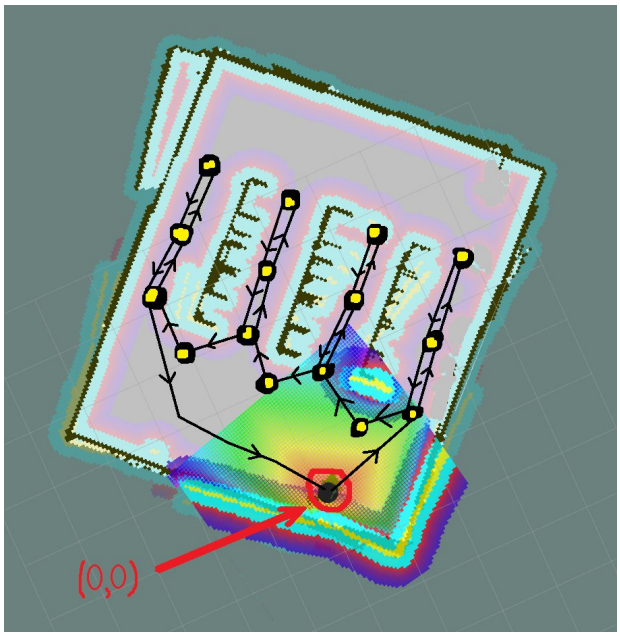

Figure 23: Robot movement first approach

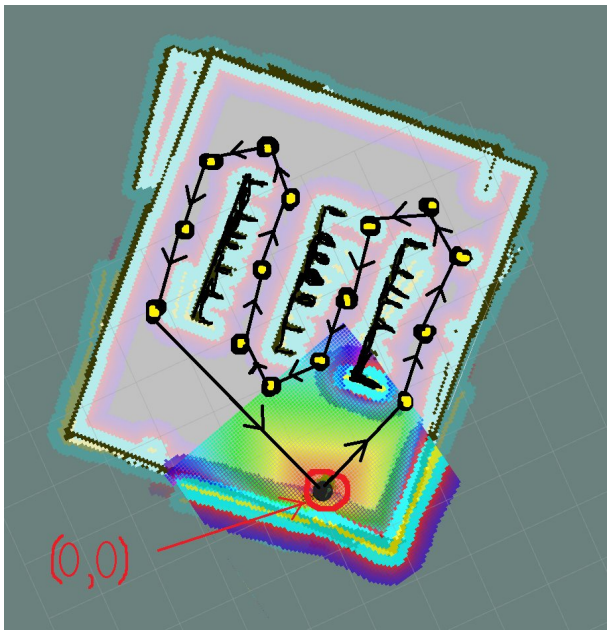

Figure 24: Robot movement second approach

Both pictures represent the scan of a warehouse from the RPLIDAR sensor mounted on to of our Turtlebot. On the first Figure the robot moves always to the end of the corridor and then returns before moving to the next corridor. On the second Figure the Turtlebot moves through the corridors by reaching the end of one corridor and then moving to the next crossing it the opposite direction.

The approach followed in this work is presented in the Figure 1. The reason why the first approach is more appropriate is because we are interested the robot to traverse the corridors always in the same direction. As described we are using two RFID antennas

mounted on top of the Turtlebot. Also RFID antennas scan(emit RF waves) only when the robot moves for the first time in each corridor so the scan is performed only in one direction and the antennas are deactivated when the robot returns back. This approach is followed because it is important to know exactly the side where each RFID antenna detected each RFID tag.

On the second Figure is another approach in this case we should keep track each time the robot change direction to keep the knowledge about robots direction to acquire knowledge about the side of the detected RFID tags.

Another solution could implement a completely different approach. In any case it is important to keep in mind that the information about the side of each corridor where RFID tags were detected is important for the final result. So another approach could cover both cases by implementing a dynamic algorithm with the ability to detect the exact direction of robot movement and determine the side of each RFID antenna.

The final choice is the movement described first because as it already mentioned we want to scan each side of warehouse's corridors with a specific RFID antenna. So the left RFID antenna always scans the left side of the corridor and the right RFID antenna scans always the right side of the corridor. In any other case the RFID antennas should be deactivated or paused so no RFID scan is performed. This approach is the simplest for getting the information on which side of the corridor each RFID tag was detected.

6.1.1 Description of the Algorithm

For the implementation of the navigation algorithm ROS packages where used. ROS provides most of necessary packages needed for implementing such behaviors. In this subchapter is presented the general algorithm created for ROS robots and specifically in our case Turtlebot navigation. The implemented algorithm should meet the next goals:

1. Avoiding obstacles. (implemented by ROS in move_base package)
2. Setting parameter time limit for reaching current goal – position.
3. If goal is not reached within the time limit set goal as failed.
4. Setting parameter number for repeating waiting time if robot is still moving.
5. If robot is still moving wait again for time limit duration.
6. Setting parameter retry limit for reaching current goal – position.
7. If current goal failed retry for number of times.
8. Move to the next goal-position in case of failure or success reaching the current goal-position.

9. Keep a list of successfully reached positions.
10. Setting parameter for radius
11. In case if an obstacle appears and the robot avoids it check if the robot passed by any other possible goal position within a preset radius.
12. Robot has to move to the positions exactly in the predefined order and not in reverse.
13. Mission of the predefined positions – goals is defined in a JSON file.
14. Scan for RFID tags only while passing for the first time from a warehouse corridor (always at the same direction).
15. Dynamically detecting the side on which RFID antenna is scanning. (future)
16. Stop scanning for RFID tags if already passed from that position in the past.(future)
17. Publish current goal (starting position, current position, destination position)
18. Collect Data from RFID tags
19. Publish Data from RFID tags
20. If the position of the goal is not reachable, Turtlebot stops the movement and retries to complete the goal again for number of times specified in the parameters
21. If Turtlebot fails with the current goal it continues with the next goal in the JSON file. The current goal is considered as failed.
22. In case that the Turtlebot passes near a goal that isn't completed, and the distance from this goals is less than the predefined radius the goal is considered as successfully completed.
23. Starts an OdomService that provides the Odometry data that contains the position of the Turtlebot in the world to the Clients.

6.2. Steps performed by the navigation algorithm

Here is presented the general architecture of the navigation algorithm which was used for navigation of Turtlebot. Although because it uses ROS it can be used on any robot that supports ROS.

Parameters:

1) goal wait duration : Parameter for the maximum waiting time duration for the current

goal to complete (in seconds)

2) goal repeat wait max : Parameter for the number of repetitions of waitings for the goal to complete if robot is still moving

3) goal re-execution max limit : Parameter for the number of maximum re-executions of the current goal in case of failure or exhaustion of waiting time

4) radius : Parameter for the radius around the goal. If robot passes from a position inside this radius goal count as success

Algorithm:

1 Read all predefined goals from the JSON file which form a path.

2 During the robot navigation perform validation of completed goals within the “radius”.

3 For each goal (position and quaternion) on the path.

3.1 Construct goal to pass to the robot.

3.2 Repeat for “goal re-execution max limit” times if goal not completed successfully

3.2.1 Send constructed goal to Action Server to be executed by the robot.

3.2.2 Wait for the robot to finish the execution of the goal for duration specified “goal wait duration”

3.2.3 If the robot is still executing the goal wait again for duration specified in “goal wait duration” for “goal repeat wait max” times.

3.2.4 - If goal finished within the allocated time with goal status “SUCCEEDED” move to the next goal, set goal as completed and move to the next goal.

3.2.5 - Else cancel goal

4 Re-validate if any of passed positions are within the radius of any of goals in the path.

5 Print successfully completed goals.

Also except of the previous steps described a simple ROS service is implemented for getting the Turtlebot Odometry data when it is needed. The purpose of this ROS service is

when a RFID tag is detected the position of the Turtlebot could be acquired.

For the implementation of the algorithm was used ROS move_base package.

1. ROS “actionlib” package for the creation of a SimpleActionClient of MoveBaseAction type used to send goals to the Action Server, wait for the goal to be executed, and get state of the goal.
2. ROS “Pose” geometry message type, which is composed of “Point” which is the position on the map and the “Quaternion” which is the angle.
3. “move_base” package contains all the necessary functionalities for the navigation of the robot.
4. Subscription to ROS “odom” topic for receiving the odometry of Turtlebot
5. ROS parameters for setting parameters
6. ROS service for providing Odometry data.

The created algorithm for Turtlebot Navigation was tested in a simulated environment. An indoor warehouse like environment was build with Gazebo tool to simulate a real world warehouse.

6.3. Schemes, Pictures of the algorithm, RVIZ and Gazebo

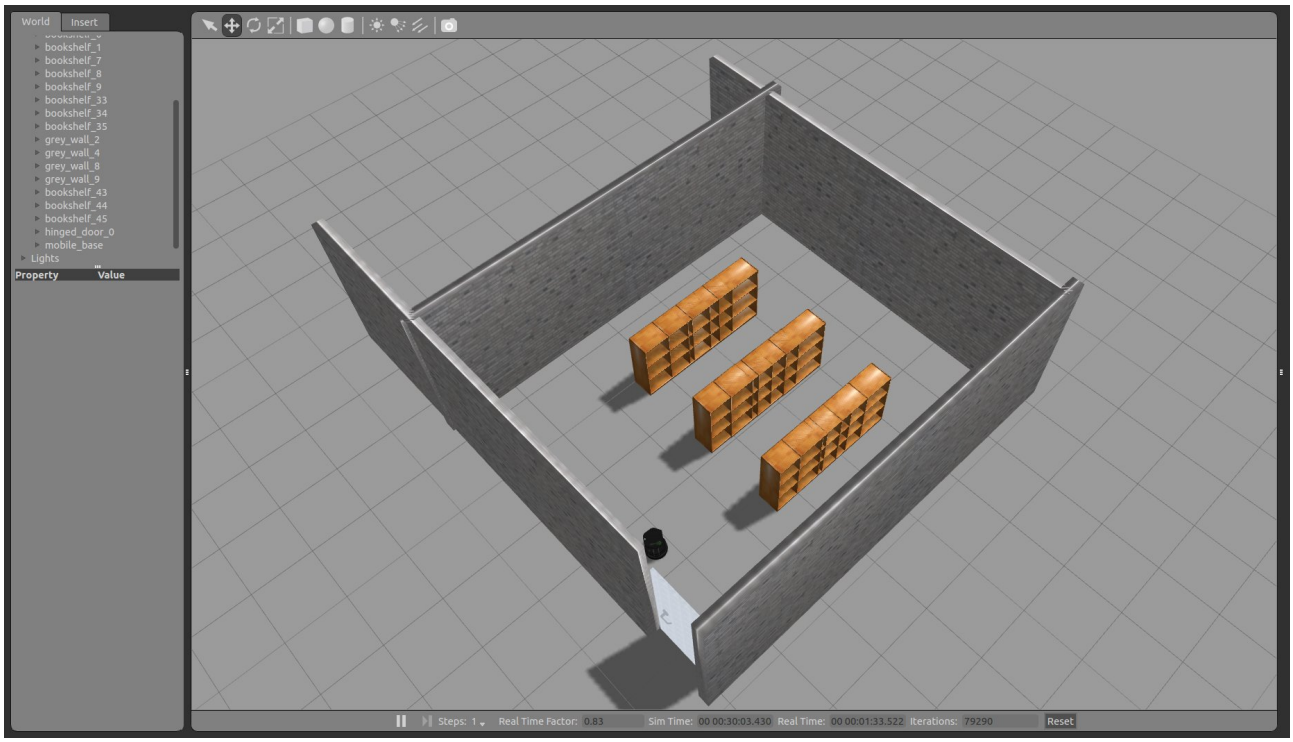

Figure 25: Gazebo simulated world of a warehouse, Turtlebot is visible at bottom corner of the room

After creating this simulated warehouse world in Gazebo it was possible to create a map of this warehouse with RVIZ tool. After creating and saving the warehouse map it is possible to navigate in the warehouse by using the created map and set goals to Turtlebot through the created algorithm. Next are the enumerated the steps that should be performed before running a simulation experiment:

1. Create your own world in Gazebo
2. Use RVIZ, gmapping, map_server, turtlebot_teleop to create a map of the simulated world.
3. In RVIZ with Publish Point functionality find all the positions-goals of the path you want navigation algorithm to send to Turtlebot.
4. Create a JSON file with the points for the algorithm to consume.
5. Run algorithm for Turtlebot navigation.
6. During the Navigation RFID antennas are detecting RFID tags places on goods in the warehouse (in the simulation simulated RFID tags are produced).
7. While Turtlebot is navigating towards set goals, messages about current goal and Turtlebot position are publish to ROS topic.

The created map of the simulated warehouse is presented in the next pictures

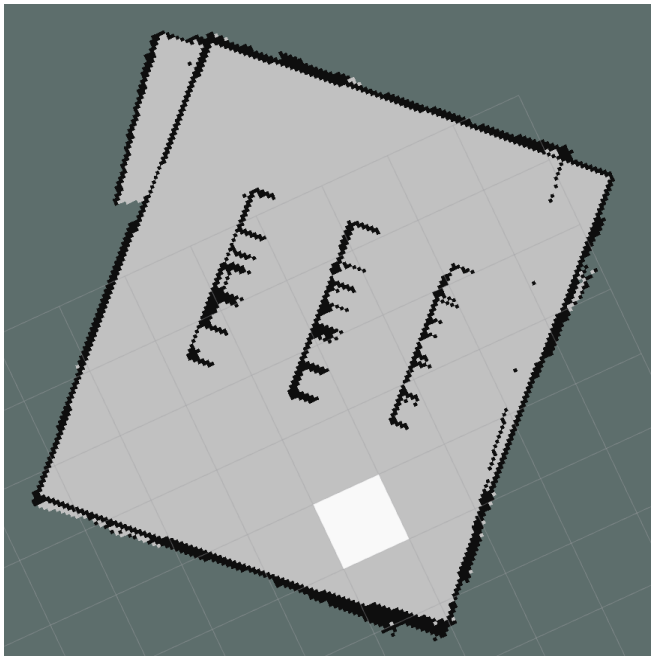

Figure 26: PNG picture of the created map representing the GAZEBO simulated warehouse

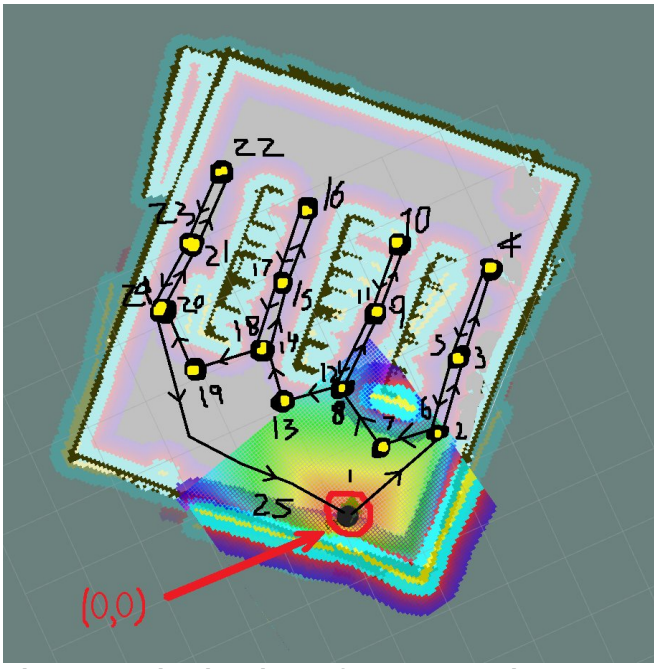

Figure 27: Simulated map from RVIZ at the moment when robot completed the navigation, robot returned to (0,0) position.

In case that an obstacle appears in front of the Turtlebot it is automatically detected the help of mounted lidar RPLIDAR A2, the created point cloud and the local planner who computes a new path to over come the obstacle.

In case that an obstacle appears on top of the position which is the current goal sent to Turtlebot, Turtlebot will stop moving and will wait until the obstacle is removed or until it the waiting time or retry limit are reached and the goal is considered as not completed or failed.

After completing or failing a goal a new goal is sent to the Turtlebot. After completing all of the goals Turtlebot stops at the last sent goal in our case it is the position (0,0) as viewed on the picture.

In the Figure 6 are presented the goals with black-yellow dots and the expected movement that the Turtlebot will perform. This is the movement that we expect from Turtlebot when we set this positions. Unfortunately because the path planners packages are already implemented by ROS and we can modify only several parameters sometimes this movement from one point to the next could be completely different. This can happen if an obstacle appears in front to the robot. The only way to overcome this problem is to implement a completely new path planner for navigation that will meet all the required functionalities.

6.4. RFID Antennas activation and deactivation

As presented in the pictures of the Turtlebot movement in the warehouse the next assumptions are obvious:

1. Turtlebot passes through the same position twice.
2. Turtlebot passes from position where no RFIDs exist.
3. Also we want the Turtlebot to perform RFID scan only when it is moving to specific direction.(to determine the side where a specific RFID is detected).

This is the reason why antennas should be deactivated(paused) or activated(resumed) at specific position. This can be performed through the command provided by the RFID reader/module which is presented in the next chapters.

7. RFID Technology

[77]RFID technology exists for a quit long time, it exploits the electromagnetic fields to identify and keep track of objects with attached special tags, Its origins date back to the WWII. However only the last years it become widely used and covered many fields of our society. Generally the fields where RFID tags could be used are Commerce , Retail, Access Control, Advertising, Entertainment, Promotion tracking, Transportation and Logistics, Passports, Infrastructure management and protection, Transportation payments, Animal identification, Healthcare, Libraries, Museums, Schools and Universities, Sports etc.

RFID uses electromagnetic fields, through this field the automatic identification and tracking of the RFID tags attached to objects is possible. The main three parts of composing this technology are RFID reader, RFID antennas, and RFID tags also some kind of software is also needed. There are two different types of RFID tags the active and passive RFID tags. The object that has attached on it a passive RFID tag is identified and distinguished by the RFID tag's contents, these tags contain unique information written on them. To collect the information from the passive RFID tags a RFID reader/antennas are emitting radio waves(electromagnetic fields), RFID tags collect this emitted energy and automatically produce responses. On the other hand active RFID tags periodically transmit ID signals. Finally the main task of RFID readers/antennas is to emitting signals to the tags and read the responses produced.

Also health issues and potential harm are important area and has to be considered as well this work is not examining such issues and there are several reports about possible health dangers[78] [79].

The Regulation and the standardizations of this technology is done by several organizations which are the International Organization for Standardization(ISO), the International Electrotechnical Commission (IEC), ASTM International, DASH 7 Alliance and EPCgloabl.

[80]Reasons of choosing this technology:

1. Well known and publicly accessibly
2. Mature enough, it exists for quite a long time.

77 https://en.wikipedia.org/wiki/Radio-frequency_identification

78 https://www.rfidjournal.com/blogs/experts/entry?5001

79 https://www.inria.fr/en/centre/lille/news/is-rfid-dangerous?mediego_ruuid=7ca6c172-ec94-400d-8952-5b272eec844d_1

80 http://morailogistics.com/9-facts-about-rfid-technology-in-logistics/, https://cybra.com/30-amazing-facts-about-rfid-technology/, and https://rmsomega.com/the-top-5-reasons-for-using-rfid-in-the-warehouse/

3. Small size of RFID tags, attachable on any surface.
4. Fast bootstrap , doesn't require much time for learning
5. Holds a great market share which is continuously growing.
6. It is possible to use RFID tags in combination with barcode technology.
7. RFID technology is used in many applications world wide.
   (a) Access cards, and wristbands with integrated RFID technology.
   (b) RFID chip charged with money used in public transportation in big cities like Hong Kong,New York, Singapore, Moscow and Rio de Janeiro.
   (c) Vehicles/cars get benefited from RFID tags for quality control, and quality assurance.
   (d) Objects and products are identified and tracked by the tags when they are transported from place to place etc.
   (e) Animals tracking and identification
   (f) Libraries with more than thirty million library items worldwide hold an RFID tag.
   (g) Vatican is keeping track of more than 2 million ancient manuscripts
   (h) Sports for tracking the athletes
   (i) Waste tracking RFID to track for each household its waste
   (j) etc.

[81]There are probably more technologies that could be used for the detection and recording of warehouse objects. Technologies like Barcord(small distance, durability), NFC(small distance, high price)[82] or even Bluetooth(price) are all possible solutions. Although RFID approach has proved its superiority for multiple reasons some of them are presented in the previous paragraph. Other reason for choosing RFID technology are acceptability, affordability, higher distance coverage, higher energy consumption, higher data rates, durability etc.

7.1. RFID Antennas

[83]There are many types of Antennas which are used for different types of applications. In

81 https://nfc-forum.org/nfc-bluetooth-and-rfid-unraveling-the-wireless-connections/

82 https://blog.atlasrfidstore.com/rfid-vs-nfc

83 More technical information about these specific antennas could be found on the official website of Kathrein https://www.kathrein-solutions.com/products/hardware/rfid-antennas/wide-range-70-antennas.

our project we used a Wide Range UHF RFID Antennas 70 degrees 865 – 868MHz. (UHF – Ultra High Frequency)

Figure 28: Kathrein Wide Range 70 degrees Antennas, source: https://www.kathrein-solutions.com/products/hardware/rfid-antennas/wide-range-70-antennas

RFID antennas act as a transmitter and receiver. Antennas can operate only with the help of RFID reader. Antennas at the same time can act both (a) as a signal transmitter and (b) as a signal receiver. RFID Antennas transmit radio waves, after the emitted radio waves are collected by passive tags, so passive tags immediately respond so Antennas an receive the RFID tag signal. The main two different types of antennas are linear and circular polarized antennas. The readers range also vary there are short-range (<=30cm) and long-range antennas(>30cm). On long-range antennas the presence of dielectrics(electrical insulators) between the reader and tags can have a negative effect on the signal quality.

Finally although we are using circular polarized antennas there are also linear polarized antennas out there and the right choice can have a great impact on the final results. The main difference is that Linear polarized need predefined RFID tag orientation and the tag must be fixed at the same level with the antenna. The range of linear antennas is bigger that that of circular polarized antennas due to the concentration of the emission. On the other side Circular polarized antennas, such the antennas that we are using in our project, has emission of electromagnetic fields that look more like a corkscrew, so the electromagnetic waves are covering two planes in opposed to one plane of linear antennas. On each wavelength one complete revolution is performed. In our case since we are not sure and we can't the RFID tag orientation the right choice are circular polarized

antennas.

7.2. RFID Reader/Module

[84] [85]RFID reader/antennas main task is to read tags positioned in the space. RFID readers is like convert the electrical current into electromagnetic waves which are then transmitted in the surrounding space where the tags collect the transmitted waves and use them as energy by converting them back to electrical current.

There are several types of RFID reader (a) Passive Reader Active Tag (PRAT) system, (b) Active Reader Passive Tag (ARPT) system, and (c) Active Reader Active Tag (ARAT) system. In our case we use Active Reader Passive Tag (ARPT) system approach in this case an active reader emits interrogator signals and then receives from the passive tags the authentication responses.

Figure 29: MTI RFID RF Reader/Module, source: https://www.mtigroup.com/upfiles/e_pro_tb01332508202.pdf

7.3. MTI RFID Reader

[86]In this work MTI UHF RFID Reader/Module model RU-861-010 developed by Microelectronics technology inc.[87] was used. MTI Reader/Module is connected to computer

84 https://www.impinj.com/about-rfid/how-does-rfid-work/

85 https://en.wikipedia.org/wiki/Radio-frequency_identification

86 https://github.com/mti-rfid/RFID_Explorer/blob/master/MTI%20RU-824%20RFID%20Module%20Command%20Reference%20Manual%20v3.3.pdf

87 http://www.mtigroup.com/upfiles/e_pro_tb01332508202.pdf

though USB interface and it is installed as a HID(Human Interface Device). The host processor uses the USB VID and PID combination to find the MTI RFID reader/module. The USB VID and PID numbers are both as follows: **USB Vendor ID: 0x24E9 / Product ID: 0x0824.**

The configuration of the MTI RFID Reader/Module to make it operational could be performed through the host. For the configuration of the MTI RFID Reader/Module a application could be downloaded from the github [88] with a graphical interface for Windows, also all the necessary documentation is included. The MTI RFID reader/module is compatible with ISO 180006C for this reason a command set is provided for programming commands. The following ISO 18000-C tag-protocol operations are reachable though the provided command set: Inventory, Read, Write, Kill, Lock, Block Write, Block Erase.

The MTI Mac firmware provides report packets for presenting tag-protocol operation response data to the host. The command set supports the configuration and control of the individual antenna ports on the RFID reader/module. MTI Reader/module has the capability to perform operation of read and write with two independent external antenna ports. The maximum output is +30dBm for 300 power level and maximum range of antennas is 12 meters. Also host has access to OEM configuration data area (store and retrieve the specific hardware configuration and capabilities of the RFID reader/module) on the RFID reader/module is provided by the command. Also low-level control of the RFID reader/module's MAC firmware is supported by the command set. From the low-level control it is possible to perform software reset operation, reseting the RFID reader/module to a default idle state, pass to low power stand-by mode, and perform in-the field upgrade.

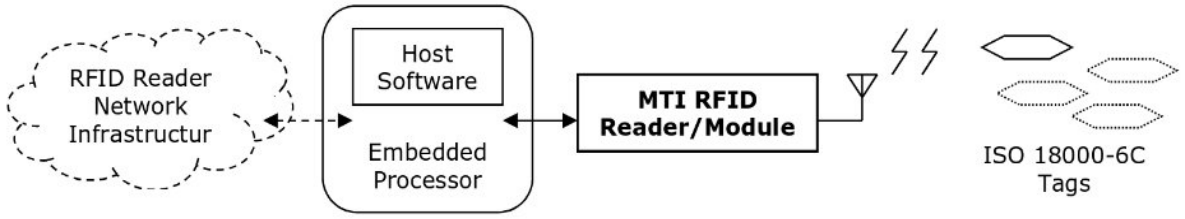

Figure 30: Typical MTI RFID Reader/Module Application Architecture, source: MTI RU-824 RFID Reader/Module Command Reference Manual Version 3.3, https://github.com/mti-rfid/RFID_Explorer/blob/master/MTI%20RU-824%20RFID%20Module%20Command%20Reference%20Manual%20v3.3.pdf

From the Command Set available generally the following operations can be performed: RFID Reader/Module Configuration, Antenna Port Configuration, ISO 18000-6C Tag-Select Operation, ISO 18000-6C Tag-Access Parameters, ISO 18000-6C Tag-Protocol Operation, RFID Reader/Module Control Operation, RFID Reader/Module Firmware Access, and

88 https://github.com/mti-rfid/RFID_Explorer

RFID Reader/Module Region Test Support. The command set consists of hexadecimal values which are precisely described in the official manual of MTI Reader/Module. The length of the command and response packets are fixed at 16 bytes length. There are 3 different formats of messages. The Command Packet Format, the Response Packet Format and the Report Packet Format.

7.4. RFID Tags

[89]RFID Tags are Radio Frequency Identification Tags, specifically RFID tag is an electronic tag which is used for assigning IDs for the identifications and tracking purposes. For the operation of RFID tags radio frequencies are required. The architecture of the tags include a chip used for precessing data, a memory module used for string the data and an antenna for receiving/sending radio frequency waves. RFID tags can be of two different types they can be active or passive. The active tags need an embedded power source to periodically transmit its ID signal, in contrast passive tags operate without a power source by collecting emitted energy from the RFID reader/antenna module. The passive tags to operate normally need to be illuminated with a power level approximately 1000 times stronger than than for the signal transmission. The active tags contain a memory module and have longer range comparing to the passive tags which use all the electric current for the transmission their ID number. The frequencies in which the RFID tags operate can be different depending on the operation frequencies of the whole system which can be Ultra high frequencies - UHF, High frequencies - HF or Low frequencies – LF. Also RFID tags come in multiple forms which depend on the application where they are used. RFID tags can take the almost any form most common forms are plastic card, thin sticker, or a small chip, etc. that why RFID tags have unlimited possibilities and uses, they can be used in almost every area for tracking and identification because of they variable forms and their durability. Finally RFID tags are a superior way for identification compared to the barcode technology but both technologies can be used together.

Figure 32: RFID tags used in the project

Figure 31: Different types of RFID tags, source: https://learn.sparkfun.com/tutorials/rfid-basics

89 https://www.techopedia.com/definition/24273/radio-frequency-identification-tag-rfid-tag
https://internetofthingsagenda.techtarget.com/definition/RFID-tagging
https://en.wikipedia.org/wiki/Radio-frequency_identification#Tags
https://learn.sparkfun.com/tutorials/rfid-basics

7.5. Configuration of RFID Antennas and Reader

Before mounting all the parts of the RFID technology (antennas, reader, and tags) on top of Turtlebot it was important to examine the configuration of different parameters of RFID reader like power levels, decibels, dwell time, and the number of inventory cycles. Also it was necessary to examine different types of RFID tags in relation with the previous parameters.

The trials of the MTI RFID reader we used can be performed on a Windows machine though the MTI graphical interface which provides all the necessary tools for experimentation to any user even a beginner with little knowledge about RFID technology. Though this tools it is possible to play for example with the power level values (1/10dBm) to select the proper power level for a specific application.

The main parameters that are available for configuration according the RFID antennas are:

1. Enabled/Disabled Status: Determines if an antenna post will be used for the tag-protocol operations.
2. **Power level(1/10dBm)**: This is antennas transmission Power in 1/10dBm increments. It indicates the power supplied to the transmit antenna port when it is being used.
3. **Dwell Time**: This value indicates the maximum number of milliseconds spent on each logical antenna during a single cycle during a tag-protocol operation cycle.
4. **Number of inventory cycles:** This value indicates the maximum number of inventory cycles are spent on a logical antenna before switching to the next available antenna.
5. Physical Port: This value specifies the physical port to which the logical antenna is bound for transmission of data.
6. Operation mode(Continuous or non continuous): Continuous options is for running inventory continuously, and non-continuous option is for running a single inventory cycle.

* Only one of dwell time and inventory cycles can be set to zero value at the same time. Also dwell time, number of inventory rounds and the RF power are configured on a per-antenna-port basis.

The main commands we are interested in are four:

1. Run inventory: Starts inventory and the rfid reader starts receiving tag values.

2. Stop: Stops permanently the inventory.
3. Pause: Pause the inventory until the resume command is sent.
4. Resume: Resumes the inventory if previously inventory was paused.

Also post singulation criteria and selection criteria could be specified but this options where not examined in this work because we are interested in receiving all possible tags without making any distinction, at least at this point. After the tags being collected it is possible to process them in any way we desire. Generally speaking RFID reader has many different available functionalities that someone can take advantage of. Other available configurations are “Select Criteria”, “Algorithm”, “Post Singulation”, “GPIO Pins”, etc.

Another Tool of MTI application is “RF Test”, it is useful for experimentation and for choosing the right values because through its panel it is possible to change the Power levels and select the most convenient for a specific use case. In our specific case this tool helped for choose the proper Power value in relation to the distance of the antenna from the RFID tag.

7.6. Measurements performed with RFID Antennas and Reader

The following table present several approximate measurements performed for selecting the desired Power Level values. The Power values tested were 100 – 10dBm, 150 – 15 dBm, 200 – 20 dBm, 250 – 25 dBm, and 300 – 30 dBm. For the measurements where used only passive RFID tags, and specifically a plastic card, a soft sticker square shaped, and a soft sticker long rectangle shaped. Next is presented a table with the approximate gotten values. This measurements were performed for the selection of the Power Level value for our specific experiment which is performed in a warehouse environment where Turtlebot is moving in the middle of a corridor , between of the warehouse shelves. In our case the corridor wasn't wider than 2/2.5 meters. A general rule is that larger RFID tags have better read range than smaller RFID tags that have shorter read range.

Table 2: Measurements for RFID tag detection with different type of RFID cards and different Power Levels

Power Levels	Plastic card	Square sticker	Rectangle sticker
100 - 10dBm	0 ~ +-1m	0 ~ +-0.7m	0 ~ +-0.5m
150 - 15dBm	0 ~ +-1.5m	0 ~ +-1/1.2m	0 ~ +-1/1.2m
200 - 20dBm	0 ~ +-2-2.3m	0 ~ +-1.2/1.5m	0 ~ +-1.2/1.5m
250 - 25dBm	0 ~ +-2.5m	0 ~ +-2m	0 ~ +-1.5/2m
300 - 30dBm	0 ~ +-4/4.5m	0 ~ +-3.5/4m	0 ~ +-3.5m

We are not interested in hundred percent precision from this measurements what we want is approximately to check the distances in contrast with the Power Level values and the

different Types of RFID tags we had at our disposal.

The right choice of the Power Levels is important for four main reasons:

1. Generally not loosing any RFID tags located around RFID antennas from both sides of the warehouse corridor shelves.
2. Avoid receiving RFID tags from alongside corridors to the corridor the Turtlebot is moving.
3. Receiving RFID tags if Turtlebot try to avoid and obstacle and moves not in the middle of the corridor but close to one of its sides.
4. Reduce the power consumption since RFID reader and antennas use power from a mobile device (laptop).
5. Avoiding emission of unnecessary radio frequency waves to the environment.

From the previous table as it is obvious the higher the Power levels the better the coverage of the RFID antennas. The higher the Power Level is set as it is expected the rates of RFID tags was higher. Also as it is obvious the hard plastic RFID tags had better results from the small sticker tags. Finally the experiment was performed at a corridor of approximate width of 2.5 meters the preferred choices were the Power Levels of 200 – 20dBm and 250 – 25dBm.

There are many factors that influence the reading distance and the quality of the RFID tags. Such factors are the materials, the position and the orientation of the RFID tags, the angles of the antennas, antenna polarization and other reader configurations.

7.7. Integrating RFID Technology with Turtlebot 2

An algorithm implemented in Python is used for configuring, running basic commands and running inventory is used. This python program is used for the communication with MTI RFID reader/module and the RFID tags collection. During the collection the RFID tags the data collected is saved or published with the help of ROS topics. Apache Kafka, ROS logging and simple file explained in next chapters. Also the fact that python is used makes it a lot easier for integrating with ROS since ROS supports python.

7.8. Mounting RFID Antennas on top of TurtleBot

RFID Antennas have to be mounted on top of Turtlebot. Both will be mounted on top panel of Turtlebot one on the right side for scanning the right side of the corridor and the other on the left side for scanning the left side of the corridor. Because Wide Range 70 degrees antennas are used they should and the robot is to close to the ground the antennas are mounted with a slit angle as shown in the picture. For exact angles and position of each antenna separate experimentation has to be performed considering different factors like

how high and how far the goods are stored from the Turtlebots antennas. The angle of Antennas is not a big problem (and it is easily adjustable) since we are using circular polarized antennas the plane and orientation of RFID tags in relation with the RFID reader/antennas doesn't affect the detection rate.

7.9. Other issues related to RFID technology - Problems and concerns about RFID

Fore implementation of a commercial system several more issues should be studied, and a more precise research and experimentation need to be performed about RFID technology.

7.9.1 Materials interference with RFID technology

[90]There are materials that reduce the RFID tag detectability by absorbing the emitted RF energy. In a warehouse this could be an serious issue for example if for example bottles of water(containers of liquids) are stacked next to each other it can lead to RF energy absorption and to low rate RFID tags detection. Any objects with high RF energy absorption level will reduce the emitted energy reducing the energy reached to the passive RFID tag and also the RFID antennas can be affected in a bad way.

1. **Objects containing high amount of water:** such objects will reduce the RF energy in the UHF (Ultra high frequency) by absorbing the RF energy. This will make it harder to read the UHF RFID tags that are behind such objects.

2. **Liquids:** Liquids generally are good RF energy absorbers. This can lead to non detected RFID tags. When scanning RFID tags that where behind a human body or a tank of water were not detected. Higher power lever could possible solve this problem. Also sometimes due to signal reflection these RFID tags where detected by the RFID module.

3. **Carbon** is another element thats absorbs UHF energy. In case that the boxes are covered in carbon it can possibly lead to bad RFID tag detectability.

4. Any material interfering with RF UHF can cause unexpected results to the performed experiments if not studied. Such material can interfere, block, absorb, or reflect the RF waves.

5. Other competitive frequencies(radio waves).

To overcome this problems tests have to be performed, the arrangement of the object

90
- http://www.rfidjournal.com/blogs/experts/entry?10691

- https://en.wikipedia.org/wiki/Electromagnetic_absorbers
- https://www.quora.com/What-materials-block-or-refract-radio-waves
- https://blog.atlasrfidstore.com/improve-rfid-read-range
- paper : http://www.av.it.pt/conftele2009/Papers/55.pdf
- paper - https://www.pmmi.org/sites/default/files/Proposed-Guidelines-for-the-Use-of-passive-RFID-Transponders.pdf

should be studied for lowest RF energy absorption amounts. Another approach is by isolating the RFID tags from the RF absorbers by inserting another material between them like metal foil or plastic RFID tags covered in metal. Also a solution are reflective surfaces. All these cases should be studied in case of implementation of a industrial product.

7.9.2 Multiple interposed object with RFID tags

Imagine a scenario where RFID tags are position on top of boxes in a warehouse and for some reasons there are three boxes each with its own RFID tag. This boxes are interfering with the RF signal and the power of the signal is reduced.

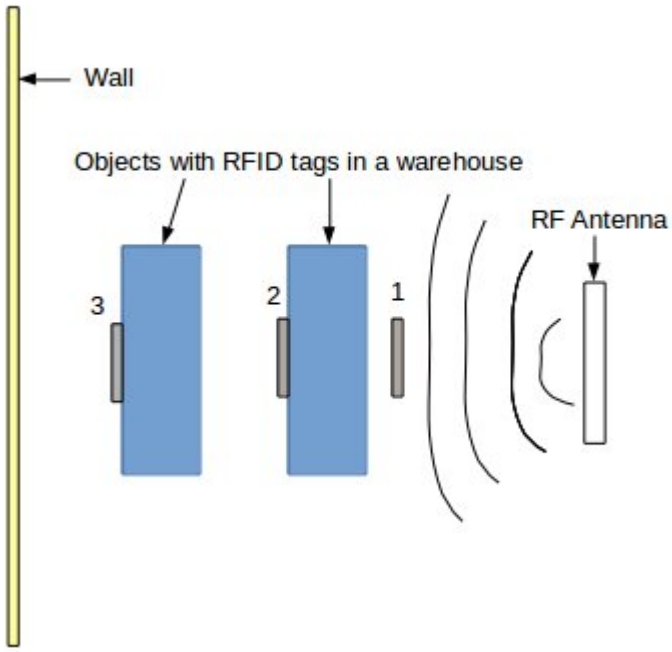

Figure 33: RFID tags placed between the objects of a warehouse, 1,2, and 3 are RFID tags.

As shown in the previous picture we have 2 boxes with RFID tags located on top of them. Also an RFID tag is placed in front of the boxes.

- RFID tag #1 (1m distance): This is the first RFID are no objects blocking the signal of RF Antenna. This RFID tag is always detected because it is close enough and no objects absorb the signal power from antenna.
- RFID tag #2 : As presented on the picture this RFID tag is placed on the left side of the right object, so the RFID tag is located between the two objects. In this case different behaviors were observed depending on the power of the signal, the distance and the reflection. In our case this RFID tag wasn't detected always because the signal was blocked by the right object and due to the close distance from the left object no reflection was possible. To overcome this problem this RFID tag could be placed on the other side(right side) of object or on top of the object (top side).

- RFID tag #3 : This RFID tag is visible to the RFID Antenna and it is detected as expected. Because of the reflection from the surrounding surfaces like walls etc. this RFID tag is detected normally. Although this RFID tag is detected it could be also placed on top of the object.

7.9.3 Interference of RFID reader/antennas on multiple autonomous robots

In the future work if it is planned to use more than one robots scanning on multiple corridors of a warehouse for products it is important to make a more precise research according the power levels , dBm and other parameters at which antennas are emitting. Although this is not considered as a problem since the detection of an RFID tag from one antennas doesn't exclude the detection of the same RFID tag from a another antenna.

7.9.4 Plane and orientation of RFID tags in relation with the RFID reader/antennas

Although in our case after experimentation we didn't have any issues with different orientations and planes because we are using circular polarized antennas it should be carefully chosen what type of antennas will be used Linear of Circular polarized antennas. Linear polarized antennas need a predefined orientation of the RFID tags and also the RFID and the antennas has to be placed at the same plane.

7.9.5 Optimal Antenna and Reader Selection

There are unlimited option for selecting the perfect RFID antenna for a specific application. One of such parameters was in our case is the circular polarization of the antenna. Another parameter is the frequency that the antenna emits there are LF, HF, and UHF.

7.9.6 Privacy – Security

[91]There are concerns according of the RFID tags security recently raised. For example unauthorized reading of RFID tags can be a risk to RFID privacy. Also RFID tags can be vulnerable to different types of “attacks” thats why techniques and protocols for making the RFID tags more secure should be applied. This problems are not studied in this work.

7.9.7 RFID Health concerns

[92]As with all technologies emitting electromagnetic waves and radio frequencies, RFID technology they can have possibly some impact on the exposed human whose body absorbs RF energy, so precautions should be taken like the distance of antennas from the researcher during the test. This subject is not part of this work and should be studied

91 https://en.wikipedia.org/wiki/Radio-frequency_identification

92 https://www.inria.fr/en/centre/lille/news/is-rfid-dangerous?mediego_ruuid=645197fd-2389-4d6d-8ef4-af5f14059c42_0

https://www.researchgate.net/publication/224328848_Impacts_of_RF_radiation_on_the_human_body_in_a_passive_RFID_environment
http://www.rfidjournal.com/site/faqs#Anchor-Are-63368

further. There are many other studies already performed that can enlighten all the concerns according the possible health impact.

8. LIDAR Technology

[93]LIDAR is an acronym for Light Detection and Ranging. LIDAR is a surveying method that uses a pulsed light laser in combination with a sensor that measures the reflected pulses of light to compute the distances to the scanned surface. The purpose of LIDAR is to map the environment that it scans. The scanned surface differs according to the application where LIDAR is used. It can be room's walls or obstacles, Earth's surface, forest or the bottom of an ocean. The range/distance data collected also has to be combined with other data like position, orientation, GPS, scan angles, and calibration data to get the final result. The combination of all the data collected LIDAR reconstructs and generates a representation of the environment scanned, this representation could vary from 2D to 3D. The image produced by LIDAR is a precise point cloud. This point cloud is a group of elevation points of high precision. LIDARS can be used from ground, air of even space.

LIDAR systems use different types of wavelengths depending on the type of the target surface it is scanning. The wavelength vary from 10micrometers to 250nm(UV). LIDAR usually are consisted of a Laser, a Scanner, and a Navigation system.

Finally LIDAR has many uses some of them are commonly known like high-resolution maps and autonomous cars which are quit popular lately. LIDAR has two types of applications which are airborne and terrestrial which are applied in practice in many areas like Agriculture, Archeology, Geography, Forestry, Transportation, Autonomous vehicles, Geology, Robotics, etc.

8.1. RPLIDAR A2

[94]In this master thesis Laser Range Scanner RPLIDAR A2[95] is used which is a Lidar similar to what was explained in previous chapter. RPLIDAR is an affordable comparing with other systems 2D LIDAR solution. The team that developed RPLIDAR is called RoboPeak Team, SlamTec company. RPLIDAR produce output which is suitable for map building, slam and 3D object/environment model construction. The output generated by RPLIDAR is appropriate for creating maps, performing slam, and building 3D models. Next are listed some of RPLIDAR A2 general specifications[96] [97] [98]:

93 https://en.wikipedia.org/wiki/Lidar
https://oceanservice.noaa.gov/facts/lidar
https://www.youtube.com/watch?v=EYbhNSUnIdU

94 http://bucket.download.slamtec.com/a7a9b856b9f8e57aad717da50a2878d5d021e85f/LM204_SLAMTEC_rplidarkit_usermanual_A2M4_v1.1_en.pdf

95 Official RPLIDAR website: http://www.slamtec.com/en.

96 Measurement Performance of RPLIDAR for Typical values : http://www.slamtec.com/en/Lidar/A2Spec

97 http://www.slamtec.com/en/Lidar/A2

98 https://www.robotshop.com/community/blog/show/unboxing-rplidar-a2-360deg-laser-scanner

- Range radius: 0.14m - 12m/18m (6 meters on other sources)
- Ultra thin: 4cm
- Sample rate: 2000-8000 times (measurement frequency) (typical - 4000)
- Scan Rate: 5 – 15Hz (scan frequency) (typical 10Hz)
- Angular resolution: 0.45 – 1.35 (typical 0.9)
- Work of years: 5
- Low noise brushless motor
- Wireless power, and optical communication technology.
- Light Source: Low power infrared laser light.
- Laser Safety Standard: Class 1 (safety to human and pets)
- Laser triangulation ranging principle, and high-speed RPVision range engine measures distance data 8000 times per second and has excellent performance in a long distance.
- It performs clockwise 360 degree omnidirectional laser range scan.
- ROS packages “rplicar_ros and rplicar_python” are available for RPLIDAR A2.

Figure 34: RPLIDAR A2

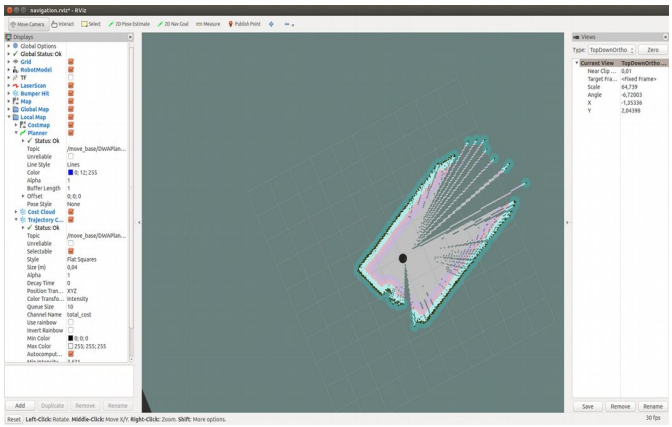

Figure 35: Room scanned with RPLIDAR A2 mounted on top of Turtlebot represented in RVIZ tool

8.2. RPLIDAR versus Kinect

[99]Kinect sensor is the default 3D sensor that comes with the included hardware of the Turtlebot robot. Although both sensors use infrared light the way this two sensors work is

https://www.roscomponents.com/en/lidar-laser-scanner/155-rplidar-a2.html

different. Kinect doesn't contain any mechanical parts except of a tilt motor so the movement it performs is totally dependent of the robot's movement, on the other side RPLIDAR spins 360 degrees around itself with the help of a brushless motor and can reach 8000 sample rate. This rotation of the RPLIDAR is important because it eliminates the need of the robot to perform the “recovery rotation” for clearing the space around it. The fact that RPLIDAR performs this rotation gives to us much more data about the surrounding environment since it collects data from all possible directions. For the Kinect to produce the same result robot has to perform a full rotation. Also as described in RPLIDAR characteristics the range radius of the RPLIDAR is much higher approximately 14 meters comparing to Kinect's range of 4 meters.

Specifically in our case the “recovery rotation” and generally the rotations performed by the Turtlebot when Kinect is being used can spoil the results of the RFID Antennas mounted on top of it. This happens because sometimes Turtlebot has to perform “recovery rotation” to clear the space or estimate it's position in the world. Each RFID antennas has to collect RFID tags data only from one side of the corridor to avoid mixing data about each RFID tag's side and produce accurate data about the side of the corridor on which each RFID tag is located. This is possible with the help of RPLIDAR because by performing a clockwise 360 degree omnidirectional laser range scan it eliminates the need for “recovery rotation”.

Finally except from previously described reasons why RPLIDAR is chosen, both sensors RPLIDAR and Kinect produce similar results not taking in account the areas that are not visible by the Kinect. Also as it is expected due to its superior characteristics RPLIDAR has higher overall results compared to Kinect's. The negative aspect about RPLIDAR is the set up and mounting of the sensor compared to Kinect's out of the box functionality. However all the necessary ROS packages for ROS – RPLIDAR interoperability are available.

Finally it is probably possible to use both technologies for even better result. Kinect sensor could be located closer to the ground and detect obstacles in front of the Turtlebot, while the RPLIDAR sensor could be placed on higher position scanning the area around the Turtlebot providing a more precise and accurate data, and at the same time preventing Turtlebot from unnecessary rotations.

99 Kinect https://msdn.microsoft.com/en-us/library/jj131033.aspx
https://msdn.microsoft.com/en-us/library/hh438998.aspx

Figure 36: RPLIDAR is mounted on the top plate of Turtlebot, Kinect is located under Rplidar.

Figure 37: Closer look of RPLIDAR and Kinect mounted on Turtlebot

8.3. Integrating RPLIDAR A2 with Turtlebot 2

Generally speaking the procedure of integrating RPLIDAR A2 with Turtlebot 2 is pretty straight forward. As already described ROS is organized in packages which which main aim is to provide necessary functionality simple as possible out of the box approach. Fortunately a research and development team called RoboPeck founded in 2009 has taken care of this issue by providing as with the implementation of the RPLIDAR ROS package. **The RPLIDAR package is available** both by downloading it from the “robopeak” github [100] and building it in a catkin workspace, or by installing it directly from the package repository by running the “sudo apt-get install <package_name>” command in the terminal. RPLIDAR packages support Hydro, Indigo, Jade and Kinetic ROS distributions.

After completing the package installation the next important step is the selection of the **appropriate position on to of Turtlebot.** There are many different position of possible RPLIDAR on top of Turtlebot. Next are described different possible mounting position.

1. Mounting RPLIDAR directly on top plate of Turtlebot. This is a peaty straight forward approach. This position is not bad in case that there are not other devices mounted on Turtlebot's top plate.

100 https://github.com/robopeak/rplidar_ros

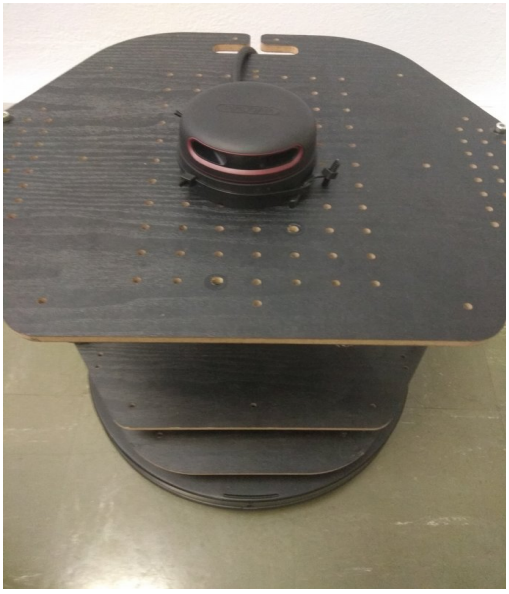

Figure 38: RPLidar mounted on top of the top plate of Turtlebot

Figure 39: Close view of RPLidar mounted on top of the top plate of Turtlebot

2. The second possible position for mounting RPLIDAR is again relative to Turtlebot's top plate but in this case mounting RPLIDAR exactly under top pate turned exactly 180 degrees around y axes. In this case RPLIDAR is positioned upside down mounted on top plate with its bottom side, exactly above the Kinect sensor. This approach after tests performed appeared to be reliable and the results were satisfying. The map produced with RPLIDAR mounted in this position was accurate. The reasons why this approach is attractive are first of all because there are no other devices mounted around RPLIDAR preventing it from scanning successfully the surrounding area and at the same time because it is positioned somewhere in the middle not close but at the same time not far from the ground, approximately at the same level with Kinect.

Figure 40: RPLIDAR mounted on under the top plate of Turtlebot

3. Another possible position for mounting RPLIDAR on top of Turtlebot was again above the Turtlebot's top plate, but also above the RFID antennas placed also on the top plate of Turtlebot. This is position is being examined because first of all we don't want to cover any side of RPLIDAR and prevent it from loosing data of the surrounding area, but also what we want is to mount RPLIDAR as close as possible to the ground to locate all the obstacles of any sizes which possible could block TurtleBot smooth movement.

4. Finally RPLIDAR could be placed on top of middle plate or on top of bottom plate. Both of these approaches are attractive because the RPLIDAR is places somewhere in the middle of Turtlebot, so it is covering both larger and smaller object as much as possible.

In a project presented online with a previous model of RPLIDAR A1 it is mounted approximately as we describe in the second approach, they place it under the top plate above the Kinect sensor and the results they presented were quit promising.

Regardless of the approach chosen there two are the main things to consider when mounting any RPLIDAR like sensors on top of Turtlebot.

1. First of all the spinning RPLIDAR should not be covered from any side with other possibly mounted devices on top of Turtlebot. In case where even a small device is mounted somewhere around RPLIDAR, it preventing the the laser from scan and sensor to receive effectively and the produced result will be defective.

2. The distance from the ground or in other words the displacement of RPLIDAR on y-axes (also on x-axes and z-axes). Mounting RPLIDAR close to the ground will probably prevent it from detecting higher objects like chairs, while mounting RPLIDAR to high above the ground will prevent it from detecting smaller objects like bags or smaller boxes.

After mounting RPLIDAR at the desired position on top of Turtlebot we still need to perform several adjustments to get the desired result. As explained in previous chapters when a sensor is mounted on top of the Turtlebot or any other robot we need to define a transformation to keep the relationship between coordinate frames at any point in time, in this specific case we have to define a “tf” of type “static transform publisher”. By defining this transformation the exact position of the RPLIDAR sensor relative to its parent plate which is the top plate link, ROS will always know where exactly RPLIDAR is placed in the space relative to the top plate and implicitly to the base link of the robot. In simple words by making this configuration of the “tf” transform will produce a translation offset that makes a relation between the “base link” and the RPLIDAR laser sensor. By adding this transformation we know where the RPLIDAR is placed in relation to the world frame, what is the pose of the RPLIDAR relative to the base link, and what is the current pose of the

base frame in the map frame.

In all cases the RPLIDAR was mounted approximately in the middle of Turtlebot plates. For example the following displacements are x = 0.015, y = 0.00, and z = -0.01 which are relative to the top plate for the second case examined where RPLIDAR is mounted upside down on the top plate. For the same case the angular displacement in radians was quaternion_x = 0, quaternion_y = 3.14159265, and quaternion_z = 0. This information is passed to Turtlebot's launch file as a “tf” transform of type “static_transform_publisher”, also in this transform is important to specify the parent link which is the plate “/plate_top_link” and child link RPLIDAR which is “/laser”

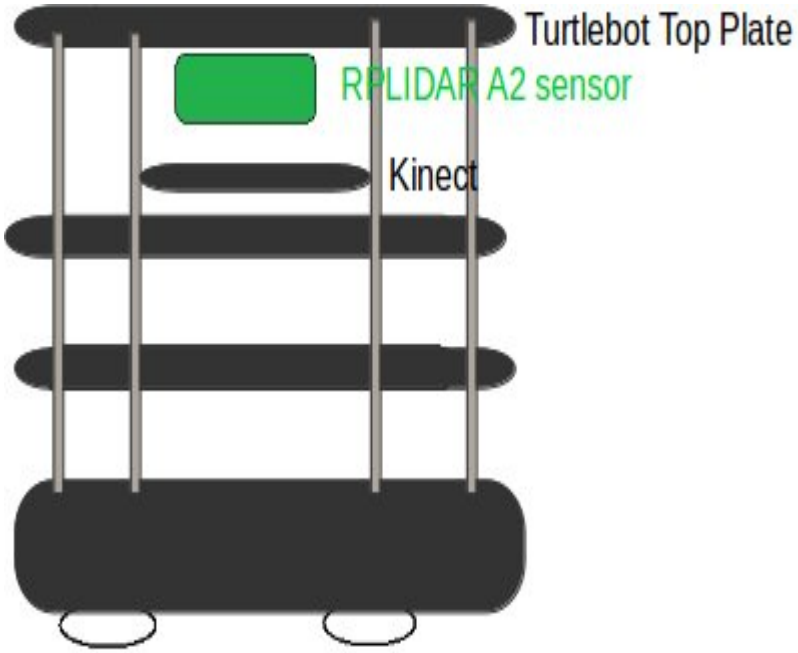

Figure 41: RPLIDAR is mounted under its parent plate "top plate" and turned 180 degrees y-axes

It looks like: <node pkg="tf" type="static_transform_publisher" name="laser" args="0.015 0.00 -0.01 0 3.14159265 0 /plate_top_link /laser 50"/>

8.4. Problems Detected RPLIDAR

Data from RPLIDAR is taken under consideration by Turtlebot only when it is moving, in other case when Turtlebot is static data from RPLIDAR is not received by Turtlebot, although RPLIDAR is spinning normally and collecting data about. This has multiple implications, for example when Turtlebot is stationary and a object falls from a shelve in front of it or when a person passes by, neither the box or the person are detected. This behavior can be observed with RVIZ. But since Turtlebot is always on the move this is a

minor problem, also we are interested about the object mostly when Turtlebot is moving to avoid collisions.

RPLIDAR is not detecting small object located on the ground. For example a lying cable that is several centimeters high is not detected. So Turtlebot will try to move towards that direction and only when it's movement is interrupted by this object Turtlebot will stop.

9. EDL

As already presented in previous work section, EDL stands for Experiment Description Language. EDL is a DSL - Domain Specific Language and it was developed as a main component of RAWFIE system which stands for Road-, Air- and Water-based Future Internet Experimentation. EDL provides as with terminology for defining experiments for mobile IoT in our case a robot called Turtlebot 2. EDL is simple and it is similar to XML or legacy programming languages. With the help of EDL we create a mission for our Turtlebot with the path of positions – goals which our robot Turtlebot has to follow. The created mission is in JSON file which includes a list of “waypoints” which represents the path of positions(goals) of an indoor environment which Turtlebot has to follow.

To create the JSON file with EDL we have to first create a map of the required area, where the robot will navigate. The procedure for map creation with mounted Lidar or Kinect sensors is presented in a previous chapter, as well as in the appendix. The output of the map creation are two files a YAML and a PGM which are used for the navigation and the mission creation for the Turtlebot. Also before creating a JSON file of “waypoints” it could be useful with the help the RVIZ tool to navigate with Turtlebot through the created map to verify the path it should follow. Finally we can use the graphical interface or the editor described in the *EDL paper* already presented to produce the JSON file of mission “waypoints”.

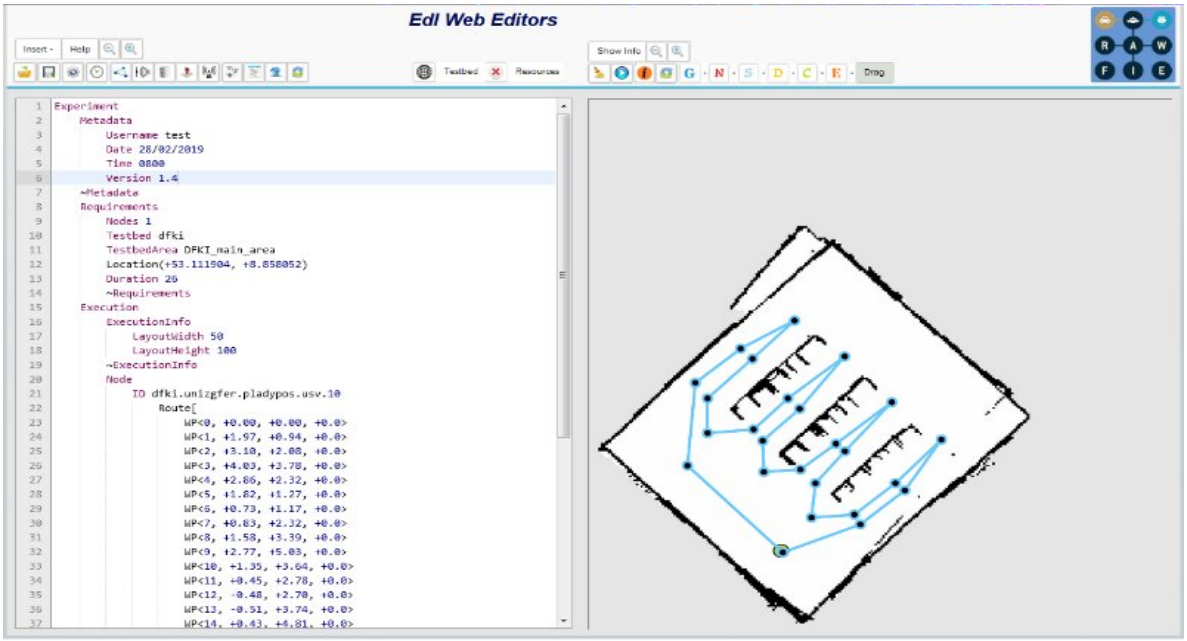

Figure 42: Here is presented the EDL Web Editors, left side is the json editor and on the right side is the graphical map with the waypoints

In this picture we can observe the editor on the left for the EDL and the graphical interface on the right side. The map of our simulated indoor environment appears in the graphical interface which was produced by scanning the simulated environment with the Turtlebot's simulated laser. Exactly the same result is produced with the physical Turtlebot or any other robot in real environment.

The created file of “waypoints” can be replaced by a custom made JSON file with positions and angles. The only thing that will change is how the program parse the JSON or any other file since the structure will be different.

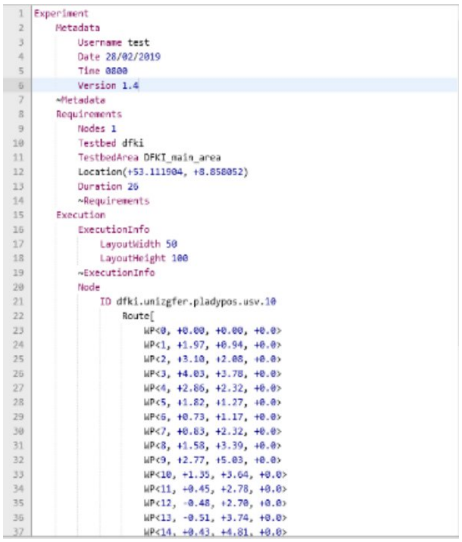

Figure 45: EDL JSON File

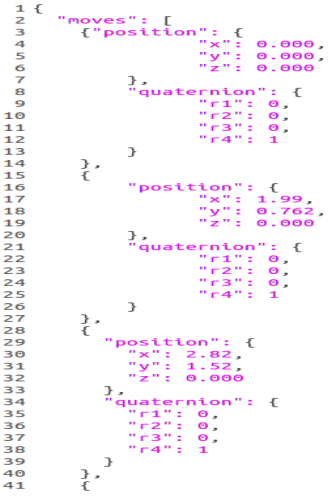

Figure 44: Custom JSON File

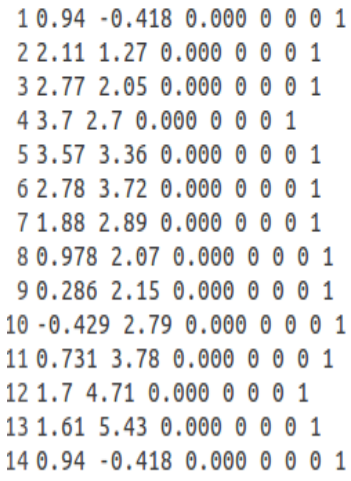

Figure 43: Simle Text File

10. Apache Kafka

Apache Kafka[101] is an open source software meant for building streaming and processing applications developed in Scala and Java by Apache Software Foundation, although it was originally developed by LinkedIn. Apache is a distributed system with the capability to manage trillions of events per day. It is based on a distributed commit log and it provides the functionality of subscribing and publishing enormous amount of data to multiple systems and applications in real time. Apache Kafka can handle enormous number of messages and perform real-time stream processing on the messages. The three main functionalities Apache Kafka provides are Publish & Subscribe streams, Process streams, and Store streams. The main characteristics of Kafka are the fact that it is distributed, it can scale horizontally, it is fault tolerant, and as it is obvious in the next paragraph it used in production for many companies

[102]Apache Kafka is widely accepted use cases of Kafka are Messaging and Website Activity Tracking, Log Aggregation, Stream Processing, Event Sourcing and Commit Log. [103]These use cases are used by thousands companies as airbnb, LinkedIn, Netflix, Uber, Twitter, Cloudflare, Mozilla, Yahoo, Coursera, Oracle, etc. and the list can continue for at least two paragraphs since thousands of companies use Kafka. So it is obvious that there is something attractive in Apache Kafka that makes it quite a popular tool among all these well known companies that develop industrial applications.

10.1. Why Apache Kafka

In this master thesis “Publish & Subscribe” part of Apache Kafka here are presented several reasons why Apache Kafka is preferred to other possible solutions[104].

1. Open-Source
2. Free
3. Used by industrial applications
4. Scalable
5. Fault tolerant (distribution & replication)

101 https://www.confluent.io/what-is-apache-kafka/

https://en.wikipedia.org/wiki/Apache_Kafka

https://kafka.apache.org/

102 https://kafka.apache.org/uses

103 https://cwiki.apache.org/confluence/display/KAFKA/Powered+By

104 https://techbeacon.com/what-apache-kafka-why-it-so-popular-should-you-use-it,

6. **Publish – Subscribe messaging system**

7. StackOverflow, Google, and Kafka GitHub Apache Kafka related question has grown up exponentially

10.2. Apache Kafka for RFID tags publishing

RFID antennas are recording RFID tags while moving in a warehouse. The collection of RFID tag data is done with an already implemented algorithm with minor changes from[105] *E-Pres* paper presented in the beginning of this thesis. All these data has to be somehow published to the interested parties. This task is performed by Apache Kafka Publisher. When a new RFID tag is detected it is published to a topic by Apache Kafka publisher if the RFID tag detected regardless if this RFID tag is duplicate or not. Data published to a predefined topic is later consumed by consumers who have subscribed to the published topic. So we create a producer consumer like architecture. Moving deeper into Apache Kafka isn't the target of this work so default apache configurations are implemented. For a more detailed configuration more Kafka components need to be configured: topics, topics partitions, partition offset, replicas of partition, brokers, Kafka cluster, leader node, follower node, etc.

> producer = KafkaProducer(bootstrap_servers=bootstrap_servers)

…

> producer.send(self.topic, value=bytes(content), timestamp_ms=timestamp_ms)

In the previous lines Apache Kafka Publisher's basic use case is presented written in Python programming language.

105 https://github.com/donMichaelL/reader_MTI_RU_861_010 & https://github.com/donMichaelL/e-pres

11. RFID TAG – POSITION – PATH – DATA COLLECTED

In this chapter are explained the data that we are interested in collecting. While the robot is moving, two main types of data are produced, for the data handling are used simple data structures.

1. Turtlebot's navigation data
    a) Current Position of the TurtleBot
    b) The Current Goal Turtlebot is traversing
        1. Start position
        2. End position
2. RFID tag data
    a) RFID value
    b) Antenna (0 or 1) located the current RFID
    c) RSSI value

The RFID data is produced on the side of RFID algorithm and the data about the position and goals is produced on the side of the ROS. The RFID data collection is performed with the MTI RFID RF module with minor changes performed on an already implemented algorithm[106]. The problem here is the combination of both RFID data and Odometry(pose) data. Both has to be located inside of a ROS catkin workspace. Also although it is possible to implement ROS programs in both C++ or Python it is easier to use the same language for both ROS and RFID tags detection. Here is presented the part implemented in this work for data handling.

11.1. Publishing RFID Tags on top of Apache Kafka publisher

The first mechanism for transmitting in real time the collected RFID tags is Apache Kafka publisher – subscriber. The collected RFID tags are published though the Apache Kafka publisher on a predefined topic waiting to be consumed by the interested parties.

106 https://github.com/donMichaelL/reader_MTI_RU_861_010 & https://github.com/donMichaelL/e-pres

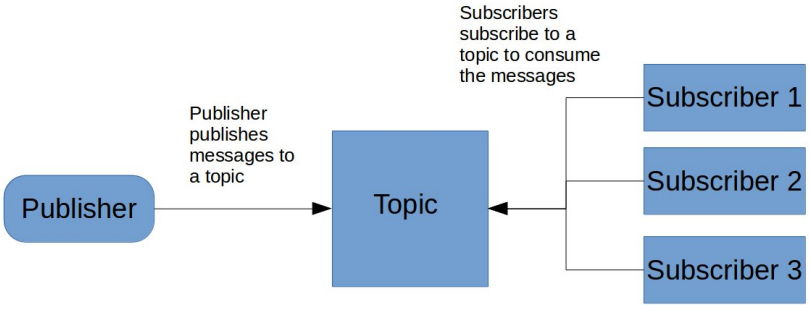

Figure 46: Simple Abstract Apache Kafka Architecture

> producer = KafkaProducer(bootstrap_servers=bootstrap_servers)

…

> producer.send(self.topic, value=bytes(content), timestamp_ms=timestamp_ms)

Streamed data from Apache Kafka can be consumed from anyone interested in these data from a known address on a predefined topic.

11.2. Publishing on a ROS Topic the collected RFID Tags, Goal, and Position

Since in this project ROS is used for the control and navigation of the Turtlebot, it could be useful to post the collected RFID tags as ROS topic. This could be useful in the future applications because RFID technology could be useful not only for the goods detection in a warehouse. For example ROS could read the specific RFID tags, first filtering out the RFID tags that are unnecessary and finally using this tags verify and determining a more precise location.

Data published on ROS Topic could be used in a scenario with multiple robots in a warehouse. Data collected from a group of robots could be combined analyzed between the robots in a peer to peer manner. By exchanging this data robots could succeed in the following:

1. Better RFID tag detection.
2. Determining with more precision the position of each RFID tag.
3. Sharing information about already scanned areas of the warehouse.
4. Better navigation in the warehouse.
5. Sharing information about obstacles.
6. Better load balance of resources(robots) in the warehouses

7. etc.

So a simple structure was implemented for publishing to a ROS topic the discovered RFID tags, current position, and current goal. This structure could be improved in the future with implementation of a dedicated type of message for RFID tags sent though ROS topic.

Creating a topic

self.pub = rospy.Publisher('rfid_topic', String, queue_size=10)

… … …

Publishing to a ROS topic

self.pub.publish(content)

This is a simple example implemented in Python for initializing and then publishing to a ROS topic.

11.3. ROS Logging

ROS Logging is another mechanism provided by ROS described in previous chapter. This mechanism is also provided by ROS, It is very useful for fast an easy monitoring of what is going on within an implemented ROS program. It is possible to view the output directly on the terminal or though a program provided by ROS and called “rqt_consol”. It is a really simple and convenient way for real time logging with the help of ROS.

rospy.loginfo("[" + str(timestamp) + "]" + str(content))

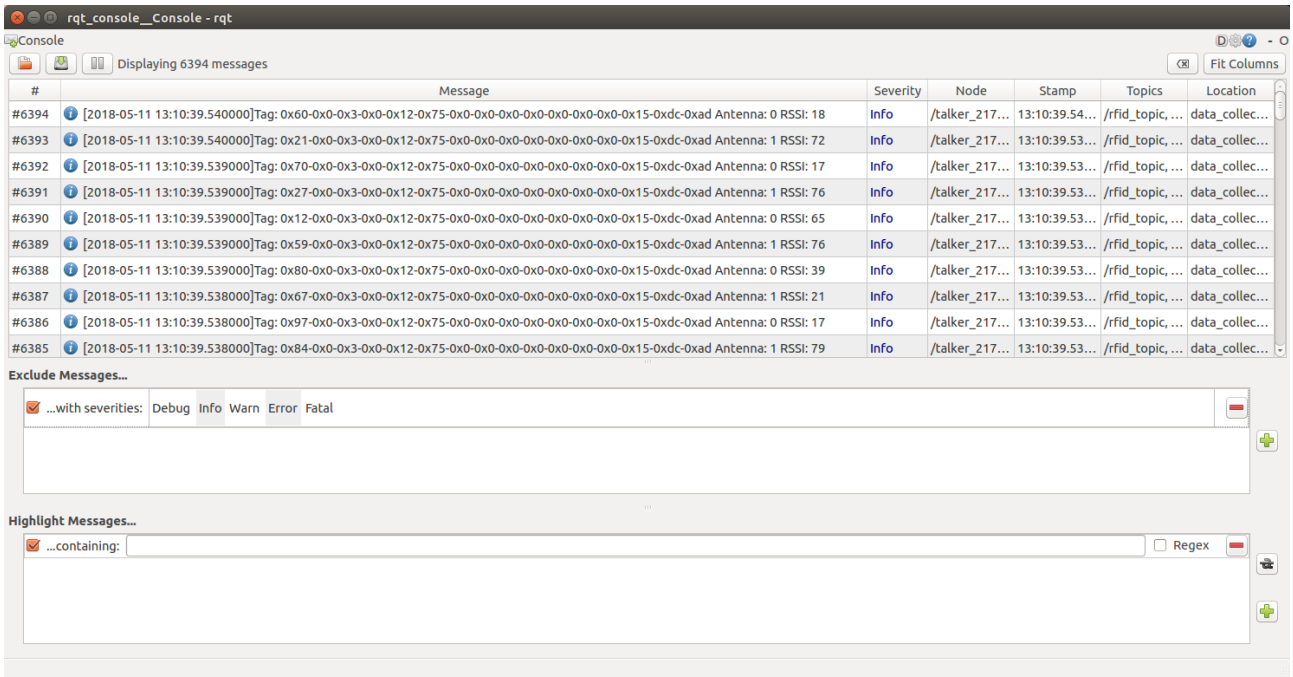

Figure 47: RQT console for showing ROS logs

11.4. Simple File Storage

Finally the collected RFID tags are saved in simple text file, this could be replaced by a database for example by a RDBMS or a Non Relational Database, but in this specific example for as the results would be the same. RFID tags need to be stored for further processing and study this is the main reason why RFID tags are saved.

11.5. Simple hash filtering of collected RFID tags

Another approach of handling the RFID tags is by storing them in a specially designed hash structure. The RFID tags that are collected by the RFID reader are stored where no duplicates can exist. Each RFID tag located by the RFID reader could appear multiple times in each appearance each RFID tag has a different value of RSSI which is the received signal strength indicator. By inserting the tags directly into the hash structure when a new tag appears it is inserted into the structure. If the newly inserted RFID tag wasn't detected before it is just inserted although if it has been detected previously the RSSI value of the previous and the current RFID tags are compared and the tag with the highest RSSI value is kept. This structure is important because we want to get the position of the robot when the signal of the RFID tag was as strong as possible.

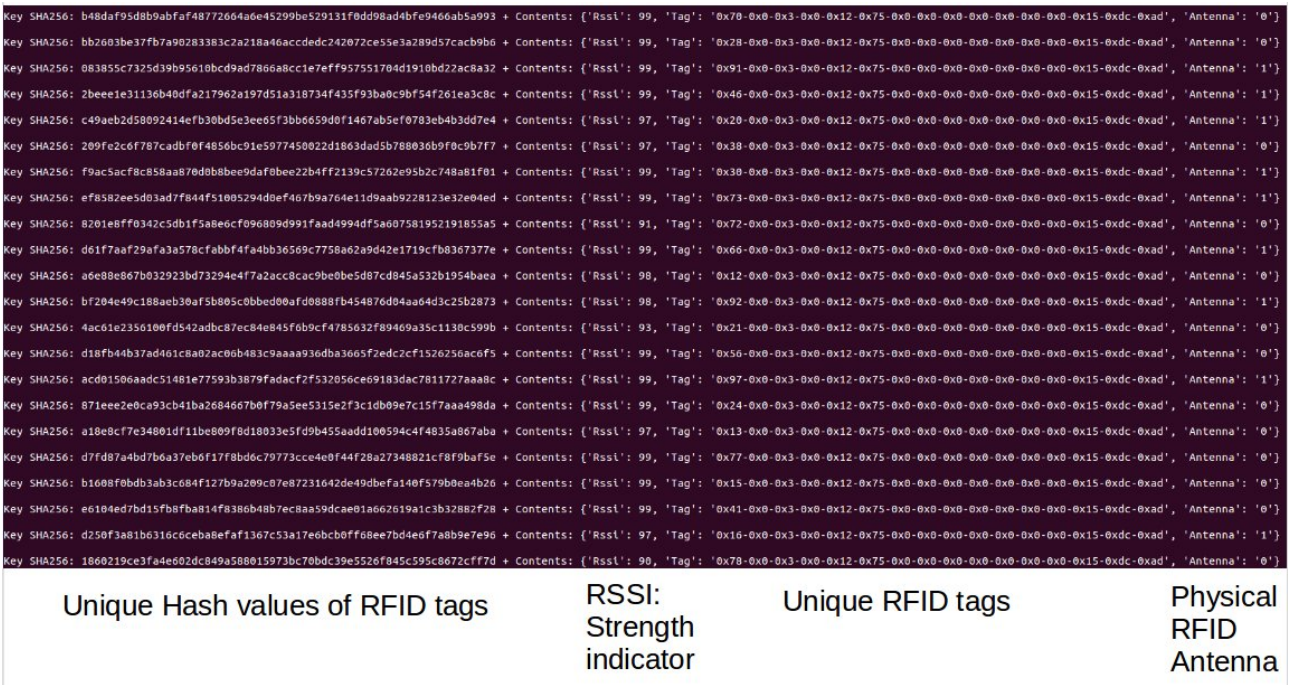

Figure 48: Hash keeps unique RFID tags with the highest value of RSSI

12. FINAL RESULT OF ALL SYSTEMS INTEGRATED TOGETHER

At this point all the separate parts are combined together

- Turtlebot
- Laptop running ROS
- Software
- Lidar
- RFID Antennas/Reader

Next are presented an abstract picture of all the system integrated together and also two pictures of the life cycle of the data movement starting from the RFID tags and finishing with the last stage of implemented Data Handler code. Through these two picture we can observe the architecture and software stack of the project.

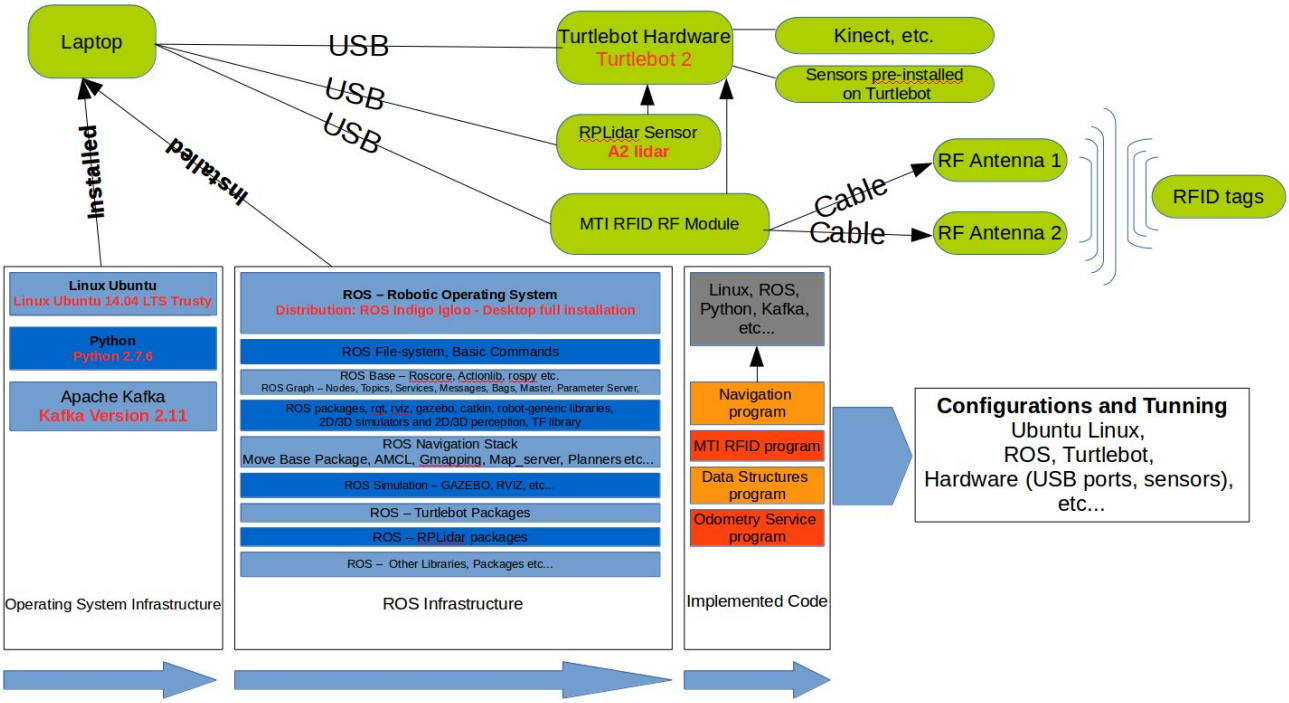

Figure 49: Abstract presentation of all systems integrated together, Green: Hardware, Blue: software installed or used, Orange: code implemented.

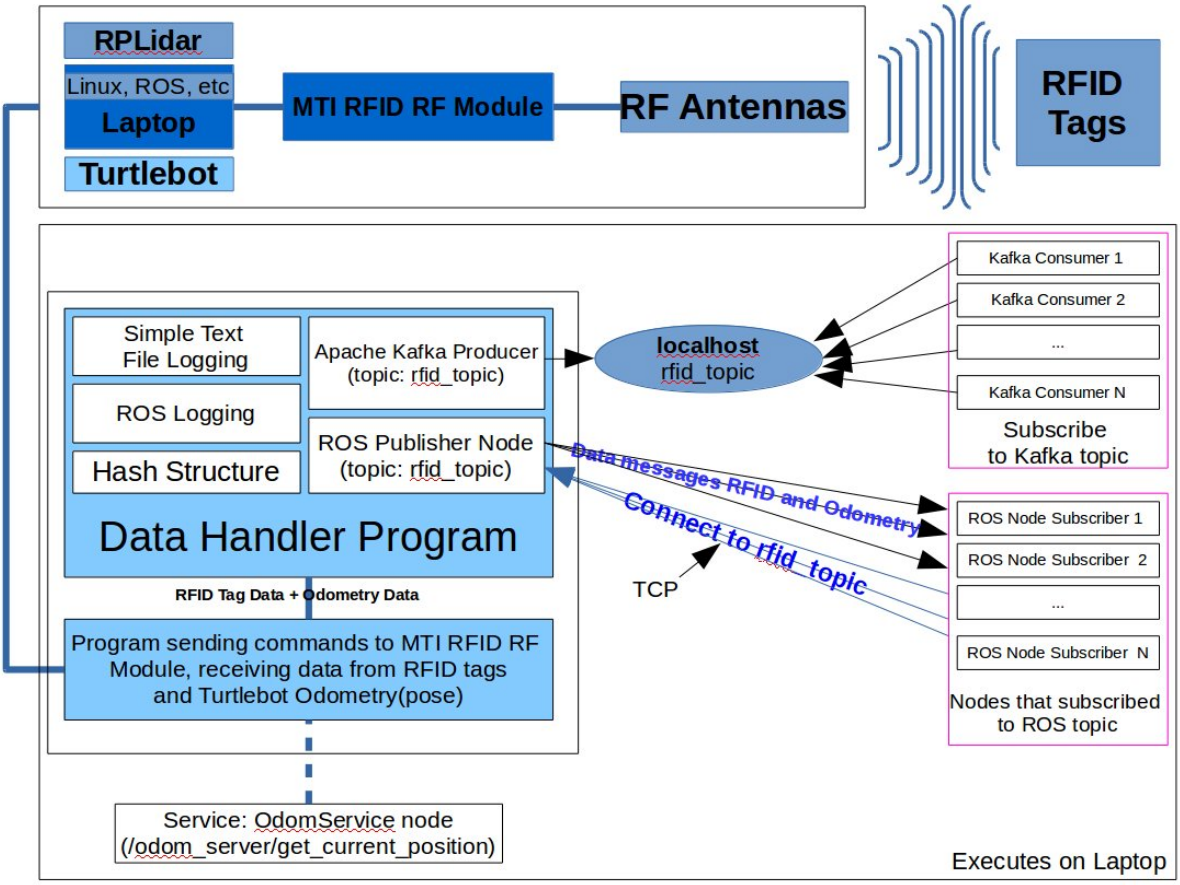

Figure 50: Data starts as a signal collected from RFID tags and Odometry and ends at Data Handler Program

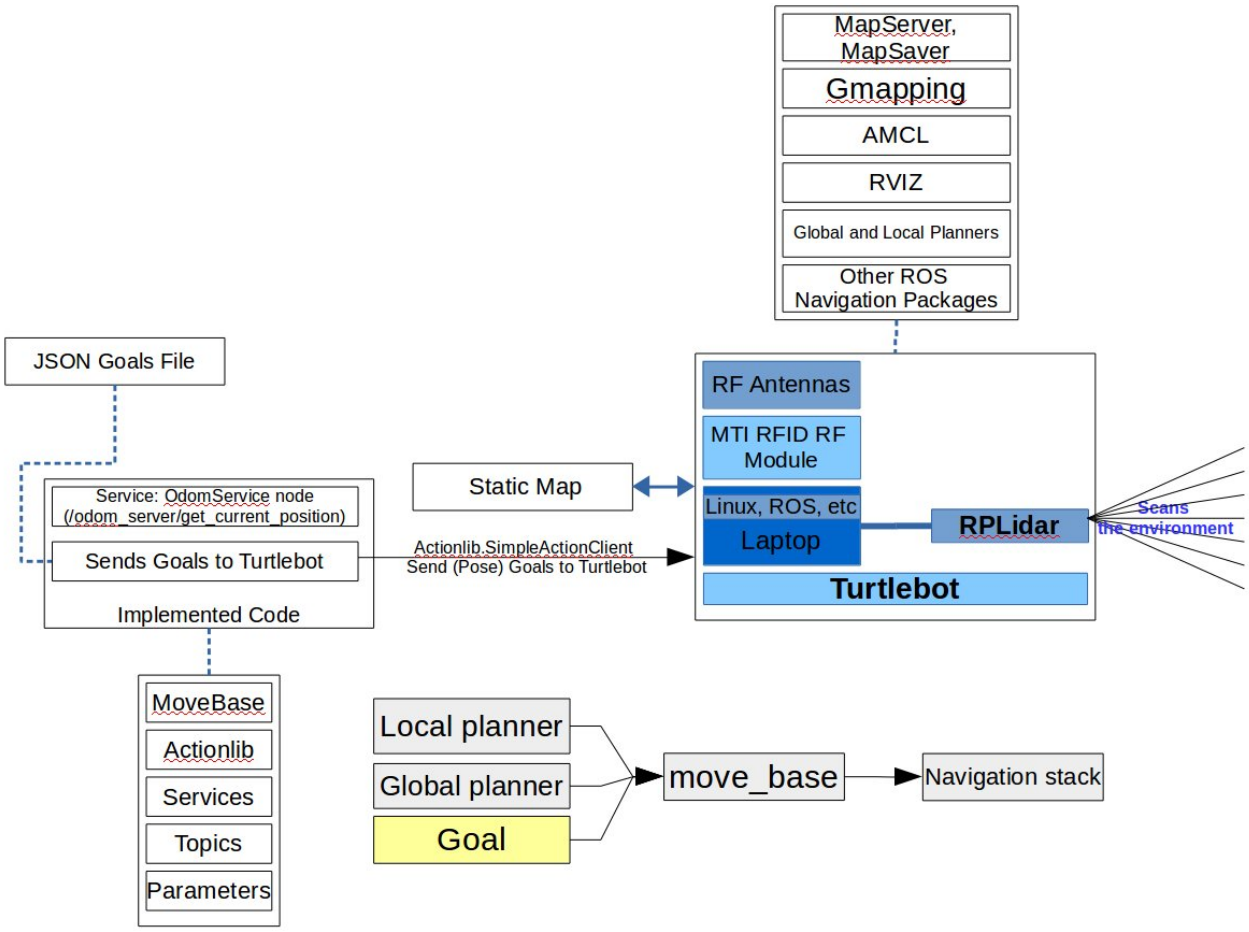

Figure 51: This picture represents approximately how a Goal is sent to Turtlebot.

Finally it is time to perform several final tests to get a spherical picture of the final system's functionality. It is important to mention that there are many possible improvements or changes but they are part of a possible future work performed in this domain.

Figure 52: This picture shows approximately how the final system looks like with all technologies integrated during the experiments and tests performed.

12.1. Experiments

Two types of experiments are performed the first type and the more accessible is in the simulated world and the second type of experiments is in the physical world. The implemented navigation program used for the Simulated and Physical world experiment is the same. For both experiments the results were as expected.

12.1.1 Simulated world experiments

Performing experiments in a simulation has many benefits. The experiments in the simulated world are performed to examine the movement of the robot in a warehouse world created in Gazebo. The experiments are performed with the help of Gazebo, and RVIZ tools to test if the implemented code for the navigation of the Turtlebot in the warehouse functions properly. The results about the movement of the Turtlebot were as expected and no extraordinary behaviors were detected.

To test the navigation algorithm which is described in previous chapters a simulated world was created with Gazebo. After the creation of the Gazebo world a map of the world was created with RVIZ tool. After the completion of the two previous steps it was possible to navigate in the created simulated warehouse. During the simulation we examine several cases:

1st Case : Navigation in the warehouse without obstacle appearances.

This is the simplest case of all where Turtlebot robot moves on the predifined path of goals without any unexpected behavior. Each time our robot completes a goal (reach the set position), next goal is sent until all goals are completed which means that our robot has

successfully traversed whole path of goals. So in this case the robot would move as described in previous chapter. If this case would be applied in the real physical world all the RFID tags should be detected normally

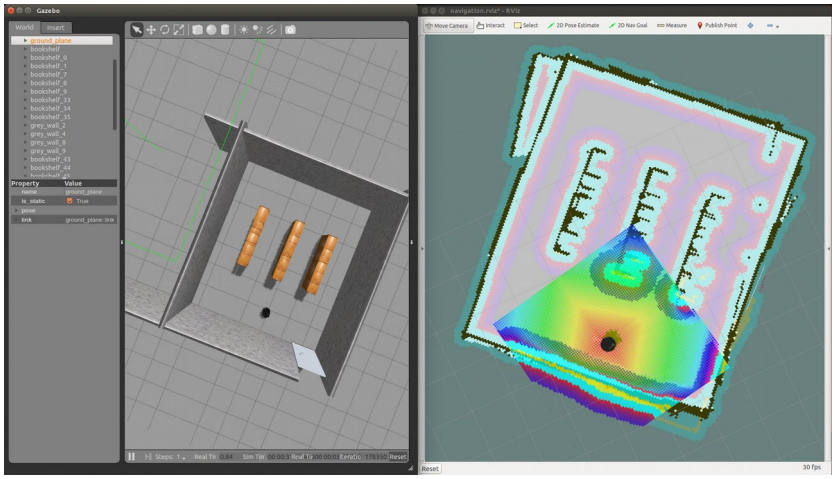

Figure 53: In this picture we can observe a simulation where no obstacles appear lying in the corridors of the warehouse.

2nd Case: Navigation with medium size obstacle occurrence.

This obstacle doesn't cover the whole width of the corridor so Turtlebot has enough space to overcome the obstacle from the left or right side of the lying obstacle. In this case our robot Turtlebot with the help of ROS packages described in previous sections recomputes the path and continues it's movement on the new path overcoming the object. Although an object appears on Turtlebot's path it doesn't affect the scanning of the RFIDs, normally all RFID IDs are expected to be detected by the RFID antennas installed on top of Turtlebot.

Navigation in the simulated warehouse while the obstacle is already lying on the path of the Turtlebot.

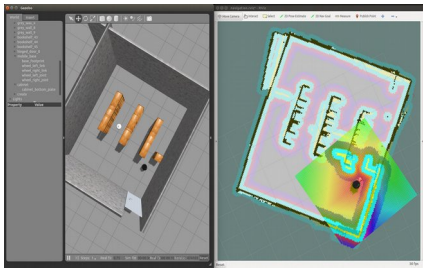

Figure 56: An obstacle appears on the Turtlebots path

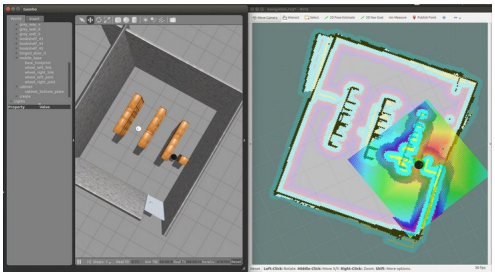

Figure 54: Turtlebot avoid the obstacle by moving around the left side of the object

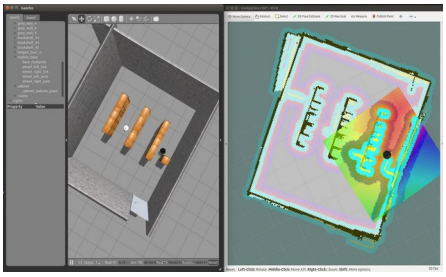

Figure 55: Turtlebot continues moving towards the specified goal

In this case we examine the case in which an object appears to lie in on the warehouse corridor where normally it shouldn't be. The size of the object is not covering the whole width of the warehouse corridor so the robot can overcome the obstacle by recomputing the path to the goal position(pose). As presented on the screenshots made from the simulated experiment in Gazebo and RVIZ Turtlebot detects the obstacle on its planed path, recomputes a new path, and overcomes the obstacle from the left side finally reaching its destination.

Unexpected obstacle appears in-front of the Turtlebot.

This case is similar to the previous case since the object is detected by Turtlebot's sensors so it can overcome the obstacle lying on its path the same way as presented in previous section.

Small objects that are not visible to the Turltbot.

In this case depending on the position of the Turtlebot sensors like Kinect and RPLidar, Turtlebot will locate or not this lying object. With the default configuration in the simulation Turtlebot doesn't detect a small object lying on the flour and continue moving towards its goal. Sometimes this behavior can have no consequences and the obstacle will be pushed away by the Turtlebot and in other case Turtlebot could be blocked and stack without the ability to perform any movement. Normally in this case Turtlebot will detect all the RFIDs as expected.

3rd Case: Navigation with large obstacle appearance. This obstacle covers the whole width of the corridor and Turtlebot doesn't have enough space to pass from any of two sides.

In this case our robot and specifically Turtlebot can't overcome the obstacle since it covers the whole width of the warehouse corridor. In this case Turtlebot as expected with the help of ROS packages creates a new path and pass from a different warehouse corridor. The problem that rises in this case is that Turtlebot leaves the warehouse corridor in which it was moving and moves through a completely different route. So it is following a completely different direction from the one described in a previous section. Since it is moving in a different direction it would be harder to detect which RFIDs are on the left side and which are on the right side of the warehouse corridor. Also during the change in the path a different corridor is scanned that has as result to scan the same corridor more than once. Last but not least the area where this large object is located is not scanned by the Turtlebot since it is not approachable

This case requires a future investigation to be performed and with a different solution to be applied. A solution applied for this case has to take in consideration Turtlebot's movement, the RFID antennas and the RFID detection.

Possible solutions are:

- A possible solution is to implement an algorithm for the robot direction detection so the direction of the robot movement inside of each corridor would be computed. By knowing the Turtlebots direction it would be obvious which RFIDs are detected on the left side and right side of the warehouse corridor.
- A different solution could be a deactivation of RFID antennas in such case until the completion of this goal.

Another possible solution for such case could be a total robot deactivation and an alert delivery until the obstacle removal.

12.1.2 Physical world experiments

To perform the experiment in a real environment, we have to perform the following steps:

1. Create a map of the surrounding environment in which the experiment is planned to be performed. The steps for map creation can be found in the Appendix of this work.
2. After the map creation we can try the navigation of Turtlebot in the world with the help of the created map. Steps for navigation of Turtlebot are described in the Appendix section
3. After the map creation with the help of the RVIZ tool we need to choose the Goals positions on the created map and after that create the JSON file of goals that contain waypoints.
4. Next step is to start the navigation with the help of the implemented program. Turtlebot will move on top of waypoints specified in the JSON file.
5. After being sure that the navigation of Turtlebot in the world is performed smoothly it is time to activate the RFID antennas. So we run the implemented algorithm for RFID tag data collection.
6. Finally after running the experiment we expect to have collected 15 unique RFID tags.

The integrated hardware Turtlebot, RPLIDAR, MTI module, RFID antennas, etc are presented in the following picture

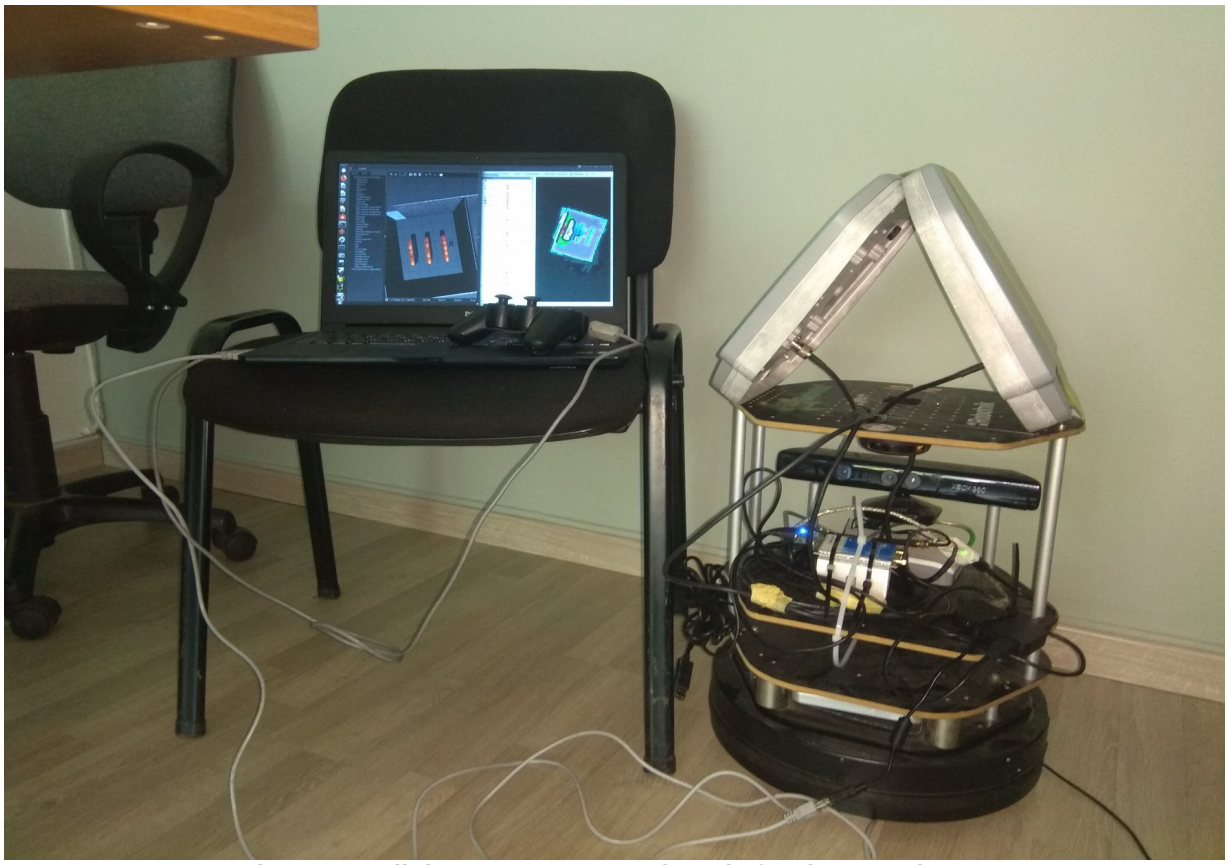

Figure 57: All the parts connected ready for the experiment

After the completion of all steps we are navigating in physical world and at the same time the MTI RFID RF Module antennas are scanning for RFID tags placed around the Turtlebot. The room where the experiment was performed is presented in the next pictures is visible where the RFID tags are placed

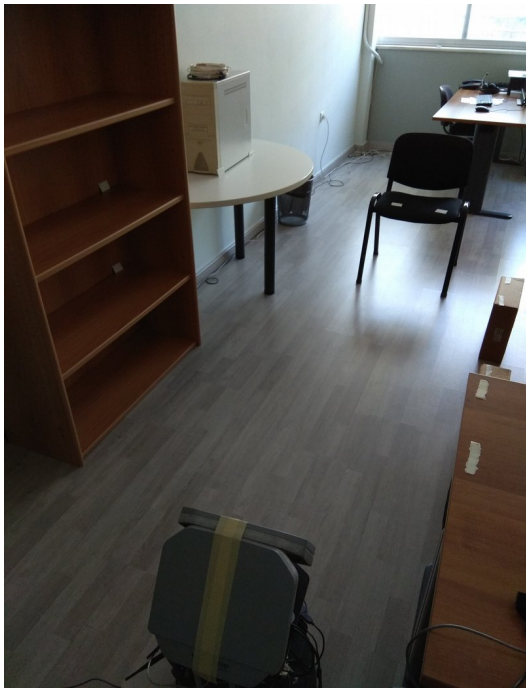

Figure 58: Room where the experiment was performed

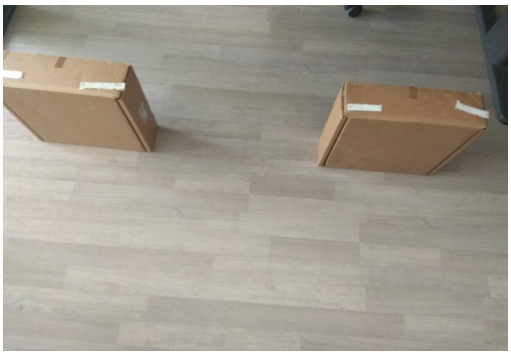

Figure 59: Right side of the Room were the boxes where positioned

Figure 60: RFIDs position on different hights

In the next picture the map of the room where the experiment was performed is presented also with red dots the positions of the RFIDs are marked.

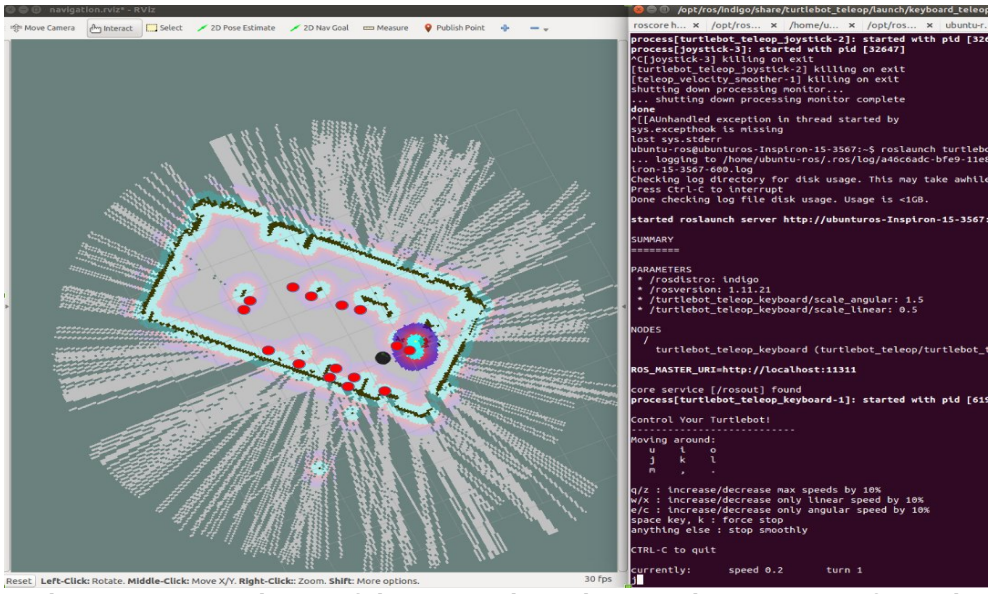

Figure 61: Created map of the room where the experiment was performed

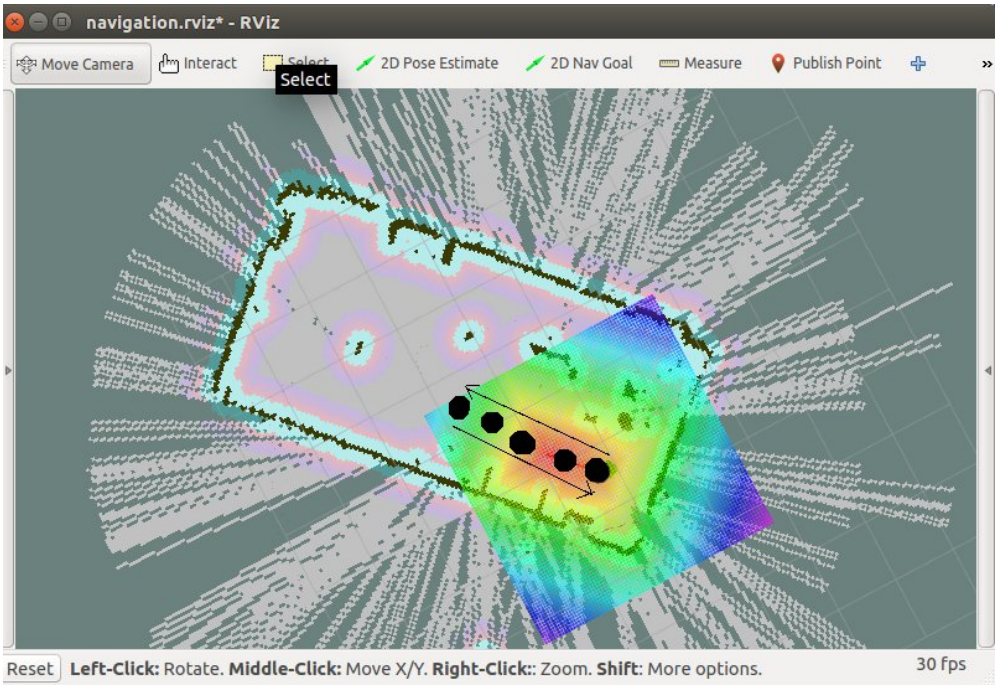

Figure 62: Turtlebot's path is marked with black dots. This picture shows the performed movement.

Finally after running the experiment from the data collected we can observe that all the RFID tags where detected by the MTI module/antennas. The total number of RFID tags was 15 and the detected RFID tags were 15 as well. Before placing the RFID tags, the room was scanned for other RFID tags.

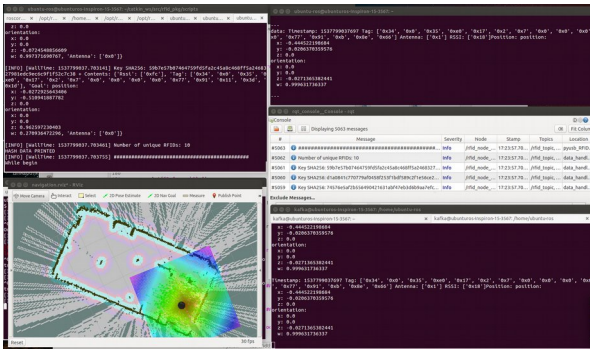

Figure 63: Windows while running the experiment in the left top window it is visible that 10/15 RFIDs tags are detected, RFID tags and Odometry data collected during the experiment.

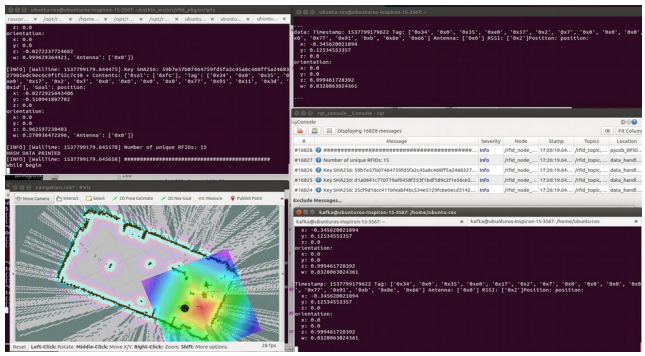

Figure 64: Windows while running the experiment in the left top window it is visible that 15/15 RFIDs tags are detected, all RFID tags are detected

During the navigation and the RFID detection both RFID tags and Odometry data were collected and stored in a hash structure which hold unique RFID tags with the highest RSSI with exact position where this RFID tag was detected. So it is possible to confirm the position the Turtlebot during the RFID tags detection.

The experiments in the simulated and in the physical worlds prove that the integration of ROS, Turtlebot, RPLidar, and RFID technologies is possible and successful for the automated RFID detection.

13. FUTURE WORK, IMPROVEMENTS AND ADDITIONS

13.1. Better implementation and design of custom ROS path planner algorithm

ROS has already several implemented path planners although they are pretty good for what they are meant to do there is steal space for improvement. A path planner should be implemented that could meet the following requirements.

1. Let the user choose if he want to set a position for the goal inside an object (combination of “global path planner “and “carrot planner”)
2. When suddenly an obstacle appears on top of set goal's destination position let the robot move as close as the user defined in the configuration parameters to the goal.
3. Let the user retrieve the path without sending it to the robot independently from where the goal is set and return a path as close as possible to the position of the set goal.

As far as we know there are no implemented ROS path planners global or local with three functionalities.

13.2. Experiment with other systems for better getting more complete experiment results and evaluation

Other substitute systems could be used for the implementation of this work not because the system selected here are not appropriate but to have a more spherical results and more solid prove that the systems and technologies used here have been chosen right.

13.3. Experiment in different environments

The environments available for the experimentation were limited so a more extensive and precise evaluation of the system could be performed. Experimentation in different environments could reach the robotic system to its limits providing a greater set of results for analysis.

13.4. Additional sensors

An example of an additional sensor used is a “fire sensor” this sensor could provide additional level of security to the already installed fire detection systems. For example a robot moving in a warehouse could faster analyze and detect a possible fire in dense warehouse full of goods. Also in a warehouse where more than one robot are working such kind of sensor could add a higher level of security if a fire appears.

13.5. Intruders detection

Although there are many levels of security at warehouses with thousands of goods. Another level of security could be added though a special intrusion detection algorithm running on the robots inside of a warehouse. Most of robots has mounted cameras or

sensors for navigation this sensors could be used for motion detection for detecting movement or facial recognition for detecting authorized working stuff. If an unexpected behavior is recorded by the robot an allert could be send to the security staff.

13.6. Positions with tags usage of two different tag types (1.products, 2.positioning)

Two different tag types could be introduced to a warehouse. The first type could be for detecting and recording, or in other words performing inventory as described in this master thesis and a second sensor for better position in the warehouse could be introduced. This would elevate the navigation skills of the robots and give more abilities for navigating inside of the warehouse and any other space.

13.7. Using GPS or Wifi positioning of the robot in the warehouse

A more precise systems like wildly known GPS or even Wifi could be used for better positioning of Turtlebot or any other robot in a any space where it is performing its work.

13.8. A commercial completed Web/Mobile/Desktop Application for live monitoring, data filtering, revision of saved RFID data.

For a more commercial solution it is important to implement a fully operational system that will host all the necessary for the configuration and handling of the robotic system.

13.9. Implementation of a complete package for RFID and ROS integration (if possible)

ROS is really evolved robot system the reason why this happens is because it open source and the community builds and shares its creations. It could be really useful a package implemented in ROS specifically for the MTI Reader/Module (used in this master thesis) for out of the box use of the MTI reader alongside with ROS by just installing the package alongside with the per-installed ROS system.

13.10. Implementation of an algorithm for detection of what was expected and what was really detected in the warehouse.

It is often that goods are stolen from warehouses this leads to economic loss. This could be avoided with the system implemented in this master thesis. In this master thesis an algorithm is already used for the collection of RFID tags mounted on top of object. By keeping also a database of all objects that are expected to exist in the warehouse at the current moment it is possible to track the difference between the detected goods and the expected goods that normally should be on the shelves of the warehouse.

13.11. Implementation of an algorithm for automatic RFID antenna and tag direction detection.

Since robot in several cases can change direction this can create confusions about which tags were collected on which side, left or right of the warehouse corridors. In our case we are interested about the exact side of the corridor where the RFID tag was detected, left or right. So an algorithm that wouldn't mind the direction towards which the robot is moving.

This algorithm would automatically detect the side on which the RFID tags were detected. An algorithm like this needs to detect the direction of the robot movement.

13.12. Automatic antenna activation and deactivation

Robot could periodically perform validations of the position where it is moving and if it has already passed from a position the RFID antenna inventory could be paused automatically.

14. CONCLUSION

It is obvious that a lot of interesting work could be be done in the domain of robotics, automation etc. and in the fields relevant to this work. Speaking about this work the result met our expectations although several unexpected problems where met mostly relevant to the behavior of the technologies used. The knowledge acquired though this work is very constructive and useful for future research in many different fields. The final result presents a working prototype in a supervised and controlled environment functioning as expected.

15. ABBREVIATIONS - ARCTICS – ACRONYMS

Abbreviation	Full Phrase
ROS	Robotic Operating System
RFID	Radio Frequency Identification
LIDAR	Light Detection and Ranging
GPS	Global Position System
Wifi	Wireless Local Area networking (Wireless Fidelity)
DSL	Domain Specific Language or more precisely
EDL	Experiment Descriptive Language
RDBMS	Relation Database System
RSSI	Received Signal Strength Indicator
NFC	Near Field Communication
UHF	Ultra High Frequency
HF	High Frequency
LF	Low Frequency
RF	Radio Frequency
UV	Ultra Violent
dBm (dB_{mW})	Power ration in decibels (dB) of the measured power referenced to one milliwatt (mW)
dB	decibels
HID	Human Interface Device
USB	Universal Serial Bus
Rviz	ROS Visualization
JSON	Javascript Object Notation

Abbreviation	Full Phrase
SDK	Software Development Kit
API	Application Programming Interface
XMLRPC	XML Remote Procedure Call (RPC)
RPC	Remote Proceude Call
XML	Extensible Markup Language

APPENDIX I

I.1 Technologies used in this master thesis

- ROS Indigo
- Turtlebot 2
- RPLIDAR A2 lidar
- RFID Antennas, MTI RFID Reader, RFID Tags
- Linux Ubuntu 14.04 LTS Trusty,
- Python Programming Language 2.7
- Apache Kafka 2.11

etc.

I.2 Linux 14.04 LTS

In this master thesis Ubuntu Trusty 14.04 LTS Linux Operating System[107] was used along with ROS Indigo Igloo (ROS Indigo Igloo is ONLY compatible with Ubuntu Saucy 13.10 and Trusty 14.04 LTS). These choices where taken due to the compatibility with the real Turtlebot robot of the lab which was available during this master thesis period for experimentation purposes. Instructions according the Installation of Linux are available on-line and are not described here. In the next chapter the installation of ROS Indigo is described. Both Ubuntu 14.04 and ROS Indigo are Long Term Supported and fully compatible with each other[108].

If you have Linux already pre-installed check your distribution version by running the following command :

```
> lsb_release -a
```

I.3 Installation and Configuration of ROS Indigo Environment

ROS Indigo Igloo[109] is on of ROS Long Term Service distributions it's main target platform that offers support is Ubuntu 14.04 LTS Trusty release (although other systems supports ROS Indigo Igloo to a **certain level** like other Linux systems, Mac OS X, Android, and Windows). It is was released on July 22nd, 2014 and it is supported until April, 2019 (Trusty EOL). Finally according the architecture there are binary packages available for both 32 and 64 bit architectures.

107 Older distributions of Ubuntu could be accessed on the following page http://releases.ubuntu.com/14.04/.

108 http://www.ros.org/reps/rep-0003.html

109 http://wiki.ros.org/indigo

I.4 ROS Indigo Installation

In this chapter several steps for the installation and configuration of ROS Indigo Environment will be described as fast as possible. For more information official ROS website information is available.

Step 1.

Check your Ubuntu distribution since ROS Indigo supports Saucy (13.10) and Trusty (14.04) for Debian packages. If you want to install Ubuntu refer to the chapter where Ubuntu installation is described. Open a terminal and type the following command **lsb_release -a** the result presented in the picture will appear.

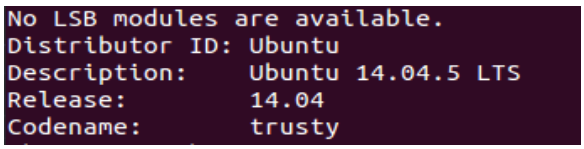

Figure 65: Ubuntu distribution 14.04.5 LTS trusty

Step 2.

Configure Ubuntu repositories to allow “restricted”, “universe”, and “multiverse”.

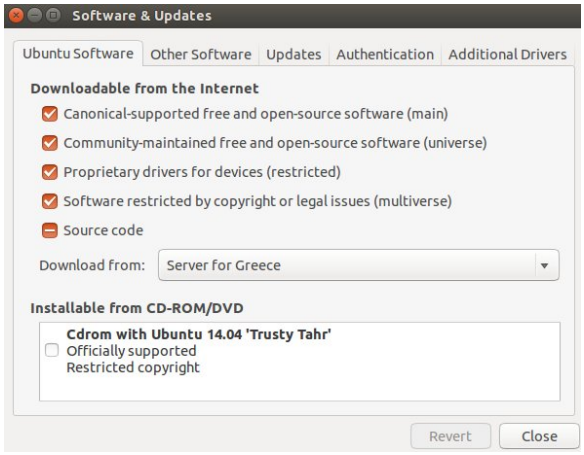

Figure 66: Ubuntu allowed repositories

Step 3.

Setup your sources.list to accept software from packages.ros.org

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release
-sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Step 4.

Setup the keys by running the following command

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

other key servers: 1) hkp://pgp.mit.edu:80 and 2) hkp://keyserver.ubuntu.com:80

Step 5.

Next run in your terminal the following command to update the Debian package index

```
sudo apt-get update
```

Step 6.

Finally it is time to install the ROS Indigo in this work the desktop full installation was installed and generally the desktop full is recommenced because it includes all the packages necessary like ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators and 2D/3D perception. To install the ROS desktop full installation run the following command

```
sudo apt-get install ros-indigo-desktop-full
```

* To find the available packages run the following command : **`apt-cache search ros-indigo`**

I.5 Turtlebot Gazebo

[110]In this master thesis Gazebo was useful to test the movement of the Turtlebot in indoor environment. Before applying the application created on real physical TurtleBot.

- To install the turtlebot_simulator pachage software run the following command:

  `> sudo apt-get install ros-indigo-turtlebot-simulator`

1. To install Gazebo package run the following command in you terminal:

110 For more information about ROS Turtlebot simulation refer to the following sources:

http://wiki.ros.org/turtlebot_simulator

http://learn.turtlebot.com/2015/02/03/1/

http://gazebosim.org/

```
> sudo apt-get install ros-indigo-turtlebot-gazebo
```

2. It is important to notice that in each new terminal that is started, we need to configure each terminal with the ROS environment variables by running the following command:

```
> source /opt/ros/indigo/setup.bash
```

*To avoid this step add the above command in **~/.bashrc** file. Changing the distribution of the ROS in our case we are running the Indigo ROS distribution :

“source /opt/ros/{YOUR_ROS_DISTRIBUTION}/setup.bash”,

after this addition each newly started terminal will be per-configured.

3. Next start the Gazebo simulation, this commands will initiate the Turtlebot in the Gazebo simulated world.

```
> roslaunch turtlebot_gazebo turtlebot_empty_world.launch
```

The following window will pop up after several seconds. In which we can observe Turtlebot standing surrounded with objects created in the Simulated world of Gazebo simulator.

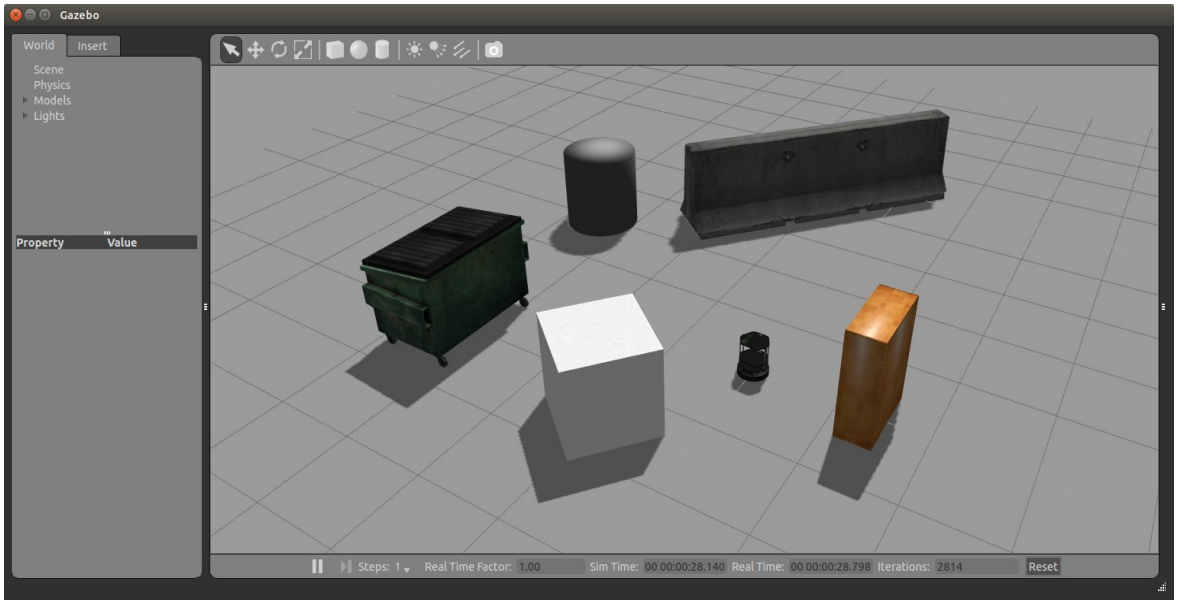

Figure 67: Gazebo simulation turtlebot_empty_world.launch

4. Finally by setting several environment variables TURTLEBOT_XXX it is possible to customize the simulated Turtlebot.

I.6 Move Turtlebot in simulated world

1. **`> sudo apt-get install ros-indigo-turtlebot-apps ros-indigo-turtlebot-rviz-launchers`**
2. **`> roslaunch turtlebot_teleop keyboard_teleop.launch`**
    1. **or** `roslaunch turtlebot_teleop ps3_teleop.launch`
3. **`> roslaunch turtlebot_gazebo turtlebot_world.launch`**

I.7 Creating Simulation of warehouse with Turtlebot in GAZEBO

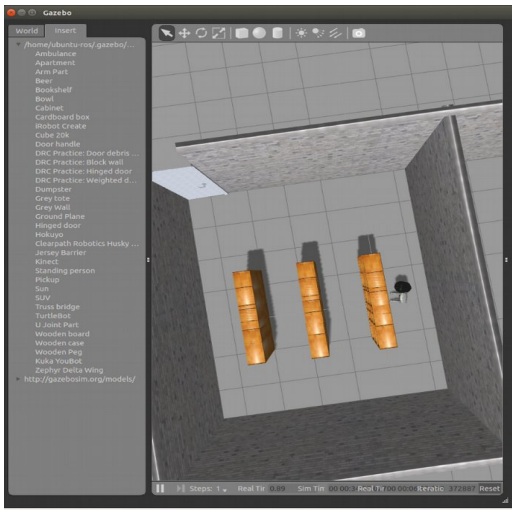

Figure 68: In this picture is presented the warehouse built with GAZEBO

1. Build your own world with GAZEBO by adding addition object to the simulation.

   >roslaunch gazebo_ros empty_world.launch

2. Save your world as **warehouse.world**

3. Open to turtlebot_gazebo

   >roscd turtlebot_gazebo

4. Move your created world to **/maps** directory

5. In **/launch** directory

   >cp turtlebot_world.launch turtlebot_world_warehouse.launch

6. In your turtlebot_world_warehouse.launch change the following bold line

```
<include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="use_sim_time" value="true"/>
    <arg name="debug" value="false"/>
    <arg name="gui" value="$(arg gui)" />
    <arg name="world_name"
value="/opt/ros/indigo/share/turtlebot_gazebo/worlds/warehouse.world"/>
  </include>
```

15.1. Create a map of previously created simulation

To create a map of previously created simulation run the following commands

1. >roslaunch turtlebot_gazebo turtlebot_world_warehouse.launch

2. `>roslaunch turtle_gazebo gmapping_demo.launch`
3. `>rosrun map_server map_saver -f my_map_warehouse`

`Before running the previous command start both Turtlebot and RPLidar.`

It will create a **my_map_warehouse .yaml** and **my_map_warehouse.pmg** files.

I.8 Navigating in previously created map

To navigate in previously created Gazebo simulation run the following commands

1. `>roscore`
2. `>roslaunch turtlebot_gazebo turtlebot_world_warehouse.launch`
3. `>roslaunch turtlebot_gazebo amcl_demo.launch map_file:=/home/ubuntu-ros/my_map_warehouse.yaml`
4. `>roslaunch turtlebot_rviz_launchers view_navigation.launch –screen`

To navigate in a real world run

I.9 Create a map of a physical world

1. `>roslaunch turtlebot_bringup minimal.launch`
2. `>roslaunch turtlebot_bringup 3dsensor.launch →` **`(kinect)`**

   `>roslaunch rplidar_ros rplidar.launch →` **`(rplidar)`**
3. `>roslaunch turtlebot_navigation gmapping_demo.launch`
4. `>roslaunch turtlebot_rviz_launchers view_navigation.launch`
5. `>roslaunch turtlebot_teleop keyboard_teleop.launch`
6. `>rosrun map_server map_saver -f my_map_warehouse`

I.10 Navigate in created map

1. `>roslaunch turtlebot_bringup minimal.launch`
2. `>roslaunch rplidar_ros rplidar.launch`
3. `>roslaunch turtlebot_navigation amcl_demo.launch  map_file:=/home/ubuntu-ros/tmp/my_map_warehouse.yaml`
4. `>roslaunch turtlebot_rviz_launchers view_navigation.launch`

I.11 RVIZ

RVIZ is a useful tool for many reason especially for navigation independently if we are

performing simulation or we are navigating in the real world.

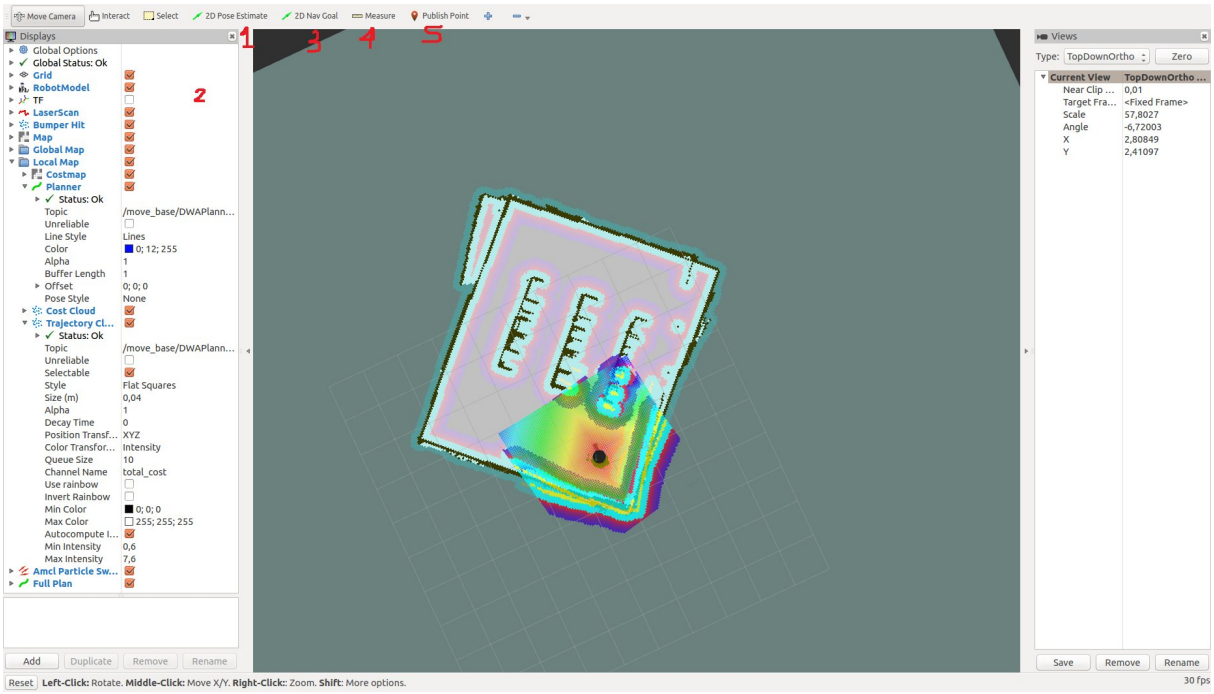

Figure 69: RVIZ, Turtlebot in the right bottom corner on top of created warehouse map

1. **2D Pose Estimate** is used for estimating the physical robot's position on the map
    - At the beginning the real position of the robot and the position on the RVIZ are not aligned. So performing 2D Pose Estimation should be performed. Sometime the robot can compute his position automatically with the help of AMCL.
2. **Left Pane** is used for changing parameters like planners or topic of laser scan and also for choosing what we want to appear in the RVIZ main panel.
3. **2D Nav Goal** is used for setting a goal to a robot, so the robot will navigate towards the specified position.
4. **Measure** is used for measuring distances.
    - In our case it was used for measuring the distance between the shelves of the warehouse.
5. **Publish Point** is used for finding an exact position (x,y) on the map.
    - Publish point was used for finding the exact positions on the map where we want to navigate. After computing all the required positions on the map it was possible to create a file with all (x,y) positions for consumption by our navigation algorithm.
6. **Central Panel** represent the main map, our robot, point cloud etc...

Generally in this RVIZ alongside with Gazebo were used always while performing experiments, navigating, creating maps. Etc.

I.12 Integrating RPLIDAR A2 with Turtlebot 2

Rplidar packages

In this project RPLIDAR is used for building a 2D map of an indoor environment, navigation and localization. Two packages **rplidar_ros** and **rplidar_python** are provided by ROS and support RPLIDAR A1 and A2, in our case RPLIDAR A2 is the one we are working with. RPLIDAR supports Hydro, Indigo, Jade, and Kinetic ROS distribution.

Assumption is made that Ubuntu 14.04 Trusty and ROS Indigo are already installed and configured, or any other distribution of Ubuntu and ROS computable with each other and rplidar packages. For Linux Ubuntu and ROS Indigo installation refer to the corresponding chapters.

Step 1 :

Installing ROS Indigo

```
> sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

> sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

(other key servers: hkp://pgp.mit.edu:80, hkp://keyserver.ubuntu.com:80)

```
> sudo apt-get update

> sudo apt-get install ros-indigo-desktop-full
```

Initialize rosdep

```
> sudo rosdep init

> rosdep update
```

Environment setup

```
> echo “source /opt/ros/indigo/setup.bash” >> ~/.bashrc

> source ~/.bashrc
```

Getting rosinstall

```
> sudo apt-get install python-rosinstall
```

Step 2

Build and setup catkin

```
> mkdir ~/catkin_ws/src/

> catkin_init_workspace

> cd ~/catkin_ws/

> catkin_make

> echo “source ~/catkin_ws/devel/setup.bash” >> ~/.bashrc
```

Step 3:

a) To check if **rplidar** package is already installed on our system we can run the following command :

```
> rospack find rplidar_ros

> rospack find rplidar_python
```

To find any available packages

```
> run apt-cache search ros-indigo
```

if rplidar_ros package is already installed this command will return a path to this ROS package on our system. If these packages are not already installed we can run the following commands:

```
> sudo apt-get install ros-indigo-rplidar-ros

> sudo apt-get install ros-indigo-rplidar-python
```

b) Inside of the **~/catkin_ws/src** clone the code from github [111]

by running the following command:

```
> git clone https://github.com/robopeak/rplidar_ros.git
```

and build with catkin by running the following commands:

```
> cd ~/catkin_ws/

> catkin_make

> source devel/setup.bash
```

Step 4 (Starting RPLIDAR)

111 https://github.com/robopeak/rplidar_ros (tested) or https://github.com/roboticslab-fr/rplidar-turtlebot2 (not tested)

Finally we are ready to plug RPLIDAR's USB into our computer USB port and run rplidar ROS package. For running rplidar ros package there are two different ways:

a) Run rplidar node and view in the rviz (launch demo with Rviz)

```
> roslaunch rplidar_ros view_rplidar.launch
```

RVIZ window will appear where it is possible to observe on top of the dark grey grid the feedback from the RPLIDAR which scans the surrounding environment in range between 15cm to 12/18m. The red marks on top of the grid represent the obstacle which exist around RPLIDAR.

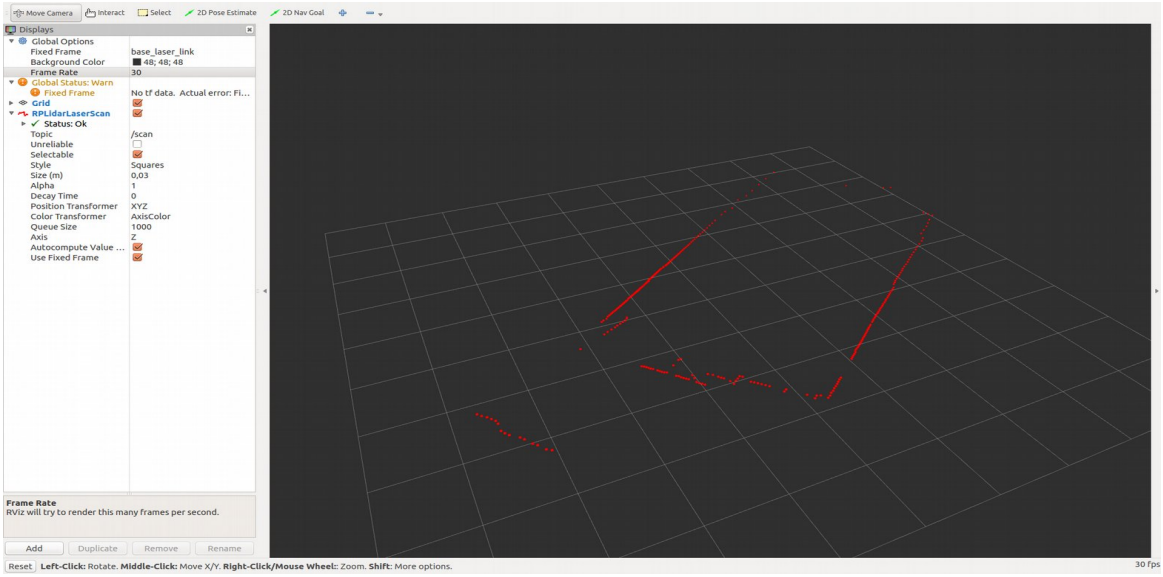

Figure 70: RPLIDAR A2 scan result of a room, read lines represent the wall of the room or other obstacles

b) b) Run rplidar node and view using test applications

```
> roslaunch rplidar_ros rplidar.launch
```

```
> rosrun rplidar_ros rplidarNodeClient
```

In this case RPLIDAR's scan result will appear in the console

```
[ INFO] [1523533183.932175750]: : [155.000015, 0.537000]
[ INFO] [1523533183.932233666]: : [156.000000, 0.532000]
[ INFO] [1523533183.932292319]: : [157.000031, 0.527000]
[ INFO] [1523533183.932349002]: : [158.000015, 0.522000]
[ INFO] [1523533183.932407436]: : [159.000015, 0.521000]
[ INFO] [1523533183.932465266]: : [160.000015, 0.518000]
[ INFO] [1523533183.932522486]: : [161.000000, 0.530000]
[ INFO] [1523533183.932580206]: : [162.000031, 0.503000]
[ INFO] [1523533183.932638383]: : [163.000015, 0.483000]
[ INFO] [1523533183.932696361]: : [164.000015, 0.466000]
[ INFO] [1523533183.932753632]: : [165.000015, 0.469000]
[ INFO] [1523533183.932811450]: : [166.000000, inf]
[ INFO] [1523533183.932869833]: : [167.000031, 0.468000]
[ INFO] [1523533183.932925922]: : [168.000015, inf]
[ INFO] [1523533183.932984369]: : [169.000015, 0.322000]
[ INFO] [1523533183.933041847]: : [170.000000, 0.310000]
[ INFO] [1523533183.933098197]: : [171.000000, 0.304000]
[ INFO] [1523533183.933156661]: : [172.000031, 0.300000]
[ INFO] [1523533183.933214444]: : [173.000015, 0.297000]
[ INFO] [1523533183.933271025]: : [174.000015, 0.296000]
[ INFO] [1523533183.933327087]: : [175.000000, inf]
[ INFO] [1523533183.933402587]: : [176.000000, 0.296000]
[ INFO] [1523533183.933472474]: : [177.000015, 0.297000]
[ INFO] [1523533183.933525861]: : [178.000015, inf]
[ INFO] [1523533183.933580111]: : [179.000015, inf]
```

Figure 71: Scan results from RPLIDAR A2

Step 5 (Troubleshooting) Adding serial-port authority of write

In case where RPLIDAR doesn't start spinning follow the following procedure.

a) Check the authority of the rplidar's serial-port:

```
> ls -l /dev | grep ttyUSB
```

b) According to the result from the previous step(in our case it is USB0) add the authority of write:

```
> sudo chmod 666 /dev/ttyUSB0
```

Step 6 (Troubleshooting) How to remap the USB serial port name

If RPLIDAR is still not responding and didn't start spinning there is a possibility that the change we performed in the previous step, for a **fixed rplidar port** run the following script from the rplidar ROS package :

```
> ./scripts/create_udev_rules.sh
```

now run the following command once again to check if changes took effect:

```
> ls -l /dev | grep ttyUSB
```

After completing the procedure of changing the USB port remap to a fixed name, change the launch file to the serial_port value, to edit the launch file type in the editor the following ros command:

```
>rosed rplidar_ros rplidar.launch
```

next edit the following line of the launch file

```
<param name="serial_port" type="string" value="/dev/rplidar"/>
```

another approach is just to change the value to /dev/ttyUSB0 or /dev/ttyUSB1 etc... every time when rplidar is mounted to a USB port.

After completing the steps 5 and 7 after running **ls -l /dev | grep ttyUSB** the following result will appear:

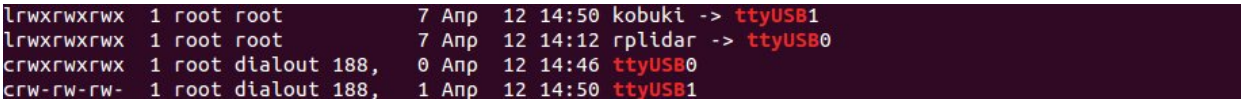

Figure 72: USB ports with RPLIDAR and KOBUKI mounted

*In my case for the changes to take effect removing and inserting again the USB devices to he USB ports was helpful.

After completing the Step 6 RPLIDAR is finally completely functional. Now it is time to complete the configuration of the launch files and the position where the RPLIDAR will be physically mounted on top of Turtlebot 2.

Step 7

Finally maybe there is a need to configure the turtlebot launch file to generate scan topic for both RPLIDAR and Kinect. To configure 3dsensor.launch file run the following commands:

```
> roscd turtlebot_bringup/launch
> gedit 3dsensor.launch
```

Change the following line :

```
<!-- Laserscan topic -->
       <arg name="scan_topic" default="scan"/>
```

change the default value from **scan** to **scan_kinect** so kinect publishes to different topic. After finishing with 3dsensor.launch file configuration we have to configure Rtabmap launch file to to let working rplidar publishing on scan and Kinect on /scan_kinect. In our case this step in not important since we are only interested in working with RPLIDAR and not Kinect. After completing the presented steps the integration of RPLIDAR and Turtlebot is complete[112].

I.13 Mounting RPLIDAR A2 on top of Turtlebot 2

Step 8

1. Adjust the proper urdf files **turtlebot_description/urdf/stacks/circles.urdf.xarco** and **turtlebot_create/../create_description/urdf/create.urdf.xacro** files appropriately by adding the appropriate content similar to :

```
<joint name="laser_joint" type="fixed">
                <origin xyz="[x] [y] [z]" rpy="[rot_x] [rot_y] [rot_z]" />
                   <parent link="[plate on which the laser is]" />
              <child link="laser" />
```

112 https://github.com/robopeak/rplidar_ros/wiki

https://github.com/robopeak/rplidar_ros/

http://wiki.ros.org/rplidar

https://blog.zhaw.ch/icclab/rplidar/

https://github.com/roboticslab-fr/rplidar-turtlebot2

```
</joint>
```

The **circles.urdf.xarco** is needed to locate where the sensor is planned to be installed. In the **create.urdf.xarco** file it looks like following code:

```
<joint name="laser_joint" type="fixed">
      <origin xyz="0.015 0.00 0.01" rpy="0 0 0" />
      <parent link="plate_3_link" />
      <child link="laser" />
</joint>
<link name="laser">
          <visual>
              <geometry>
                   <box size="0.04 0.04 0.02" />
              </geometry>
              <material name="Green" />
          </visual>
          <inertial>
             <mass value="0.001" />
             <origin xyz="0 0 0" />
             <inertia ixx="0.0001" ixy="0.0" ixz="0.0"  iyy="0.0001"
iyz="0.0"  izz="0.0001" />
          </inertial>
</link>
```

After applying that changes run RVIZ:

```
> roslaunch turtlebot_bringup minimal.launch

> rosrun rviz rviz
```

After RVIZ window appears don't forget to add robot model and fixed frame “base_link”. For this to work properly set the **TURTLEBOT_BASE** environmental variable to “**create**” in the next step change back the variable to “**kobuki**”

2. **`<node pkg="tf" type="static_transform_publisher" name="laser" args="x y z rot_x rot_y rot_z parent_link child_link  frequency"/>`**

for example

```
<node pkg="tf" type="static_transform_publisher" name="laser" args="0.015 0.00 -0.01 0 3.14159265 0 /plate_top_link /laser 50"/>
```

or run the following command

```
> rosrun tf static_transform_publisher  0.015 0.00 -0.01 0 3.14159265 0 /plate_top_link /laser 50
```

Step 9

Finally we are done with the preparation now it is time to test if everything works as expected.

1. `roslaunch turtlebot_bringup minimal.launch`
2. `roslaunch turtlebot_navigation gmapping_demo.launch`
3. `roslaunch rplidar_ros rplidar.launch` **or** `roslaunch turtlebot_bringup 3dsensor.launch`
4. `roslaunch turtlebot_rviz_launchers view_navigation.launch`

After executing the last command RViz window should appear and look similar to he following screen-shot:

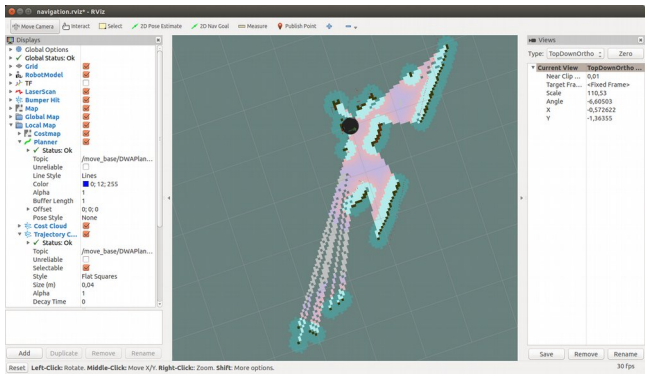

Figure 74: RVIZ Turtlebot scan with RPLIDAR A2

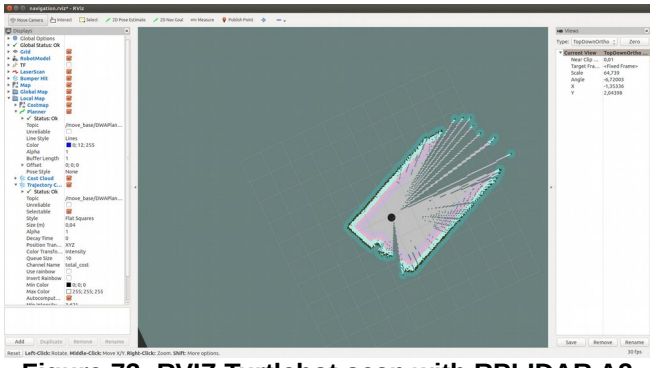

Figure 73: RVIZ Turtlebot scan with RPLIDAR A2

After this step we are ready to navigate with Turtlebot or create a map of an unknown area.

Finally an important observation according rplidar is that it **rotates with clockwise direction**. The first range come from the front. (the tail with the line).

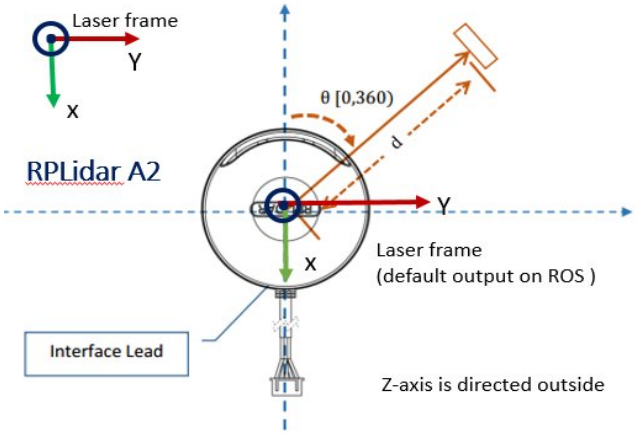

Figure 75: RPLIDAD A2 geometry representing the front the rear and the rotation direction of RPLIDAR A2 source: https://github.com/robopeak/rplidar_ros/wiki

I.14 ERROS – WARNING – PROBLEMS

PROBLEM 1.

ERROR presented next appears when **base is set to kobuki** and the following command is executed:

```
>  roslaunch turtlebot_bringup minimal.launch
```

"[ERROR] [1524658724.295353692]: Kobuki : malformed sub-payload detected. [206][170] [CE AA 55 4D 01 0F B8 00] ".

Although this error shows up it appears that the functionality we need is not affected from this error and turtlebot moves as expected.

It appears to be a USB problem computer problem but on a different source it is mentioned that this could happen due to dependencies problem and that the problem was solved after a fresh install of both OS and ROS[113].

PROBLEM 2.

WARN appears when **base is set to create** and the following command is executed:

```
> roslaunch turtlebot_bringup minimal.launch
```

"[WARN] [WallTime: 1524661448.188054] Create : robot not connected yet, sci not available "

113 https://github.com/introlab/rtabmap_ros/issues/191

https://answers.ros.org/question/52203/kobuki-malformed-subpayload/

and

“**[ERROR] [WallTime: 1524662134.356473] Failed to contact device with error: [Distance, angle displacement too big, invalid readings from robot. Distance: 16.66, Angle: 0.00]. Please check that the Create is powered on and that the connector is plugged into the Create.** ”

On-line there are several issues mentioned similar to this one but no obvious solution is suggested[114].

PROBLEM 3.

It is important to always have in mind the power supply of MTI RF RFID module because when the power is not enough the Antennas are not functioning properly as expected. In several cases when laptop run out of battery, Antennas didn't function properly. In another case during the experiments when the used USB cable connected to MTI RFID RF module was longer than 1.5m the Antennas didn't function properly. This behavior caused problems to the execution of the inventory program during the RFID tags detection.

I.15 Running Turtlebot in physical world

1. Roscore
2. Start Apache and subscribe to topic
3. `> rqt_console`
4. `> rostopic echo /rfid_topic`
5. `> roslaunch turtlebot_bringup minimal.launch`
6. `> roslaunch rplidar_ros rplidar.launch`
7. `> rosrun tf static_transform_publisher 0.015 0.00 -0.01 0 3.14159265 0 /plate_top_link /laser 50`
8. `> roslaunch turtlebot_navigation amcl_demo.launch map_file:=<path to yaml>`
9. `> roslaunch turtlebot_rviz_launchers view_navigation.launch`
10. Run odometry service

114 https://answers.ros.org/question/205105/error-bringing-up-the-robotwarn-create-robot-not-connected-yet-sci-not-available/

https://groups.google.com/forum/#!topic/ros-sig-turtlebot/GaN92YeW3aE
https://groups.google.com/forum/#!topic/ros-by-example/asVx2Eoga-k
https://answers.ros.org/question/269833/error-walltime-1504001477235354-failed-to-contact-device-with-error-error-reading-from-sci-port-no-data-please-check-that-the-create-is-powered-on-and/

11. Run navigation program
12. Run inventory program

I.16 Giving permissions to USB MTI RFID RF Module

1. Open directory for adding the rules: **cd /etc/udev/rules.d**
2. Determine the vendor id and produc id of the device : **lsusb**
3. Create a file for rules: **sudo pico rfid.rules**
4. Add the following lines

```
SUBSYSTEM=="usb", ATTRS{idVendor}=="24e9", ATTRS{idProduct}=="0861",
MODE="0777", GROUP="dialout"

SUBSYSTEM=="usb", ATTRS{idVendor}=="9449", ATTRS{idProduct}=="2145",
MODE="0777", GROUP="dialout"
```

5. Run: **sudo service udev reload** & **sudo service udex restart**

I.17 Command for running Kafka in Linux

1. **Open Kafka User :** `su kafka`
2. **Start Kafka :** `nohup ~/kafka/bin/kafka-server-start.sh ~/kafka/config/server.properties > ~/kafka/kafka.log 2>&1 &`
3. **Produce Message** : `echo "Hello, World" | ~/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic TutorialTopic > /dev/null`
4. **Consume Message** : `~/kafka/bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic TutorialTopic –from-beginning`
5. **Stop Kafka :** `~/kafka/bin/kafka-server-stop.sh`

I.18 Implemented Algorithms

In this project several algorithms where implemented for Data Handling, Goals and Movement of Turtlebot. These algorithms used several main ROS libraries, packages and core functionalities. Programs implemented are contained in the attached files.

REFERENCES

[1] Kostas Kolomvatsos, Michael Tsiroukis, Stathes Hadjiefthymiades. 2017. “An Experiment Description Language for Supporting Mobile Iot Applications”. National And Kapodistrian University of Athens. Department of Informatics and Telecomunications

[2] Michail Chatzidakis, Michail Loukeris, Kostis Gerakos, Stathes Hadjiefthymiades. 2016. “E-Pres: Monitoring and Evaluation of Natural Hazard Preparedness At School Community”. Pervasive Computing Research Group. National And Kapodistrian University of Athens. Department of Informatics and Telecomunications.

[3] Kshitija Deshmukh, Ashitha Ann Santhosh, Yogesh Mane, Saurabh Verma, Sdhana Pai. Nov 2015. “Robotic navigation and inventory management in warehouses”. International Journal of Soft Computing and Artificial Intelligence. ISSN. 3(2): 75-79

[4] Kaiyu Zheng. Sep 2016, “ROS Navigation Tuning Guide”

[5] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, Andrew Ng, 2009, ROSQ: an open-source Robot Operating System”, ICRA Workshop on Open Source Software, Kobe, Japan, 2009

[6] “Effective Robotics Programming with ROS” Third Edition by Anil Mahtani, Luis Sanchez, Enrique Fernandez, Aaron Martinez, Publisher Packt, 2016

[7] http://www.ros.org/

[8] http://wiki.ros.org/

[9] http://learn.turtlebot.com/

[10] https://en.wikipedia.org/

[11] https://www.turtlebot.com

[12] https://www.mtigroup.com/

[13] https://www.slamtec.com/en/Lidar/A2