People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering

Department of Power and Control

Final Year Project Report Presented in Partial Fulfilment of the Requirementsfor the Degree of

MASTER

In **Electrical and Electronic Engineering**

Options: **Control,Computer**

Title:

RFID platform for mobile robot navigation

Presented by:

- **RAIS Mohamed Cherif**
- **ZEROUATI Abdelmalek**

Supervisors:

Dr, A. MAACHE **Dr, I.AKLI**

Registration Number:………/2018

ACKNOWLEDGEMENT

First and foremost, we would like to thank ALLAH the almighty, for giving us the strength, knowledge and the ability to undertake this project study and complete it satisfactorily. Without his blessings, this achievement would not have been possible.

We would like to express our deepest sense of gratitude to the supervisor Dr. AKLI Isma and Dr. MAACHE Ahmed for his guidance, support and help throughout the different phases of the project, Mr. DJAZAIRI Salim the future Doctor with God's willing, And to all and every teacher who taught and guide us from primary school to the last year of study.

DEDICATION

Every challenging work needs self-efforts as well as guidance of Elders those who were very close to our heart.

My humble effort I dedicate to my loving

***parents** and my **family members**, Whose affection, love, encouragement and prays of day and night make me able to get such success and honor.*

*Along with all **my friends**, **hardworking** and respected **Teachers.***

Abdelmalek ZEROUATI.

ABSTRACT

Localization , map-building and motion control (navigation) are the three major tasks for a successful mobile robot navigation. This report studies a hardware in the loop simulation system using the Radio Frequency Identification (RFID) technology for the design and implementation of cost-effective and modular control technique to navigate a mobile manipulator in a virtual static environment.

The implemented navigation system is mainly composed of two parts: the RFID hardware platform and the Gazebo simulator for the robotic virtual environment. The RFID platform consists of multiple antennas, an RFID reader, and RFID tags which are placed in a three dimensional workspace. These tags provide necessary information about the environment for the reader. This information are then passed, in real-time, to the Gazebo simulator in order to generate appropriate control actions for the robot's actuators. These control actions make the robot reach target positions or track predefined trajectory in the virtual environment depending on the desired task. The system was successfully implemented and gave promising preliminary results.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY

A* : A star.
Actionlib : Action library.
AIDC. : automatic Identification and Data Capture.
AMCL. : Adaptive Monte carlo Localization.
API. : Application Programming Interface.
CAD. : Computer Aided Design.
D* : D star.
DBMS. : Data Base Management System.
EPC. :Electronic Product Code.
GPIO. : General Purpose Input Output.
OPC UA: Open Process Control Unified Architecture.
RFID : Radio Frequency Identification.
ROS:Robotic Operating system.
RVIZ : Robot Visualization tools.
SBC. : Single Board Computer.
SLAM. : Simultaneous Localization and Map Builduing.
TID. : Tag Identifier.
TF : Transform.
URDF. : Unified Robot Description File.
WMR :Wheeled Mobile Robot.
XML: eXtensible Markup Language.

GENERAL INTRODUCTION

Among the most common mobile robot navigation strategies, a large amount of effort has been put on navigation strategies where vision sensors are employed as key perception tools for a robot to realize its location. Without any doubt , these navigation schemes provide satisfactory results in terms of robot’s pose estimation capability. Often, the performance is even better when sensor fusion techniques are used for such navigation systems. While vision-based navigation systems have significant advantages for robot’s localization, they often suffer from restrictive operating environments, deploying complex image processing techniques, dedicated hardware, and a prohibitive computational complexity. For example, they fail to operate mobile robots which are deployed for night navigation. Moreover, it is often hard to use vision information as a navigation means as opposed to robot localization one.

The rising prominence of guiding mobile robots in many real-life applications, requires the development of a new generation of indoor navigation systems.

The primary objective of this project is to design and implement modular navigation (or motion control) approach for mobile robots to navigate in indoor environment with high accuracy and robustness. A modular solution is presented for the robot navigation problem, taking advantage of the RFID technology. The proposed control approach mainly generate control actions with the help of an RFID system deployed in the robot’s operating environment.

Nevertheless, the proposed navigation strategies are not meant to substitute vision-based robot navigation systems, rather, they might be regarded as promising alternatives where vision systems fail to operate, night navigation environments, for example.

This report is organized as follows. Chapter 01 Background where RFID approach, mobile robot navigation and Hardware in the loop simulation are explained. Chapter 02 System Architecture and design is a general picture and description of the RFID based navigation system. Chapter 03 System Implementation shows all the steps needed for a successful navigation. First, an RFID application is used by the robot to communicate with the RFID tags. The ROS based simulation using Gazebo is configured such that mapping, localization and navigation are presented using Rviz which is the visualization tool. Chapter 04 Results and discussion describes the main results, issues related either to the RFID subsystem or the simulation part.

CHAPTER 01: BACKGROUND

1.1.RFID

1.1.1. definition

RFID is an acronym that stands for radio frequency identification which is a technology used to store and retrieve data remotely in memory elements that are called tags.

RFID tag is a small contact-less storage element that is composed of a ROM / EEPROM memory, coils, and circuitry for providing power and access to the digital data stored in the tag.

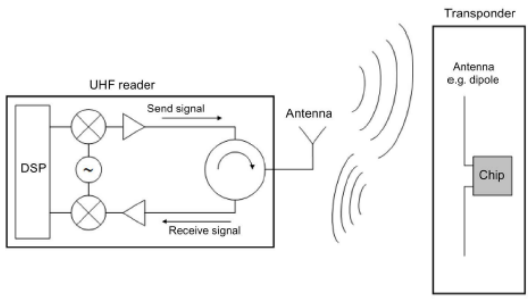


Figure 1.1:Structure of an UHF RFID system[20].

Most RFID tags are passive, i.e. they draw power from the electromagnetic field generated by the reader’s antenna. But there is also active tags which are tags that has a battery to power its circuitry.

RFID systems can also be classified according to their operating frequency (I)LF-low frequency: 30-500 KHz-, (II) HF -High frequency: 10-15 MHz-,(III) UHF -Ultra high frequency: 400-1000 MHz- where UHF are the highest range tags.

The tags used in this work are UHF passive tags, the most suited for robotic application since they are cheap (around 20 dollar cents or less) and do not require any maintenance.

1.1.2. Why RFID

RFID represents a technological advancement in AIDC (Automatic Identification and Data Capture) because it offers advantages that are not available in other AIDC systems such as barcode reading. RFID offers these advantages because it relies on radio frequencies to transmit information rather than light. The use of radio frequencies means that RFID can communicate :

- without line of sight, as radio waves can penetrate many materials.
- At greater speed, as a result, many tags can be read quickly.

- over greater distances, since many radio technologies can transmit and receive signals

1.1.3. Applications

RFID technology is used in many applications like the biometric passport but the main fields of applications are :

- Logistics and assets management.
- Access control and object tracking.
- Augmented reality and interactive gaming.

1.2.Robotics

1.2.1. definition

Mobile robots are physical systems which perform tasks in their environment and are not fixed to one physical location. Nowadays Different applications demand accurate navigation procedures for mobile robot, among them industrial and service applications.

Robot navigation means the robot ability to determine its own position in its reference frame and then to plan a path toward some goal locations. In order to navigate in its environment, the robot requires representation, i.e. a map of the environment and the ability to interpret that representation.

1.2.2. Navigation

Generally the mobile robot has to answer the following three questions :

- Where am I?
- Where am I going?
- How do I get there?

In order to answer these questions, the robot has to [2] :

- Handle a map of its environment.
- Self-localize itself in the environment.
- Plan a path from its location to a desired location.

An autonomous robot navigation system has traditionally been hierarchical , the following diagram show the closed control loop for a successful autonomous navigation :

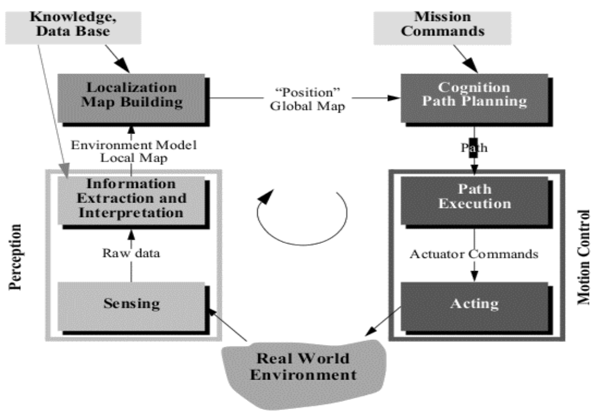


Figure 1.2: Control loop for autonomous mobile robot.[2]

1.2.3. Perception

The first action in the control loop is perception of the robot itself and its environment, which is done through proprioceptive and exteroceptive sensors. Proprioceptive sensors capture information about the self-state of the robot; whereas, exteroceptive sensors capture information about the environment. The types of sensors being used on mobile robots shows a big variety. The most relevant ones can be briefly listed as encoders, gyroscopes, accelerometers, sonars, laser range finders, beacon-based sensors and vision sensors.

1.2.4. Localization

The goal for an autonomous robot is to be able to construct (or use) a map or floor plan and to localize itself in it.

The problem of robot localization consists of answering the question: Where am I?. From the robot's point of view, this means that the robot has to find out its location relative to the environment. And, this poses difficult challenges because of the inaccuracy and incompleteness of the sensors and actuators. The robot must also have a representation of its belief regarding its position on the map. Where the design questions for belief representation are: Does the robot identify a single unique position as its current position?, or Does the robot describe its position in terms of a set of possible positions? If multiple possible positions are expressed in a single belief, How are those multiple positions ranked?

Other localization methods include the use of passive objects in the environment such as landmark-based navigation, positioning beacon systems, and route-based localization strategies.

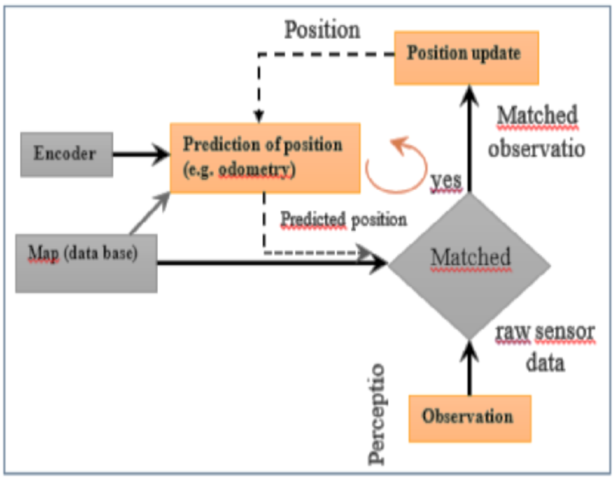


Figure 1.3: General schematic for mobile robot localization.[2]

1.2.5. Path planning

Path planning can be defined as searching a suitable path in a map from one place to another, without colliding with any obstacles. The Robot motion planning can be basically divided into two main categories: (I) local path planning and (II) global path planning:

In global path planning, the environment is known in advance and the terrain is static or the obstacles are known in advance. Hence, the path planning algorithm is able to make a complete map of the environment form the start point to the goal even before the robot starts motion.

On the other hand, in local path planning, the environment is completely unknown to the mobile robot; i.e. the environment is dynamic and unstructured or the obstacles are not known in advance. In such a situation, the robot needs to gather information about the environment in real time and update its control laws so as to achieve this.

Sample algorithms for path planning are:

- Dijkstra's algorithm.
- A*.
- D*.
- Artificial potential field method.
- Visibility graph method.

Path planning algorithms may be based on graph or occupancy grid

1.2.5.1. Graph methods

Method that is using graphs, defines places where robot can be and possibilities to traverse between these places. In this representation graph vertices define places e.g. rooms in building while edges define paths between them e.g. doors connecting rooms. Moreover each edge can have assigned weights representing difficulty of traversing path e.g. door width or energy required to open it. Finding the trajectory is based on finding the shortest path between two vertices while one of them is robot current position and second is destination

1.2.5.2. Occupancy grid methods

Method that is using occupancy grid divides area into cells (e.g. map pixels) and assign them as occupied or free. One of cells is marked as robot position and another as a destination. Finding the trajectory is based on finding shortest line that do not cross any of occupied cells

1.2.6. Motion control

A mobile robot needs locomotion mechanisms that enable it to move unbounded throughout its environment. But, there are a large variety of possible ways to move; so, the selection of a robot’s approach to locomotion is an important aspect of mobile robot design. In locomotion, the environment is fixed and the robot moves by imparting force to the environment.

Locomotion share the same core issues of stability, contact characteristics and environmental type [2] :

Stability: number and geometry of contact points, canter of gravity,static/dynamic stability, inclination of terrain.

Characteristics of contact: contact point/path size and shape, angle of contact, friction.

Type of environment: structure, medium (e.g. water, air, soft or hard ground).

In mobile robotics, we need to understand the mechanical behavior of the robot both in order to design appropriate mobile robots for tasks and to understand how to create control software for an instance of mobile robot hardware.

1.3. Technology and terminology

1.3.1.1. ROS

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

The advantage of using ROS over traditional robotic development methods is the ability to share and reuse libraries and packages in another platform in addition to the modularity and encapsulation provided by this platform which allows the system to function even if a subsystem fails.

1.3.1.2. Python

Python is an interpreter, interactive, object-oriented programming language which combines remarkable power with very clear syntax.

Furthermore is extensible in C or C++, has interfaces to many system calls , libraries and various window systems.

1.3.1.3. XML

Extensible Mark-up Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.

1.3.1.4. MySQL

MySQL is an open source DBMS that is one of the best and most robust relational DBMS available. It is scalable , well documented and available on almost any operating system**.**

1.3.1.5. Hardware in the loop Simulation

Hardware In the Loop is a form of real time simulation. FILs differs from real time simulation by the addition of a real component in the loop [11].

HILS is a technique frequently applied in computer system design, especially for embedded systems and control systems design.

1.4. Summary

This chapter provides a brief introduction to the main concepts and jargon related to autonomous mobile robots, RFID technology and Hardware in the loop simulation. Starting from the main steps and key concepts of a successful navigation followed by descriptions and definitions of the used technologies and terminologies.

CHAPTER 02: SYSTEM ARCHITECTURE AND DESIGN

Robuteur/UML is the robot proposed by the CDTA. i.e. the institution where the development took place, It has been the robotic platform for many researches and publications [7] and still is the object of study to many other researches, we may refer to the research about “RFID task planning” being conducted by a graduate IGEE student Salim Djazairi in order to get his PHD.

In this chapter we will explain the design and describe the architecture of the overall system as of its components that has been selected in order to build an RFID platform that is capable of integrating the ROS system to test it with robot and has the ability to be adaptable to fit any future work on Robuteur/UML.

2.1.System architecture

The system could be divided into five main components

(I) **Embedded platform** refers to the SBC(single board computer) where that all the other sub-systems are connected to, it acts as the switch that connects all computer systems and as the executing environment of the RFID application.

(II) **Robuteur/UML** refers to the physical robot, the embedded computer wired to it, and the electronic boards (micro controller boards), its main functionality is to connect hardware (sensors and actuator) with the high level software interface of ROS.

(III) **RFID platform** refers to the RFID reader and application responsible for the bridging and interfacing of the RFID reader with the robot through a ROS(Robotic Operating System) node.

(IV) **Simulation station** refers to the computer platform and the ROS application which are acting as the central brain of the robot.

(V) **Cloud** refers to the remote database and the API used for

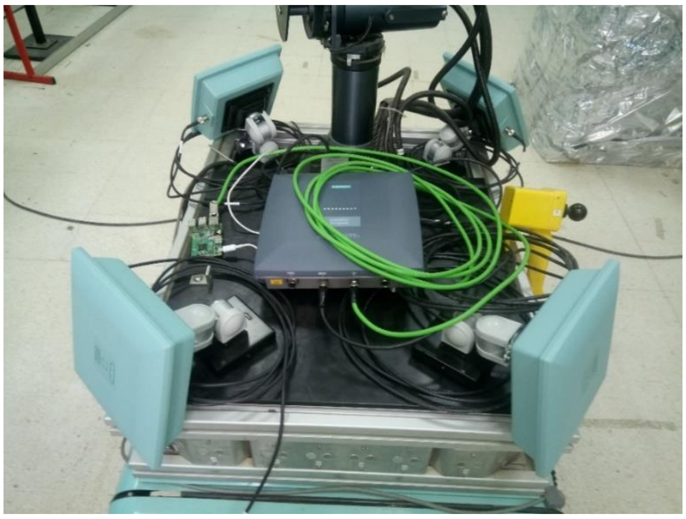


Figure 2.1: The overall system[21].

notification(SMS).Which are run from the embedded platform.

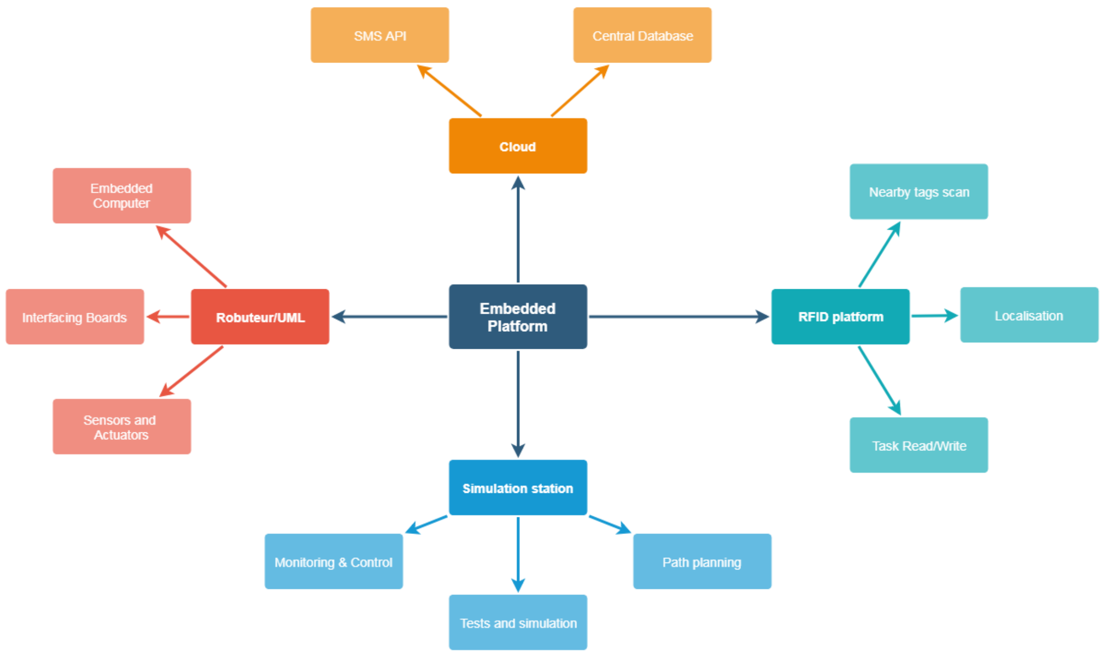


Figure 2.2: Overall system architecture

2.1.1. Embedded platform

The embedded platform is an embedded SBC (single board computer).This board acts as :

- Execution platform for the RFID application.
- A network link between the RFID reader using Cable, Robuteur/UML using Wi-Fi , and the Simulation station also using WI-FI.
- A testing and troubleshooting platform for the whole system because of it extensibility(WI-FI, GPIO, USB ,Bluetooth …etc.), flexibility(OS install on SD-CARD), ease of use i.e. no special hardware required to program it, and availability of software(ROS, python, Bash, Node-red , …etc).
- A data logging platform.

2.1.2. Robuteur/UML

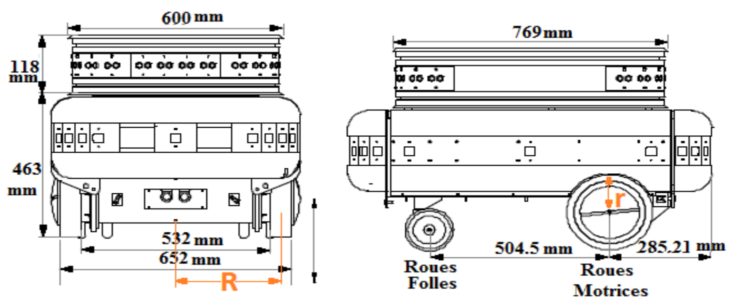


Figure 2.3: Robuteur chassis dimensions [22]

Robuteur/UML is the robot proposed by the CDTA as a platform for testing and validation.

The main features and functionalities of the Robuteur/UML are

- Communication with other robot components that uses the ROS as the simulation station , and the RFID platform.
- Real-time OS with synchronization and scheduling capabilities provided by SynDEx.
- Communication with low level and hardware through MPC555 microcontroller boards using the CAN bus.
- The MPC555 based boards are responsible of interacting with the hardware (ultrasonic ranging detector, infrared sensors , motors …).

2.1.3. RFID platform

The RFID system consists of the RFID reader which requires antennas to send radio waves and tags to store the data.

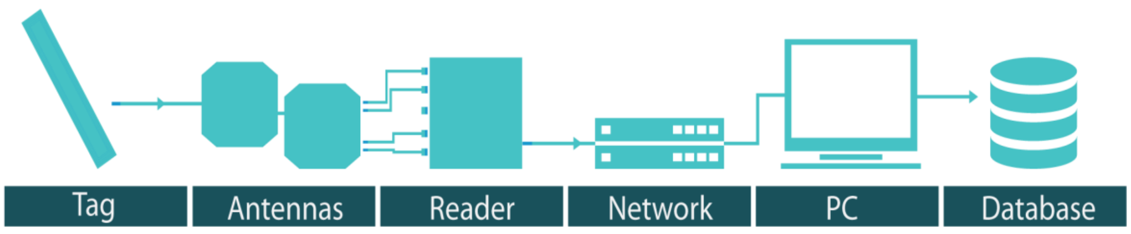


Figure 2.4: RFID hardware configuration in IT environment

A. The reader

The RFID reader used in this project is the Simatic RF 680R. The main functionality of the reader is to communicate with antennas and to give a higher level interface for interacting with Tags. The system that we are using has three interfaces WBM (web interface) , XML(IT) and OPC (automation). In our work, we have used the WBM interface when configuring the reader and adjusting antennas, while we used the XML interface in the programming part.

B. **The antennas**

The RFID antennas used in this project are the Simatic RF 650A. The antennas have been deployed over the robot in a way to give a 360 degrees coverage of its surrounding environment.

C. **The tags**

Figure 2.5: Passive RFID tag [14]

The RFID tags used in this project are the Simatic RF630L Smart-label variant. Also called transponders, these are the memory elements where the data is being stored.

2.1.4. Simulation station

This stands for the computer used to visualize, monitor, and control the robot using Gazibo and Rviz. This is also the platform where the path planning and all the robotics decisions occur.

It has the robotic and simulation packages and software

- ROS stands robotic operating system which is the meta operating system used to inter connect all the robot components and implement the robotics algorithms.
- Gazebo is a robot simulation and modeling platform used to monitor and test the reobot.
- Rviz is a robot visualization tool.

It is connected to the other sub-systems via the embedded platform (Raspberry pi) using a WI-FI network.

2.1.5. Cloud

The cloud consists of the services that either require internet as the SMS API or the components that are not mandatory for a robot to function correctly as the remote database (central DB).

2.2.Simulation Setup

In this section , we will explain all the steps needed to configure the simulation model of the robot and the control package as well. Also , a clear description of the navigation stack which is a set of ROS nodes and packages which are used for path planning and obstacle avoidance is given.

2.2.1. Robot 3D Model

At first, a 3D model of the Robot must be created. It can be done using one of the CAD tools such as Solidworks, AutoCAD, Blender and so on. This model is then transformed to an URDF file. Another technique is to write directly an URDF file that describes the kinematic and Dynamical model of the Robot. This URDF file is an XML based file which specifies the following features of the Robot[1]:

- The kinematic and dynamic description of the robot.
- The visual representation of the robot.
- The collision model of the robot.

Basically The description of the robot (URDF file) consists of a set of links (parts), elements, and a set of joint elements, which connect these links together.

The 3D model of the Robot Robuter/ULM has been previously built (at CDTA) using Solidworks and then an URDF model (Unified Robot Description Format) has been extracted, the one that is suitable to be processed in Robot operating system (ROS). This URDF file has been updated according to the simulation requirements.

2.2.2. Gazebo 3D model

To make The URDF file of Robuter/ULM accessible by gazebo simulator, gazebo references, plugins and appropriate parameters must be added. The following will clarify all the steps followed.

2.2.2.1. Gazebo reference

We use gazebo reference in the URDF file to define and add each link (part).Also, we can specify different parameters such as color , gravity and so on.

2.2.2.2. Transmition Tags

Transmition tags are elements that describe the relation between actuators and joints. Mainly , they are used such that we can control our robot joints using ROS controllers. We have to specify the hardware interface for the joint and the actuator as well.

2.2.2.3. gazebo plugins

Gazebo plugins give our URDF model greater functionalities and can add ROS messages and service calls for sensor output and motor input. Here are the plugins added to the URDF file :

a-Gazebo ROS. control

After adding the transmission tags , we have to add the gazebo_ros_control plugin in the URDF file to analyze the transmit tags and assign the appropriate hardware interfaces and control manager.

b – Differential Drive Controller

In order to move the robot in Gazebo, we have added a Gazebo ROS plugin file called libgazebo_ros_diff_drive.so to get the "Differential drive" behavior in the robot.

c- Laser

We have used the Gazebo ROS plugin file called libgazebo_ros_laser.so to simulate the Laser.

2.2.3. Creation of Meta-Package

The ROS community has established conventions for packages that define robots, their ROS bindings, and the gazebo integration. we have followed all of these conventions in our work. we have created the following packages :

● **Gazebo :** this package holds all the launch and world files needed for the gazebo simulator and different stages of the simulation.
● **Description :** this package contains the URDF file and the meshes exported from Solidworks.
● **Control :** this package has a configuration file and a launch file for ROS controllers and their interface for the joints of our robot.

2.2.4. Configuring the controllers

In order to control each joint, we have to specify a controller compatible with the specified transmission tag defined previously in the URDF file for each joint. Connecting these controllers mainly requires writing a configuration file [appendix A1] which contains three types of controllers that ROS provides :

● **Joint state controllers :** these controllers publish the states of the robot’s joints.

● **Joint velocity controllers :** these controllers are used for sending the right and left velocities to the mobile base , specifically for the wheels joints.
● **Joint position controllers :** these controllers are used for the joints of the arm mounted on the mobile base.

2.2.5. Navigation Stack Setup

The Navigation stack is a **set of ROS nodes and algorithms** which are used to autonomously move a robot from one point to another , avoiding all obstacles the robot might find in his way. The ROS Navigation package comes with an implementation of several navigation related algorithms such as SLAM, A *(star), Dijkstra's, AMCL, and so on, which can directly be used in our application.

The navigation stack will take as input: the current location of the robot, the desired location the robot wants to go, the odometry data of the robot, and data from a sensor such as laser. In exchange, it will output the necessary velocity commands to the base controller in order to move the robot to the desired location.

To sum up, we can say that the main objective of the Navigation stack is to move a robot from point A to point B , assuring it will not crash against obstacles or get lost in the process.

2.2.5.1. Hardware requirements :

The ROS navigation stack is designed as generic.There are some hardware requirements that should be satisfied by the robot. Following are the requirements:
- The Navigation package will work better in differential drive and holonomic. Also, the mobile robot should be controlled by sending velocity commands in the form :
* x,y (linear velocity)
* z (angular velocity)
-The robot should mount a planar laser somewhere around the robot. It is used to build the map of the environment.
- The Navigation stack will perform better for square and circular shaped mobile base.
Following are the basic building blocks of the Navigation stack. We can see what are the purposes of each block and how to configure the Navigation stack for a our robot :

According to the shown diagram , we must provide some functional blocks in order to work and communicate properly with the navigation stack. Following are brief explanations of all the blocks which provided as inputs to the navigation stack :

• **Odometry source** : Odometry data of a robot gives the robot position with respect to its starting position.Our odometry source is the wheel encoders.

• **Sensor source** : sensors are used for two tasks in navigation , One for localizing the robot in the map or to detect obstacles. We have used laser scan and wheel encoders.

• **sensor transforms/tf :** the data captured by different sensors of the robot must be referenced to a common frame of reference (usually the base_link) in order to compare data coming from different sensors. The robot should publish the

relationship between the robot coordinate frame using ROS tf.
• **base_controller :** The main function of the base controller is to convert the output of the Navigation stack, which is a twist (geometry_msgs/Twist) message, and convert it into corresponding motor velocities of the robot. We have used the Differential drive controller as the base controller.

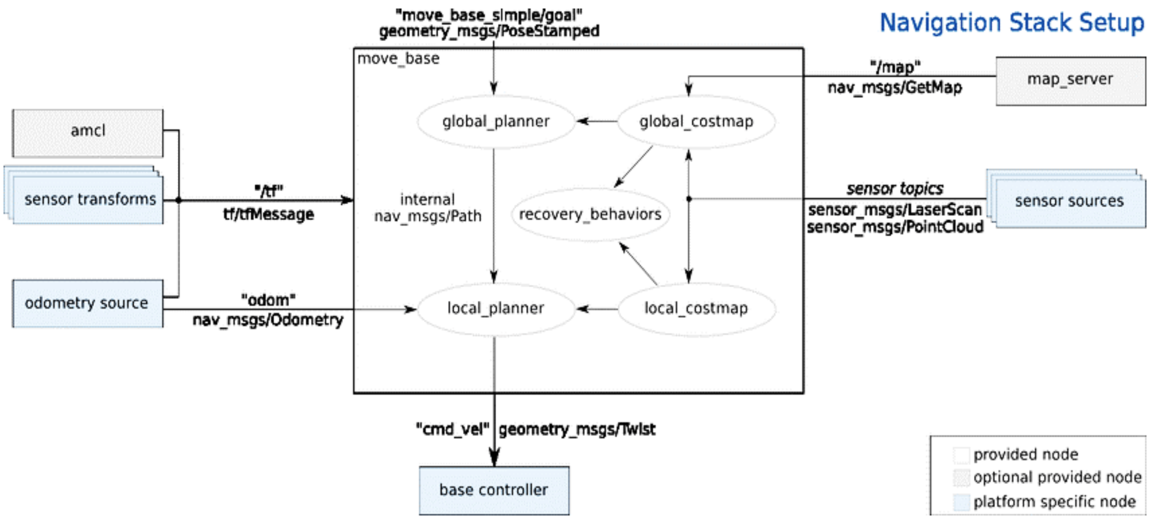


Figure 2.6: Navigation stack architecture [3].

2.2.6. The move base node

This is the most important node of the navigation stack. It is where most of the processing happens.
Basically, The move_base node is an implementation of Simple Action Server which takes a goal pose (position and orientation) as an input. We can send goal position to it using a Simple Action Client node. After the goal pose is received, the move base node links to components such as global_planner and local_planner for global and local path planning, recovery_behavior if the robot is get stuck in some obstacle, global_costmap and local_costmap, and generates the output which is the command velocity (geometry_msgs/Twist), and sends to the base controller for moving the robot for achieving the goal pose.
Following is the list of all packages which are linked by the move base node :

• **Global planner :** used for global path planning by finding the shortest path.
• **Local planner :** This package executes the global path given by the global planner**.**
• **rotate-recovery:** This package helps the robot to recover from a local obstacle by performing a 360 degree rotation.
• **clear-costmap-recovery:** This package is also for recovering from a local obstacle by clearing the costmap.
• **costmap-2D:** This package manages the costmaps of the environment.

The other packages which can be interfaced to the move_base node are :

• **map-server** : Map server package allows us to save and load the map generated by the costmap-2D package.
• **AMCL:** AMCL is a method to localize the robot in map. This approach uses particle filter to track the pose of the robot with respect to the map, with the help of

probability theory. In the ROS system, AMCL can only work with maps which were built using laser scans.

• **Gmapping**: The gmapping package is an implementation of an algorithm called Fast SLAM which takes the laser scan data and odometry to build a 2D occupancy grid map.

2.3.Summary

In this chapter we described system architecture, main components, and the different blocks of the navigation stack.

CHAPTER 03:SYSTEM IMPLEMENTATION

In this chapter we will see the system implementation.

3.1.Simulation using navigation stack

In this section , we will see the three main steps needed for a successful navigation as described previously. Also, we will see all the configurations and launch files needed for each step.The following diagram simplify the whole process and clarify the work of the navigation stack.

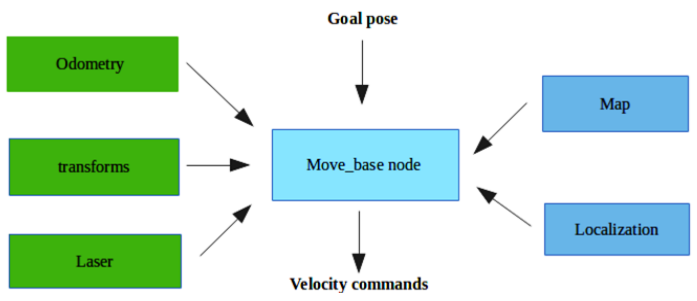


Figure 3.1: Navigation stack functionalities and operation.

3.1.1. Map Building using SLAM (simultaneous localization and map building)

Mapping

Mapping is the process of creating a spatial model of the environment surrounding the robot using its sensors information. The map is then used for localization and navigation. In order to build the map using ROS, the following commands are executed on the Linux command line

$ roscore

$ roslaunchRobuterULMgazebo.launch[Annex1]

First, we start the robot simulation in a simple created environment by executing the above launch file.

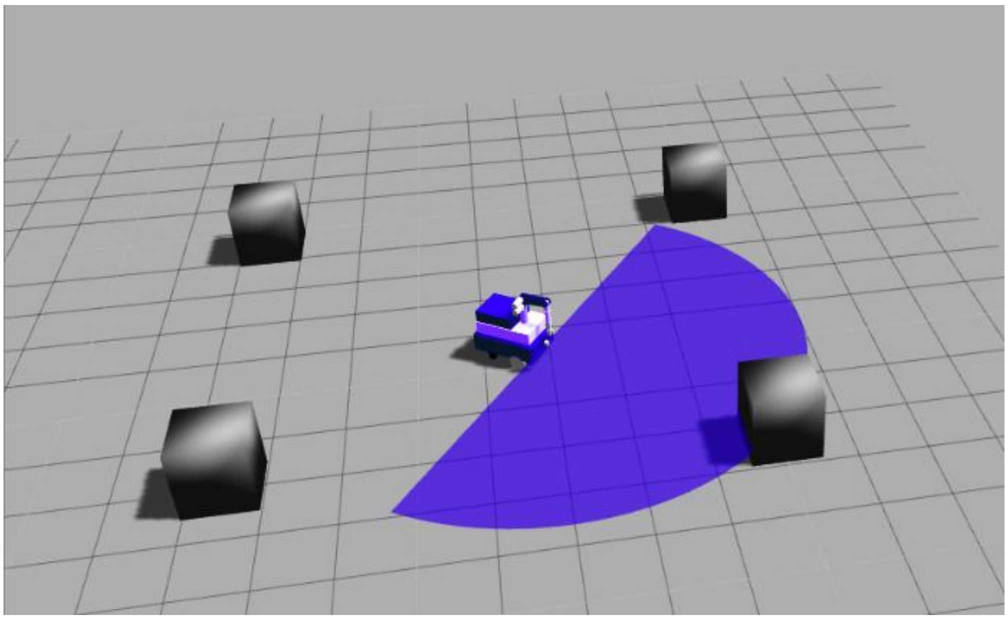


Figure 3.2: Gazebo simulator.

$ roslaunchRobuterULMgmapping.launch :

By launching the slam _gmapping node, the mapping process start.This node will use Laser scan data and odometry topics. Also , this node requires two transformations one from laser to base link and from base link to odom link. These transformations are displayed in the TF topic. The slam gmaping node publishes the map in the map topic.

$rosrunRobuterULM teleoperation.py :

by running this node the robot can move using the keyboard and explore the whole environment.

$ roslaunchRobuterULMrviz.launch :

Rviz is used to visualize the mapping process by adding specific displays (laser scan map, TF topics)

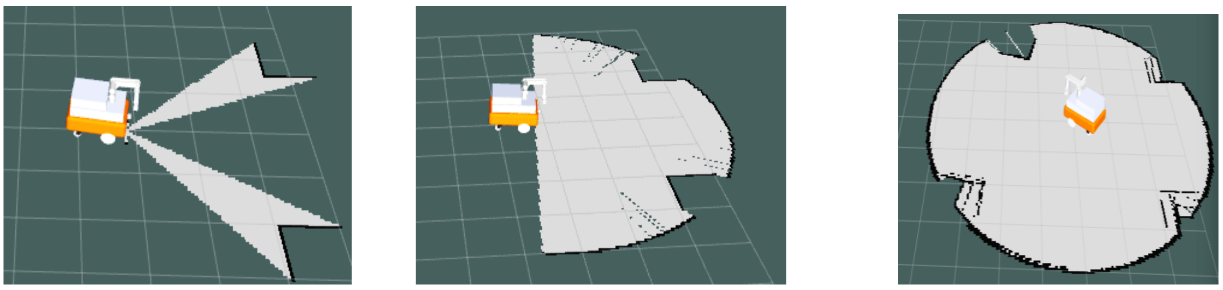
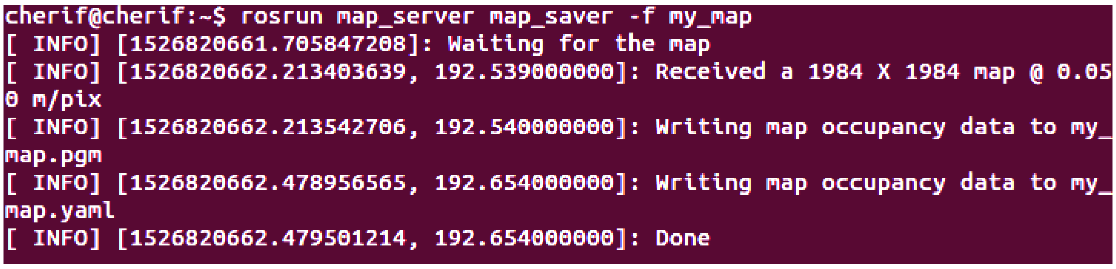


Figure 3.3: Step by step map building process

.

$ rosrun map server map saver -f mymap :
After scanning the whole environment we can save the map created using map server package. after executing the above command we can save our map :



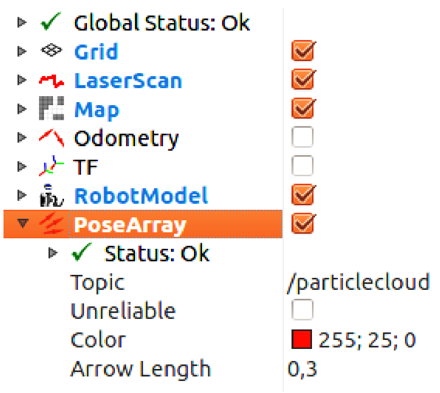
```
cherif@cherif:~$ rosrun map_server map_saver -f my_map
[ INFO] [1526820661.705847208]: Waiting for the map
[ INFO] [1526820662.213403639, 192.539000000]: Received a 1984 X 1984 map @ 0.05
0 m/pix
[ INFO] [1526820662.213542706, 192.540000000]: Writing map occupancy data to my_
map.pgm
[ INFO] [1526820662.478956565, 192.654000000]: Writing map occupancy data to my_
map.yaml
[ INFO] [1526820662.479501214, 192.654000000]: Done
```

Figure 3.4: Map saving from the CLI.

Once this process is done, the two files mymap.pgm and mymap.yaml are created. The PGM file is the map's image whereas the YAML file is the description of this map. This YAML file can be later provided by map_server node for the localization and navigation.

3.1.2. Adaptive Monte Carlo Localization (AMCL) localization

After successfully building the map, it can be used for navigation. But before performing navigation , we need a localization algorithm such that we can know the position and orientation of our robot in the map. Localization tells the robot where it is in relation to the environment. AMCL localization uses odometry, laser data and a static map.
In addition to the roscore , gazebo launch file and teleopeartion node , the following commands are used for the robot localization :
$ roslaunchRobuterULMamcl.launch[Annex1] :
This launch file will run two nodes. The first node provided by the Amcl package which does the localization. The node is highly configurable and it uses the particle filter based localization method.Localization needs the map server node to get the mymap.yaml file.
$ rosrunrvizrviz :
Now , to show that our localization algorithm is working , we will add the topics needed for the localization which are the laser scan , map and the **pose array display** which is published by the AMCL node. The following figure shows the localization.



✓ Global Status: Ok	
Grid	✓
LaserScan	✓
Map	✓
Odometry	☐
TF	☐
RobotModel	✓
PoseArray	✓
✓ Status: Ok	
Topic	/particlecloud
Unreliable	☐
Color	🟥 255; 25; 0
Arrow Length	0,3

Figure 3.5:Rviz configuration pop-up screen.

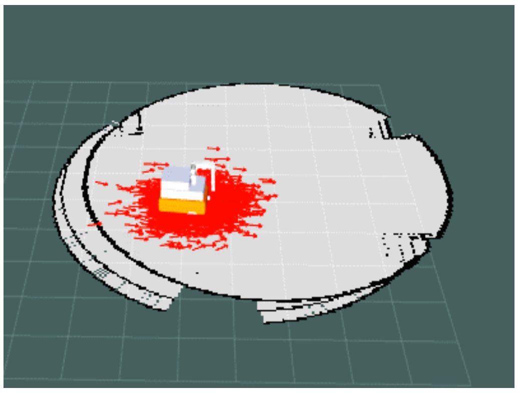


Figure 3.6: Amcl particle cloud

3.1.3. Autonomous Navigation

Once mapping and localization are successfully done then navigation can be easily achieved. Package “**move_base”** is used to accomplish autonomous navigation.
This package provides the move base node that uses localization information. This package also maintains two cost-maps each for the local and global planner. The global planner is based on A∗ search algorithm while the local planner used the dynamic window approach.
Four yaml files are used by the move base node which are : **common cost parameters**, **local cost-map**, **global cost-map**, and **base local planner**. These files are used for path planning to help autonomous navigation. We will go through each one of them in details and understand their functionalities.
- Cost-map Configuration (global and)

Our robot will move through the map using two types of navigation—global and local

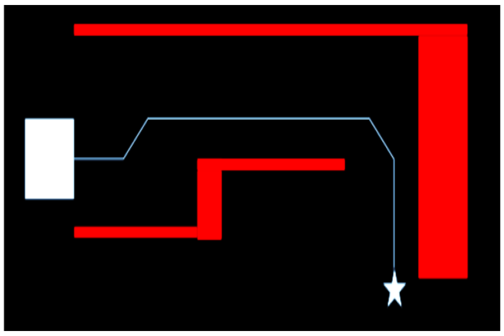


Figure 3.7: Global navigation using global cost-map.

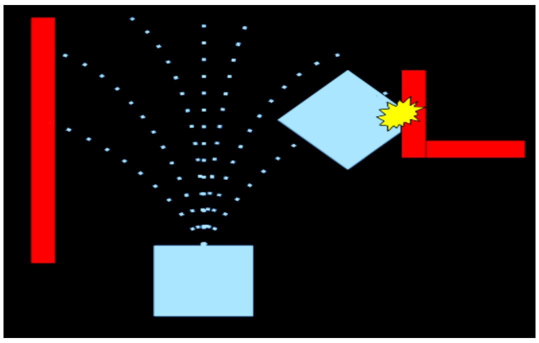


Figure 3.8:Local navigation using local cost-map

Cost-maps are used to save all the information of our map based on the occupancy grid map built previously and user specified inflation radius.The cost-maps have parameters to configure the behaviors, and they have common parameters as well, which are configured in a shared file. Configuration basically consists of three files where we can set-up different parameters. These files are as follows:

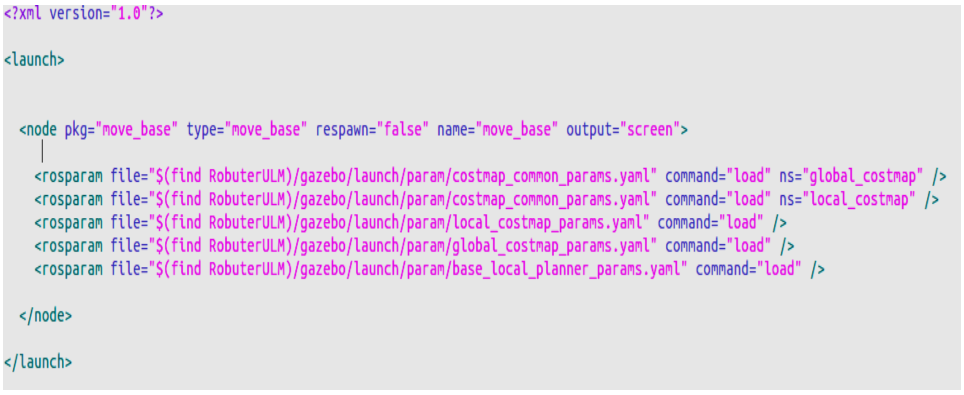
costmap common parameters
set obstacle_range to 2.5
set raytrace_range to 3.0
Fix footprint position to [[-0.4, -0.4], [-0.4,0.4], [0.4, 0.4], [0.4, -0.4]]
set inflation_radius to 0.50
set observation_sources to the value of laser_scan_sensor
global costmapparamters
set global_frame to path /map
set robot_base_frame to path /base_footprint
set update_frequency to 1.0
set spublish_frequency: 0.0
set static_map to true
rolling_window: false
local cost-map parameters

set global_frame to path /odom
set robot_base_frame to path /base_footprint
set update_frequency to 1.0
set publish_frequency to 2.0
set static_map to true
set rolling_window to false
set width to 6.0
set height to 6.0
set resolution to 1

Once we have the cost-maps configured, it is necessary to configure the base local planner. The base local planner is used to generate the velocity commands to move the robot :

base local planner parameters
set max_vel_x to 1.5
set min_vel_x to 0.5
set max_rotational_vel to 1.0
set min_in_place_rotational_vel to 0.4
set acc_lim_th to 3.2
set acc_lim_x to 2.5
set acc_lim_y to 2.5
set holonomic_robot to false
sim time : 1.0
sim granularity: 0.0325
yaw_goal tolerance : 0.1
xy_goal_tolerance : 0.1

Now, we will create a launch file for the move base node with the four yaml files that have been configured as its parameters.



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

<launch>


  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">

    <rosparam file="$(find RobuterULM)/gazebo/launch/param/costmap_common_params.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find RobuterULM)/gazebo/launch/param/costmap_common_params.yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find RobuterULM)/gazebo/launch/param/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find RobuterULM)/gazebo/launch/param/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find RobuterULM)/gazebo/launch/param/base_local_planner_params.yaml" command="load" />

  </node>

</launch>
```

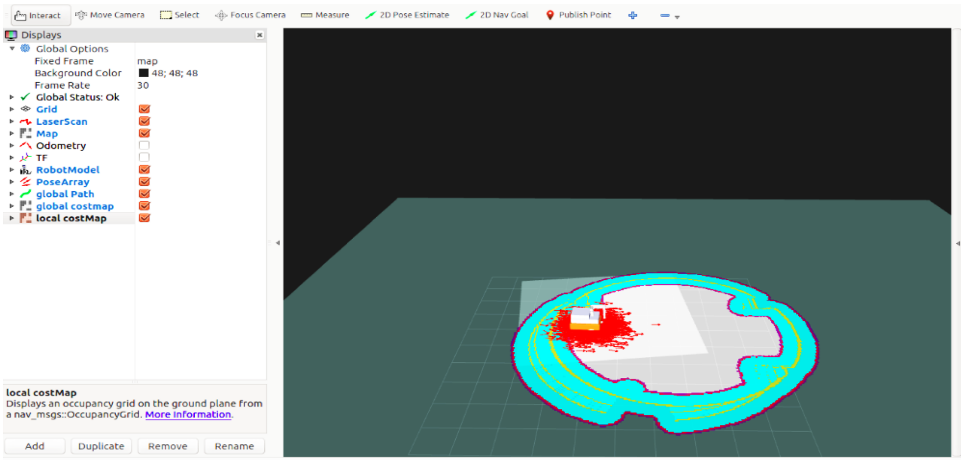
Figure 3.9:move_base launch file

Now , the previous launch file is tested by sending our robot to a specific location using the 2D Nav goal button of Rviz using the following commands :
$ roscore   // to start roscore node
$ roslaunchRobuterULMgazebo.launch

```
$ roslaunchRobuterULMamcl.launch
$ roslaunchRobuterULMmove_base.launch
```

Here, Rviz is also used to show the navigation related topics such as global and local paths , the global and local costmaps.

```
$ roslaunchRobuterULMrviz.launch
```



Global Options	
Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	30
Global Status: Ok	
Grid	✓
LaserScan	✓
Map	✓
Odometry	☐
TF	☐
RobotModel	✓
PoseArray	✓
global Path	✓
global costmap	✓
local costMap	✓

local costMap
Displays an occupancy grid on the ground plane from a nav_msgs::OccupancyGrid. More Information.

Figure 3.10: Rviz popup screen to start the robot navigation.

Now, we just perform navigation in the map that we have created using the rviz parameter "2D Nav Goal" which is used to create an optimum path for the robot to reach the needed goal. Figure below shows the step by step path execution for a desired goal in known generated map.

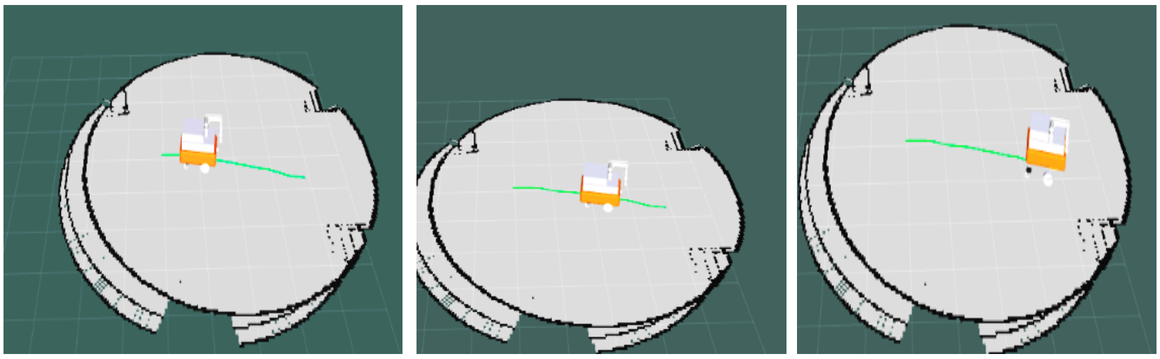


Figure 3.11: Step by step path planning execution.

3.2.RFID platform

In this section we will describe the main system layout, the application design, the database schema, and the tag naming specification.

3.2.1. Main system layout

The left figure below represents the originally planned four-antennas design. This design has the advantage of covering all the field around the robot. But due to the fact the antennas require some distance between each other to operate safely this design was changed. The adopted antennas configuration is the one described in right figure below which uses only three-antennas, the drawback of this design is that the robot has a tight blind spot, in the near front of the robot, where tags cannot be identified.

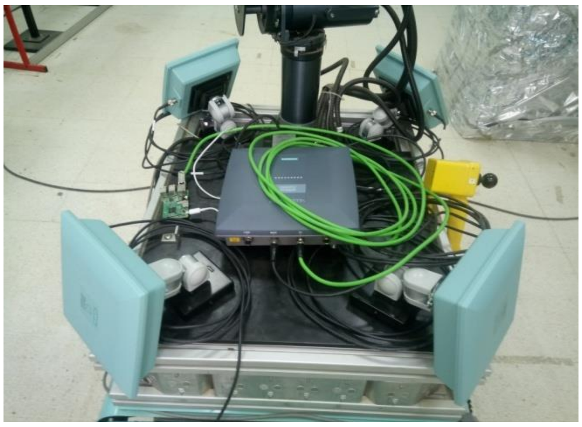


Figure 3.12: Four-antennas setup[21].

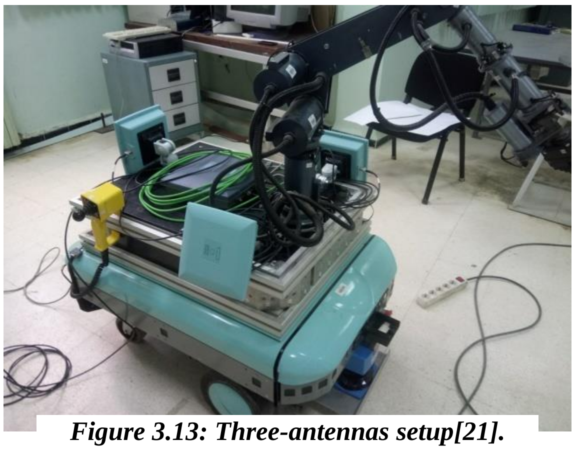


Figure 3.13: Three-antennas setup[21].

3.2.2. Application design

The application was developed using python and MySQL on Raspberry Pi 3 model B. The main functionality of the application is to make a bridge and act like an API between the Robot (ROS) and the exterior world (using Tags) by constantly scanning for nearby tags , reading their content then sending the tags information (RSSI , TID …) to the Robot platform for path and task planning.

3.2.2.1. Flow chart.

The application start by connecting through TCP/IP socket to the reader, when the connection fails it will exit the application and throw the error causing the problem.

After the application connects to the reader, a menu appears containing entries to menus, commands, and the auto mode.

That menus are the entries with “m” at the most left side, when selecting a menu entry the application show us the next menu to choose from.

The commands are entry with either + or – at the most left side, when typing a command entry the application generate the appropriate XML then send it to the reader through TCP/IP connection. When receiving the reader reply the application first it executes the quick action function that reacts based on data stored in the ID tag field for emergency procedures like robot emergency stop, change direction … etc. in

order to avoid collision or danger; Then the application stores the sent command in a file then reads the reply and stores the fetched information in the database.
The information that will be stored are RSSI , tag ID , block memory content, and other information specific to commands themselves (configuration …)

Then the application displays the menu again to select the next action.

From entries there is also the auto scan mode which leads the application to scan repeatedly for observed tag (IDs) then executes the quick action procedure, update database, and send the fetched information to the ROS.

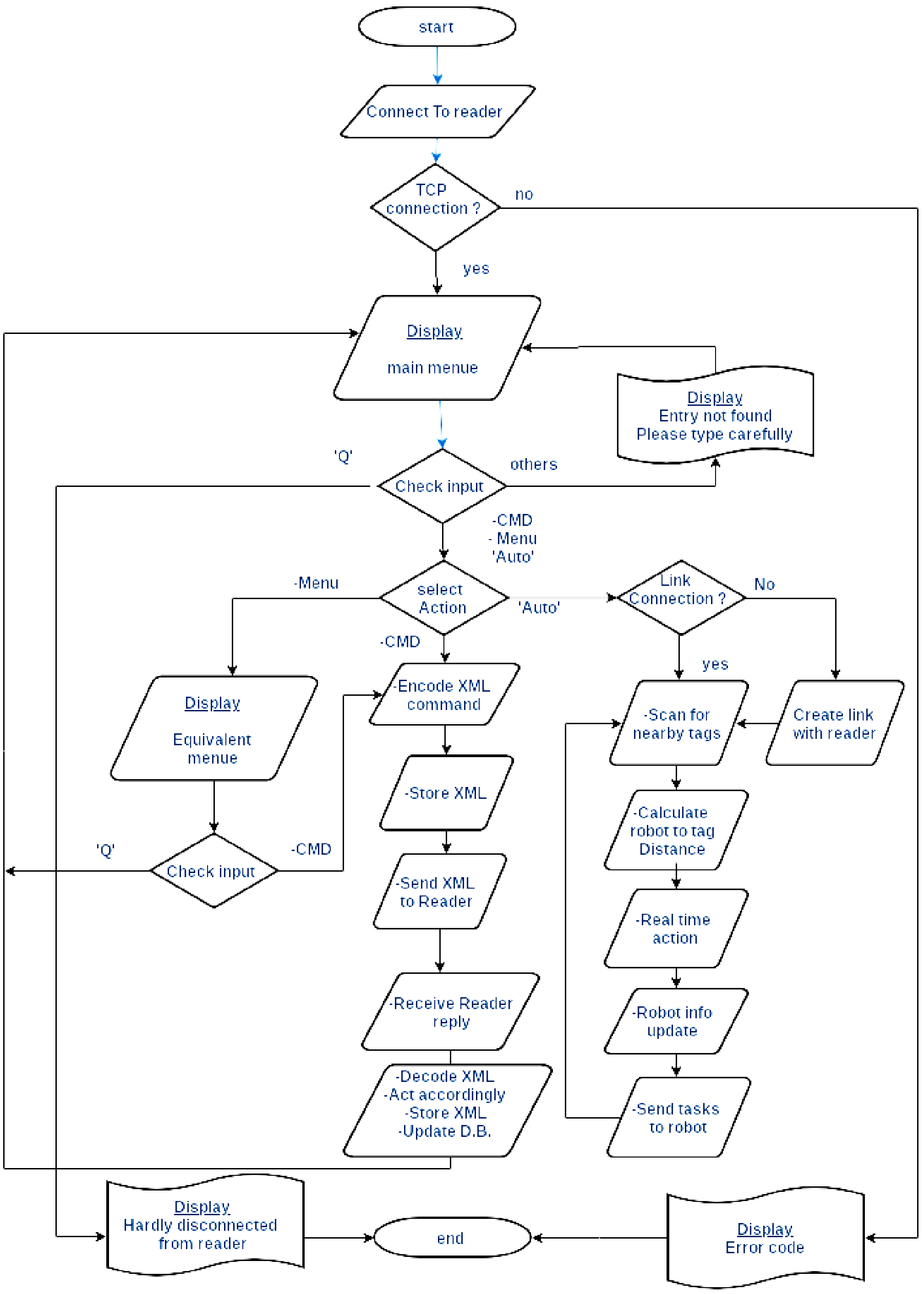


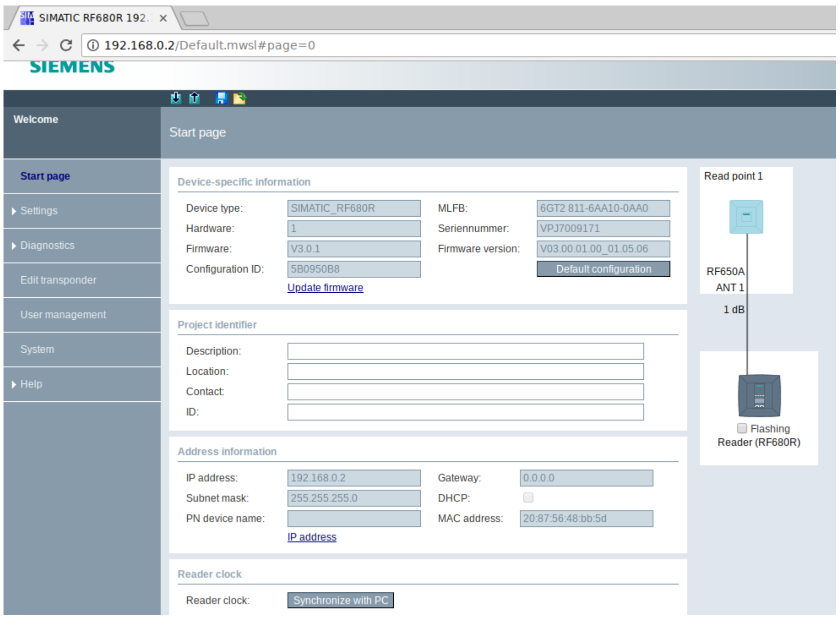
Figure 3.14: The main program flow chart.

3.2.3. RFID reader configuration

In order to use the RFID system we need to configure the reader to operate safely and accordingly.These modification are made using the web interface which is referred by WBM.
Hardware configuration : These configurations help identify equipment (antennas, cables…etc.), services, and Protocols (OPC , XML ,RF …), of the reader.

Algorithmic configuration : These configurations determine how the scan, tag handling algorithms work, we aimed to get the best performance possible(high power ramp, high RSSI delta …etc.).

Tag memory configurations: These configurations aim to define the tag memory structure and tag fields(identifiers for memory blocks).



Device-specific information

Device type:	SIMATIC_RF680R	MLFB:	6GT2 811-6AA10-0AA0
Hardware:	1	Seriennummer:	VPJ7009171
Firmware:	V3.0.1	Firmware version:	V03.00.01.00_01.05.06
Configuration ID:	5B0950B8		

Project identifier

Description:	
Location:	
Contact:	
ID:	

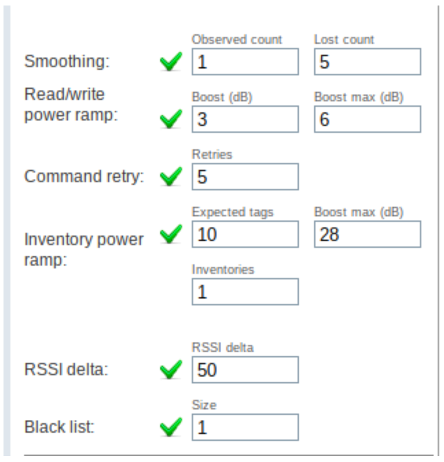
Address information

IP address:	192.168.0.2	Gateway:	0.0.0.0
Subnet mask:	255.255.255.0	DHCP:	☐
PN device name:		MAC address:	20:87:56:48:bb:5d

Reader clock

Reader clock:

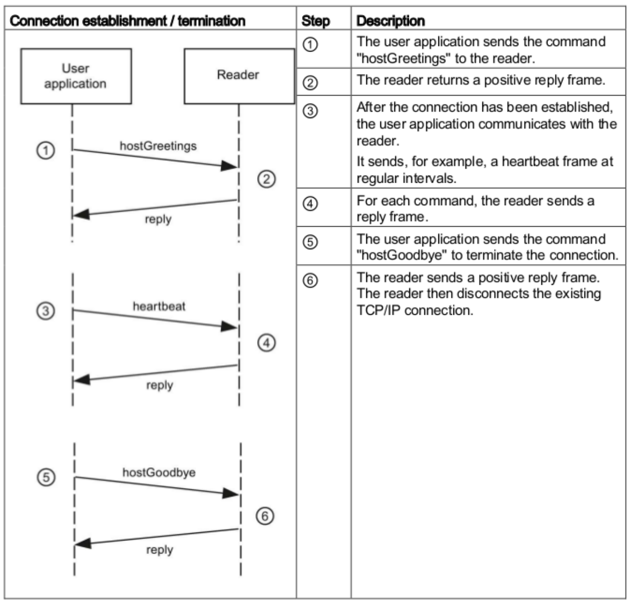
Figure 3.15: RFID hardware configuration interface[21].



Smoothing:	✓	Observed count: 1	Lost count: 5
Read/write power ramp:	✓	Boost (dB): 3	Boost max (dB): 6
Command retry:	✓	Retries: 5	
Inventory power ramp:	✓	Expected tags: 10	Boost max (dB): 28
		Inventories: 1	
RSSI delta:	✓	RSSI delta: 50	
Black list:	✓	Size: 1	

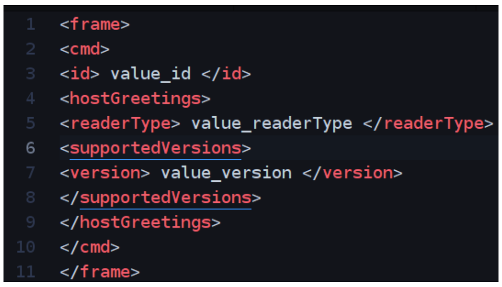
Figure 3.16: Tag scan algorithms interface[21].

The picture below shows how to establish and terminate a connection with the RFID reader.



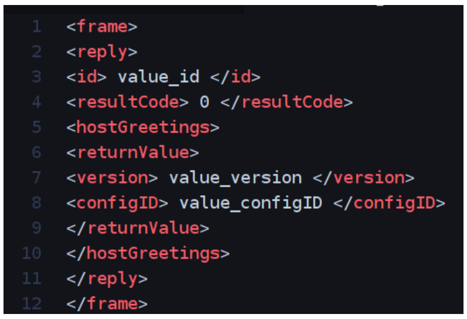
Step	Description
①	The user application sends the command "hostGreetings" to the reader.
②	The reader returns a positive reply frame.
③	After the connection has been established, the user application communicates with the reader. It sends, for example, a heartbeat frame at regular intervals.
④	For each command, the reader sends a reply frame.
⑤	The user application sends the command "hostGoodbye" to terminate the connection.
⑥	The reader sends a positive reply frame. The reader then disconnects the existing TCP/IP connection.

Figure3.19: Reader connection establishment/termination.



```xml
<frame>
<cmd>
<id> value_id </id>
<hostGreetings>
<readerType> value_readerType </readerType>
<supportedVersions>
<version> value_version </version>
</supportedVersions>
</hostGreetings>
</cmd>
</frame>
```

Figure 3.17: hostGreeting command XML

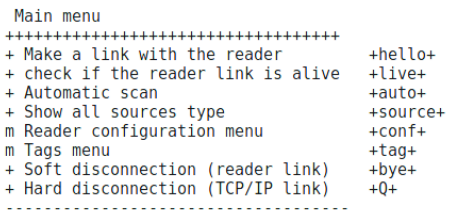


```xml
<frame>
<reply>
<id> value_id </id>
<resultCode> 0 </resultCode>
<hostGreetings>
<returnValue>
<version> value_version </version>
<configID> value_configID </configID>
</returnValue>
</hostGreetings>
</reply>
</frame>
```

Figure 3.18: host Greeting reply XML file

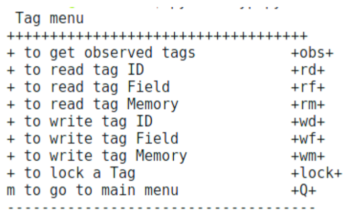
3.2.4. Application features

The auto-scan functionality is invoked by typing “auto” at the main menu.
In addition to the auto-scan functionality a manual mode is available to enable and simplify, debugging, testing, and the execution of commands.
The pictures below show the menu entries of main program where
“+” stand for implemented command
“-” stand for non-implemented command
“m” Stands for Menu entries



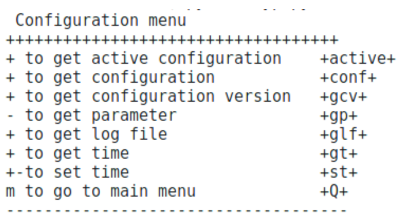
```
 Main menu
+++++++++++++++++++++++++++++++++++
+ Make a link with the reader       +hello+
+ check if the reader link is alive +live+
+ Automatic scan                    +auto+
+ Show all sources type             +source+
m Reader configuration menu         +conf+
m Tags menu                         +tag+
+ Soft disconnection (reader link)  +bye+
+ Hard disconnection (TCP/IP link)  +Q+
------------------------------------
```

Figure 3.20: Main menu entries.



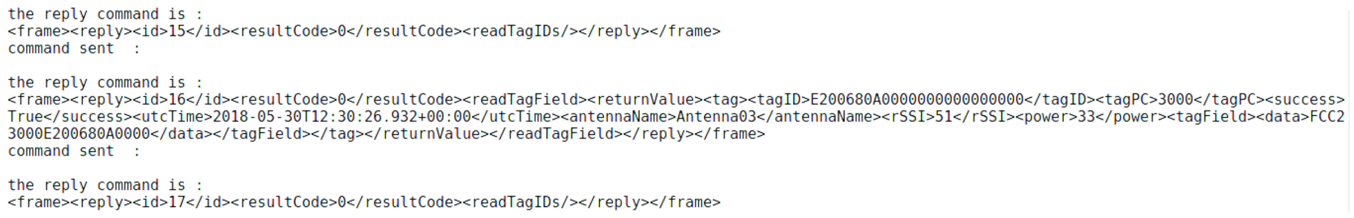
```
 Tag menu
+++++++++++++++++++++++++++++++++++
+ to get observed tags         +obs+
+ to read tag ID              +rd+
+ to read tag Field           +rf+
+ to read tag Memory          +rm+
+ to write tag ID             +wd+
+ to write tag Field          +wf+
+ to write tag Memory         +wm+
+ to lock a Tag               +lock+
m to go to main menu          +Q+
------------------------------------
```

Figure 3.21: Tag menu entries.



```
 Configuration menu
+++++++++++++++++++++++++++++++++++
+ to get active configuration    +active+
+ to get configuration           +conf+
+ to get configuration version   +gcv+
- to get parameter               +gp+
+ to get log file                +glf+
+ to get time                    +gt+
+-to set time                    +st+
m to go to main menu             +Q+
------------------------------------
```

Figure 3.22: Configuration menu entries.



```
the reply command is :
<frame><reply><id>15</id><resultCode>0</resultCode><readTagIDs/></reply></frame>
command sent  :

the reply command is :
<frame><reply><id>16</id><resultCode>0</resultCode><readTagField><returnValue><tag><tagID>E200680A0000000000000000</tagID><tagPC>3000</tagPC><success>
True</success><utcTime>2018-05-30T12:30:26.932+00:00</utcTime><antennaName>Antenna03</antennaName><rSSI>51</rSSI><power>33</power><tagField><data>FCC2
3000E200680A0000</data></tagField></tag></returnValue></readTagField></reply></frame>
command sent  :

the reply command is :
<frame><reply><id>17</id><resultCode>0</resultCode><readTagIDs/></reply></frame>
```

Figure 3.23: Application output in automatic mode.

3.2.4.1. Other features

We have also implemented two other features independently from the Main application

- Programmable SMS messaging(send a message when the reader disconnects).
- Remote database synchronization.

3.2.5. The database schema

The relational databases are organized into 4 tables : (i) “tag” is the table which holds the information related to the tags and the objects they represent (ii) “task” is the relation which holds data related to the tasks (iii) “state” is the relation which holds the robot state (iv) “notif” is the table that holds the configurations and notifications.

We have chosen to store X_cor, Y_cor, teta, as integers (SMALLINT) rather than real numbers in order to save memory and have the same exact data in the tags, as in the programming environment (database and code) which leads to the removal of conversion errors, and the improve code maintainability. Same applies to ID_EPC and data fields.

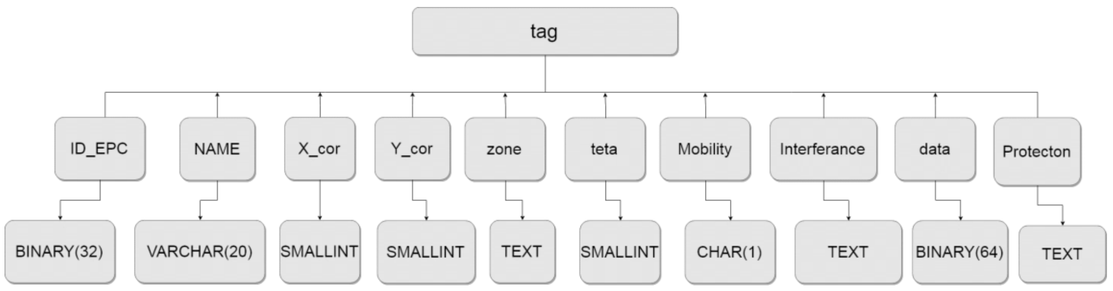


Figure 3.24: Tag relation schema.

3.2.5.1. Tag relation description

Field	Meaning	Comments	Field	Meaning	Comments
ID_EPC	Tag EPC memory content	(our tag memory size is 32 Bytes)	Name	Object name	This is added by user
X_cor	Tag x coordinate in cm related to specific zone plan	None.	Y_cor	Tag y coordinate in cm related to specific zone plan	None.
Zone	Room, sector …	None.	Teta	The angle with plan	32767 is equivalent to 2Pi
Mobility	Tag object ability to move	None.	Interference	Parameters to reduce RSSI to distance error	None.
Data	Tag user memory content.	None.	Protection	If tag is password protected and password	None.

Table 3.1: Tag relation fields description.

3.2.5.2. task relation description

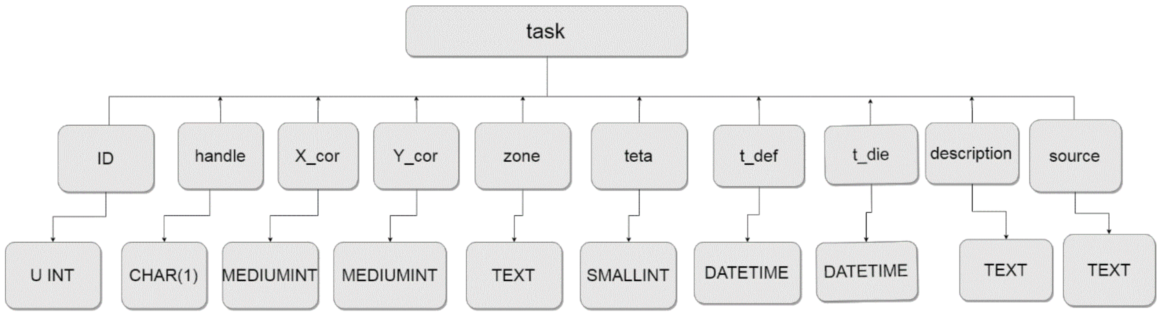


Figure 3.25: Task relation schema.

Field	Meaning	Comments	Field	Meaning	Comments
ID	Database internal ID.	None.	handle	States whether task is in process, processed, no, scheduled, etc.	None.
X_cor	Task x coordinate in cm related to specific zone plan	None.	Y_cor	Task y coordinate in cm related to specific zone plan	None.
Zone	Room, sector …	None.	Teta	The angle with plan the robot needs to be in.	32767 is equivalent to 2Pi
T_def	Time task has been defined.	None.	T_die	Time when task will no longer require execution.	None.
Description	User defined description for the task.	None.	Source	Refers to the task owner(generator).	None.

Table 3.2: Task relation fields description.

3.2.5.3. state relation description

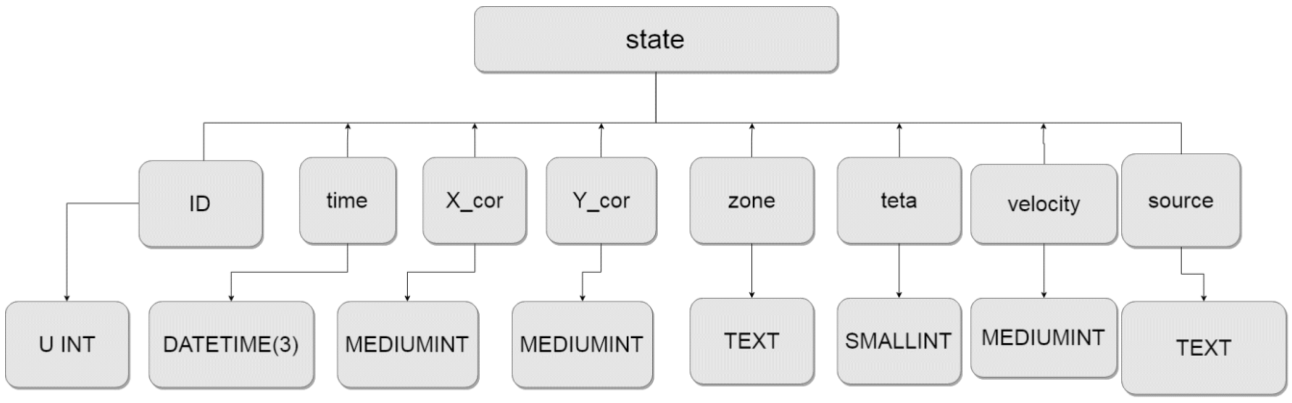


Figure 3.26: Robot state relation schema.

Field	Meaning	Comments	Field	Meaning	Comments
ID	Database internal ID.	None.	Time	Time when data was valid.	None.
X_cor	Robot x coordinate in cm related to specific zone plan	None.	Y_cor	Robot y coordinate in cm related to specific zone plan.	None.
Zone	Room, sector …	None.	Teta	The robot orientation.	32767 is equivalent to 2Pi.
Velocity	The signed amplitude of the velocity.	1 represents 10^{-6} m.s^{-1}.	source	Refers to algorithm and source (robot , tag …) used to fill generate state.	None.

Table 3.3:Robot state relation fields description.

3.2.5.4. Notification relation description

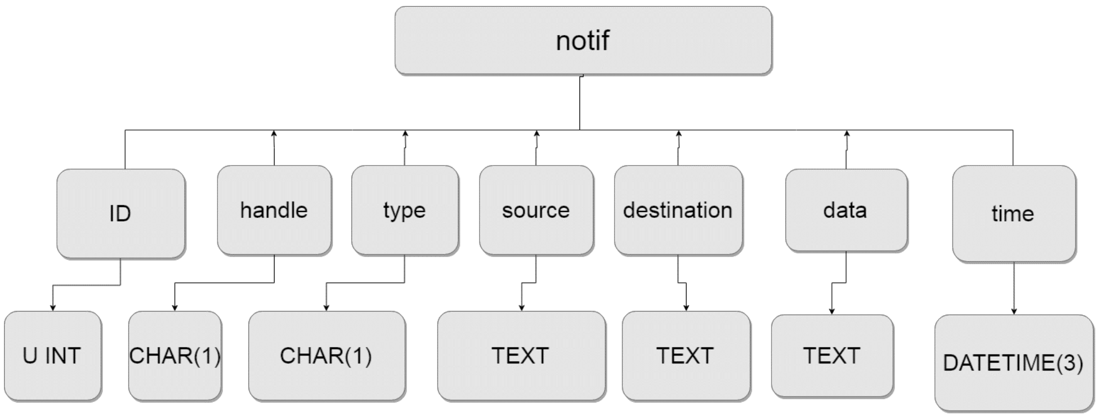


Figure 3.27: Notification relation schema.

Field	Meaning	Comments	Field	Meaning	Comments
ID	Database internal ID.	None.	handle	Notification processing status.	None.
Type	Configuration, error , warning , info …etc.	None.	source	The entity which has generated the notification.	None.
destination	The entity which has needs the notification.	None.	data	The notification code, text, xml, …etc. depending on the source.	None.
Time	Time when notification has been generated	None.			

Table 3.4: Notification relation fields description.

3.2.6. TAG naming specification

Because RFID tags have a limited amount of memory and since robot needs a real-time application, then the information about the tags (environment and objects) must be coded in a more compressed way in order to be all stored in the tags , so the robot will not need to access the main database in order to get information related to its navigation.

To further reduce latency, information and navigation data need to be stored in EPC (memory block used to store tag ID), because it requires less time for be read compared to other memory fields.

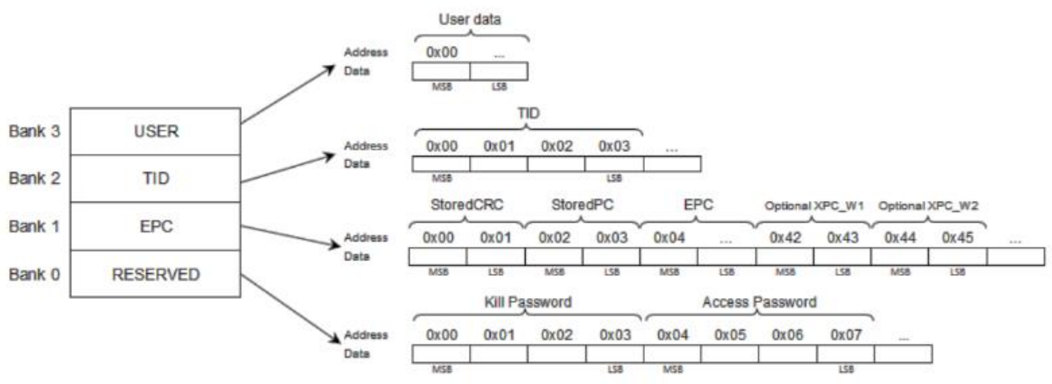


Figure 3.28: Tag memory banks(blocks).

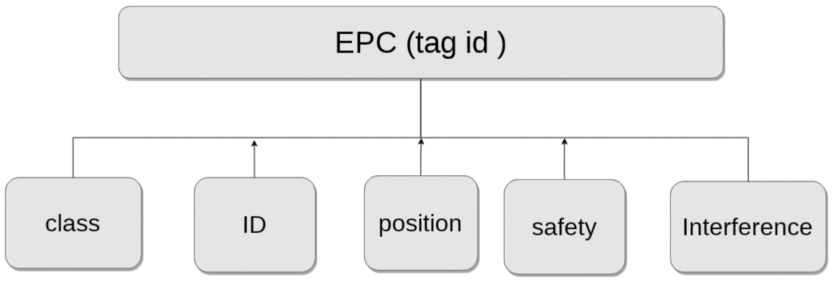


Figure 3.29: Binary representation of the tag naming specification.

Class defines what type of object does this tag refer according to its mobility.
ID define an identifier to the object
Position define the x coordinate, y coordinate and angle related the plan of the tag
Safety define the minimum distance to be kept if any from the tag in order to avoid collision or damage.
Interference define some parameter when included in distance resolution process (From RSSI to distance) will reduce the error caused by electromagnetic interference (EMI), which leads to a more reliable and precise tag localization.

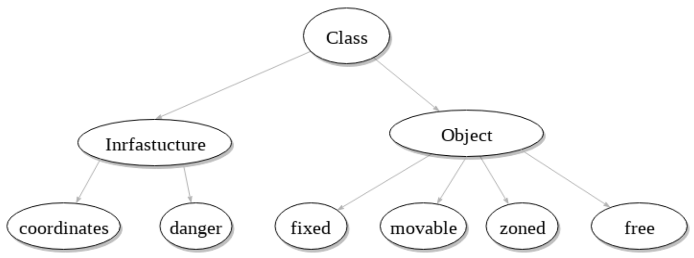


Figure 3.30: Class hierarchy.

a. Infrastructure are the tags used to help the robot identify its location and the characteristics of its environment

b. Objects are the tags used by the robot to identify the objects and the tasks related to them.

c. Credentials these are the tags that stores the necessary information (user, pass ... etc.) to help the robot connecting to the available networks and systems.

3.2.7. RSSI to distance resolution

Since we do not have a predefined command in the reader which gives the distance between the Tag and antennas, so in order to define a function that relies RSSI and distance we have conducted a series of measurements of RSSI values in some predefined position.

3.2.7.1. Experiment procedure

A number of points has been defined as: A0, B4…. with known coordinates on a paper plan which was attached vertically away from any object that may cause interference (free space) then we measure the equivalent RSSI of a tag at the predefined

points , the paper plan is shifted a little more, then we repeat the process until we cannot get any RSSI readings.

After this experimentation we measured data from more than 300 measuring point ,using these data we have defined the shape of the antennas field and the parameters for the RSSI to distance function.

The actual RSSI value depends on numerous parameters[20]:

- Distance between antenna and transponder.
- Transponder type used (tag).
- Chip used in the transponder.
- Connected antenna.
- Transmit power.

- Reflections.
- Noise level in the channel used and in neighbouring channels.
- Tag orientation compared to the antennas field.

3.2.7.2. RSSI to distance formula

The simplified formula for finding the Distance from the RSSI is [17],[18],[19].

$$D = 10^{\left(\frac{P0-RSSI}{10^{N}}\right)} \qquad \textit{(3-1)}$$

D= distance between antennas and tag

P0= -RSSI at the highest absolute value of RSSI (D=0).

RSSI = Received Signal Strength Indicator.

N = Path-Loss Exponent.

Because of the non-homogeneous antenna field, which has a narrow range in the horizontals, and a wide range in the verticals of the antennas, an RSSI value of the tag may refer to many distances depending on the zone where it is i.e. a tag having an RSSI of 60 may be at 2.30 m if it is in the good zone , and it could be 0.84 m if it is in the bad zone.

we have defined a function to compensate the shape of the antennas which is defined as follow

$$\alpha(cos(\theta))^2 + \beta(sin(\theta))^2 \qquad \textit{(3-2)}$$

Where

θ=The angle of the tag on the plan(at horizon θ = 0).
α , β= experimental values defined usinf curve fitting.

The our function becomes

$$D = \frac{c * e^{\left(\frac{P0-RSSI}{10}\right)}}{[\alpha(cos(\theta))^2 + \beta(sin(\theta))^2]} \qquad \textit{(3-3)}$$

Where c is a constant
Using Matlab curve fitting tool we have defined the coefitions and the shape of the function that relates the distance with the RSSI and θ

α	β	C
0.034	0.313	1.055 10^{-5}

Table 3.5: Curve fitting parameters

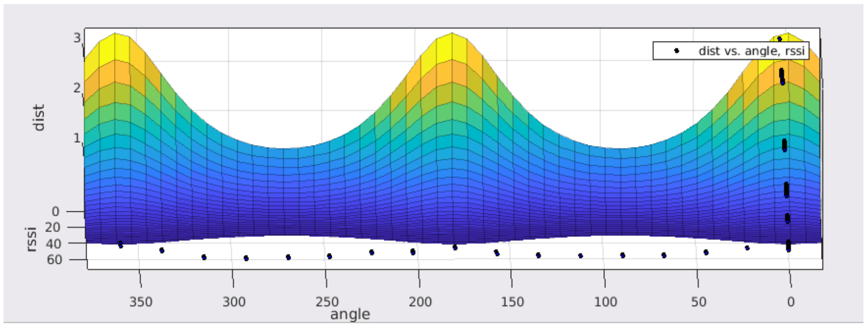


Figure 3.31: Top view of the constructed model.

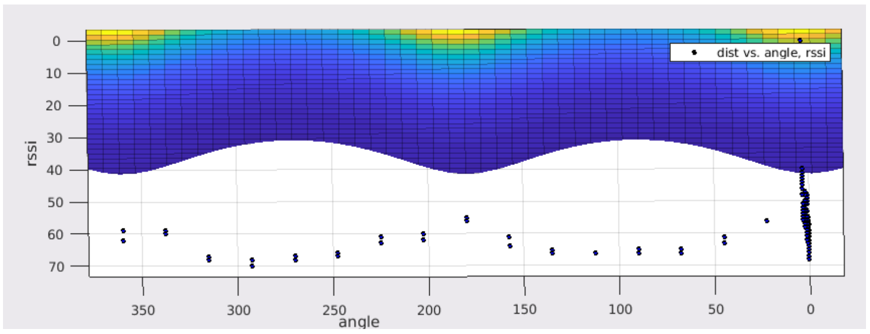


Figure 3.32: ide view of the constructed model.

<u>Note:-</u>

The graphs values in are not accurate because they are a 2D view of the 3D model.

In this graph the blue zones represents a short distance where red zones represents long distances from reader and so on.

We notice that the distance estimated for a specific RSSI increases when the angle is near 0^0 or 180^0 which is as far compliant with the antennas characteristics.

3.3.RFID Based Navigation approach.

In this part , we will see the final step in our project, where the RFID subsystem (client) will provide necessary information and give simple tasks to be accomplished by the robot (server).This can be achieved using **Actionlib package**. Basically , this package has a simple action interface that can be used to implement an action client/server application as the following :

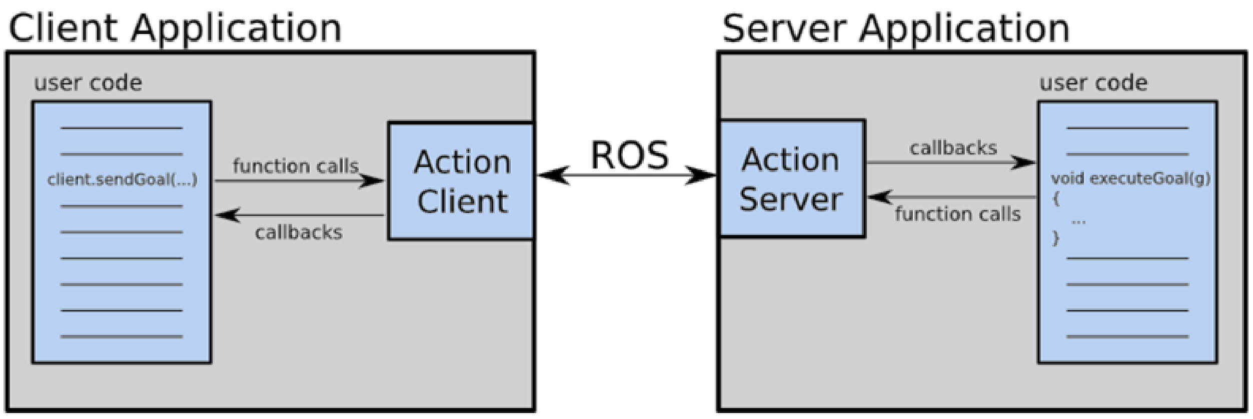


Figure 3.33: ROS action client/server interface[4].

The *Action-Client* and *Action-Server* communicate via a *"ROS Action Protocol"*, which is built on top of ROS messages. The client and server then provide a simple API for users to request goals (on the client side) or to execute goals (on the server side) via function calls and callbacks.

In order for the client and server to communicate, we need to define a few messages on which they communicate. This is with an *action specification*. This defines the Goal, Feedback, and Result messages with which clients and servers communicate.

3.3.1. Goal

To accomplish tasks using actions, we introduce the notion of a goal that can be sent to an Action-Server by an Action-Client. In the case of moving the base, the goal would be a Pose Stamped message that contains information about where the robot should move to in the world. For controlling the tilting laser scanner, the goal would contain the scan parameters (min angle, max angle, speed, …etc.).

3.3.2. Feedback

Feedback provides server implementers a way to tell an Action-Client about the incremental progress of a goal. For moving the base, this might be the robot's current pose along the path. For controlling the tilting laser scanner, this might be the time left until the scan completes.

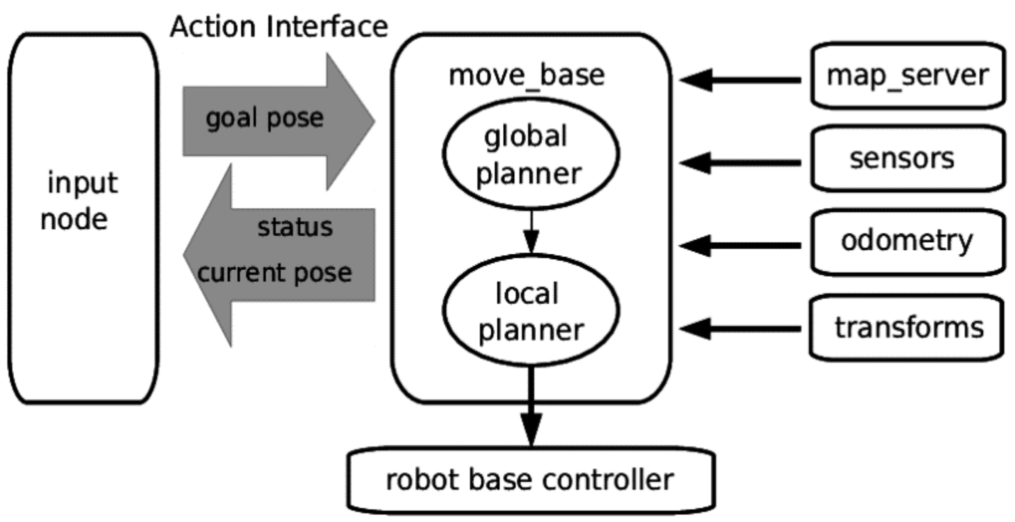


Figure 3.34: ROS action based application[4].

Now, the above application can be implemented using the following diagram:
The input node works as an action client , which is based on the RFID information gathered from tags placed at the environment and it is a part of the RFID subsystem.Accordingly, certain tasks will be ordered to the robot.Since the move_base node is an implementation of a simple action server , it will execute the goal positions given by the client node.

Since the move base is configured previously (chapter 2) we will mainly test directly our application. A simple task is ordered to the move base which is navigating a series of locations according to the RFID tags information.

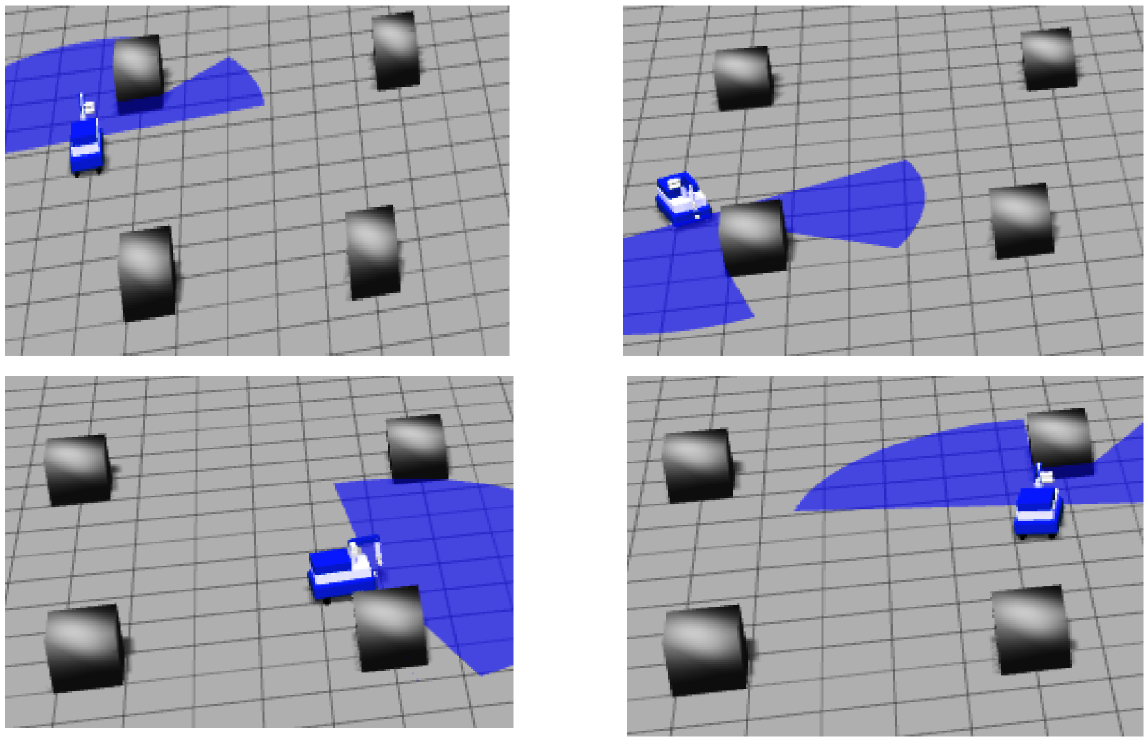


Figure 3.35:Task execution.

Summary :

In this chapter , we have seen all the implementation steps of the RFID application and robot navigation simulation.

All the necessary configurations and design of the system has been presented and explained including RFID reader, ROS, simulation configurations, tag naming conventions, database schema, and RSSI to distance calculation.

Finally, an RFID based navigation approach has been tested using ROS Action interface.

CHAPTER 04: RESULTS AND DISCUSSION

In this chapter we will discuss the problems, results, and suggestion that we have found and made to the RFID-based Robot navigation system.

4.1.RFID platform analysis

we have found that the delay between two scans is approximately 0.3 seconds which is good only for soft robotic applications.

Then we tested only with scan tag ids using the “readTagIDS” command , then the result was dramatically different, the delay decreased to around 0.050 S and sometimes less. Which is a reasonable tome response for most wheeled robotic application.

Unfortunately we could not test the final work on the robot in order to define and analysis the whole system time response and latency, the maneuverability, and safety regarding our RFID based navigation.

For RSSI to distance resolution curve fitting was successful, but requires multiple antennas and testing in order to localizeprecisely tags , which we could not realize due to time constraint.

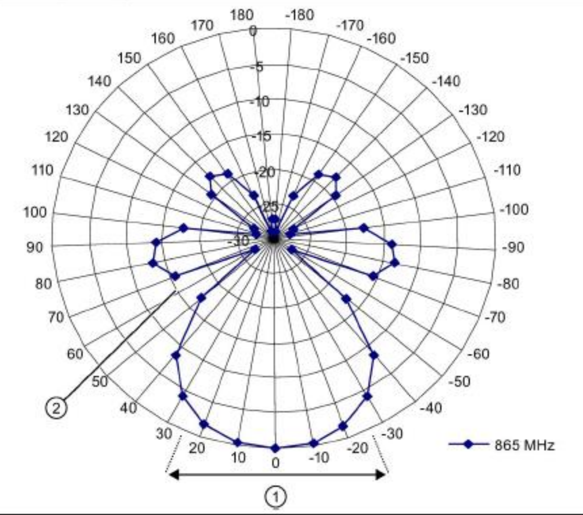
4.1.1. Advantages

- RFID reader has the advantage of the lower infrastructure cost.

1. Passive tags does not require any maintenance during their operation life.
2. Low price of RFID tag (around 20 cents each) leads to cheaper installation and extension costs.

4.1.2. Limitations

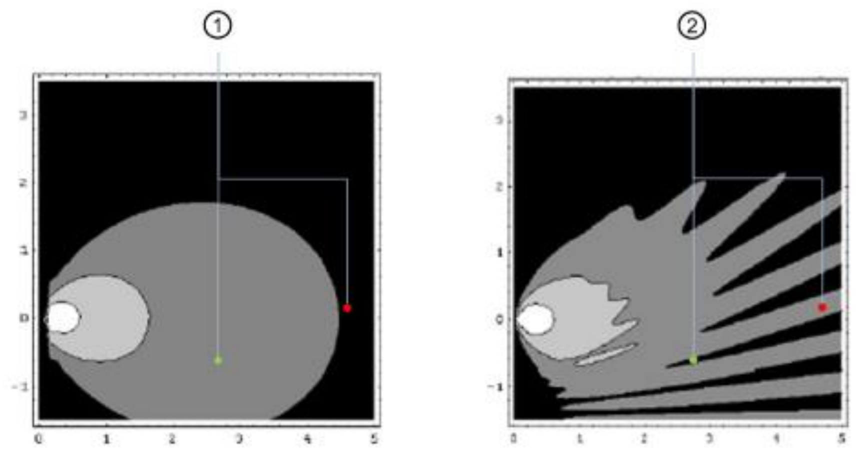
- Non-uniform distribution of the antennas fields leads to increasing complexity in distance determination.



①	Beam width
②	Spurious lobe

Figure 4.1: Directional radiation pattern in polar representation[20].

- Metallic object can cause overshoots and sometimes gaps in the antennas field.
- Areas with dense RFID tags are suseptible to higher positive and negative false detections.



① Identification situation with two transponders in an ideal radio/antenna field

② Identification situation with two transponders in a real radio/antenna field with reflections that can lead to obliteration and overshoots

Figure 4.2: Propagation of UHF RFID antenna fields[20].

4.1.3. Solutions

- make multiple scans and using the common and repeated values of measurement for calculations.
- For tags that are used to localise the robot (infrastructure), an approche to solve the problem of reliability is to store some data about the tag and how it is deployed (orientation , nearest objects) in order to compensate error in measurements.
- To reduce the problems caused by high desity tag areas is by using the black-list command which will ignore the tags that has been processed which keads to simpler inventories and less interference.

4.2. Simulation issues and observation

During the simulation part, some aspects of the environment has been detected that affected indoor mapping and autonomous navigation. These aspects are related to both the environment and the robot. In order to have precise mapping and navigation, these factors should be given proper attention. These issues and observations include :

4.2.1. Height and Orientation of the laser sensor

The position of the laser scanner on the robot is very critical to mapping and navigation. Of course sensors are always mounted on the robot at some height above the ground. One should carefully select the height and orientation of the laser sensor as these are very critical to mapping and navigation. In such situations all the objects below the laser scan become undetectable during mapping and navigation. The map obtained can be incomplete and hence the robot can collide with objects not recognized by the laser.

4.2.2. Environment Selection

Environment is very important specifically in the mapping building process.When the robot is performing navigation in a closed environment with many landmarks , the map generated is better and hence navigation will be also better.

4.2.3. Navigation stack tuning

One of the major features that has to be tuned is the navigation stack.Specifically the move base node which is the core node of the navigation stack responsible for path planning and obstacle avoidance.This node has at least four configuration files that have to be set as good as possible. Those parameters are either related to cost-maps or the base local planner.

4.2.3.1. Simulation time

Basically, the local planner takes velocity samples in robot’s control space, and examine the circular trajectories represented by those velocity samples, and finally eliminate bad velocities (ones whose trajectory intersects with an obstacle). Each velocity sample is simulated as if it is applied to the robot for a set time interval, controlled by sim time parameter. sim time is the time allowed for the robot to move with the sampled velocities. Through our work, we observed that the longer the value of sim time, the heavier the computation load becomes. Also, when sim time gets longer, the path produced by the local planner is longer as well, which is logical.

4.2.3.2. Simulation granularity

sim granularity is the step size to take between points on a trajectory. It basically means how frequent should the points on this trajectory be examined (if they intersect with any obstacle or not). A lower value means higher frequency, which requires more computation power.

4.2.3.3. Robot Footprint Model

The robot footprint model approximates the robots' 2D contour for optimization purposes. The model is crucial for the complexity of distance calculations and hence for the computation time. The footprint is used in the navigation to estimate a collision-free path through the environment. The robot’s footprint is always specified in such a way that it should not intersect with any point representing objects in the map. Circular and complete square footprints are more suitable than the rectangular ones. We have applied both circular and square footprints that gave pretty good results.
The rectangular and polygon footprint of the robot was suitable but it caused many delays and problems in the navigation. Hence the choice of footprint does not only depend upon the robot’s shape but the structure of the environment is also important.

4.3.Summary

In this chapter we have discussed the results of our work, which was good according to our estimation for running the robot platform. Unfortunately due the disk failure that had not been fixed by the maintenance staff which occurred to the robot platform, we could not run the application on the robot to test it in real world.
We have found some limitations of the RFID-platform in regards to distance resolution and localization due to the complex nature of radio signal but we proposed an approach to overcoming the limitation by storing data related to tag distortion the tag themselves.
When it comes to robotic development we have improved the packages and previous work given by the CDTA , and we have given some recommendation and consideration to be considered when developing with the Robuteur/UML platform.
The CDTA internship was good as a whole but we prefer if we had a document presenting all the tools and resources that we can access and at which condition, in order to better tackle problems and issues.

GENERAL CONCLUSION

The goal of this project was to implement an RFID based navigation system for autonomous mobile robot. The mobile robot must be able to achieve tasks by itself, including the perception of its surrounding environment, determination of its position and direction instantaneously, finding and executing its path.

At the beginning, we have stated some generalities about RFID technologies, autonomous navigation and Hardware in the loop simulation. Then, a clear description of the overall system architecture is given where the two subsystems (RFID hardware and simulation) are presented.

We started by a setting up the simulation environment including the robot model such that it can be accessible by ROS, the control package and launch files configurations. Also, the main features of the navigation stack are described and tuned according to our robot specifications.

The RFID platform is implemented using three antennas, the reader and multi tags. An application was developed using python and MySQL on Raspberry Pi 3 model B. Different configurations have been done concerning the RFID reader and tag naming specifications such that the application is working properly. Using this application, the robot can communicate with the RFID tags, read and write their contents. Finally, the received RSSI signal by the robot is transformed into the right distance using curve fitting and hence objects are localized.

To test our work, we started by investigating the simulation. SLAM algorithm is used allowing the mobile robot to model its environment and constructing the occupancy grid map by using laser and encoders sensors. AMCL algorithm is used for localization in the previously build map.Finally, the move base node is used for path planning and obstacle avoidance. Rviz is used for the visualization of topics and data needed for each step.

Finally, a successful RFID based navigation is presented. A ROS action based application is implemented where a simple task is executed by the robot based on the information given by RFID subsystem.

Robot Experimental Problems

Due to a disk failure the Robuteur/UML could not recovered by CDTA maintenance staff, which made us unable to test the work on a real environment.

First, we thank the CDTA staff for giving us the opportunity to work with some of the cutting edge technologies especially our supervisor for all the support and advice they give us during the internship. However, we have some comments and remarks for improvement.

- Equipment testing and maintenance took a long time.
- Lack of a clear view regarding the internship process, the available equipment and resources in order to save time and increase productivity for both staff and students.

Future work

- Enhance the RFID Application connectivity so it could be integrated with other applications.
- Using neural networks in tags localization.
- Adding notification features like SMS, e-mail, and voice.
- Adding a MongoDB to store the serialization of the xml command in order to be queried efficiently.
- Deal with a dynamic environment with many obstacles and moving objects.

REFERENCES

[1] Lentin joseph “ Learn robotics using python” PACKT PUBLISHING 2015.

[2] Roland Siegwart and Illah R. Nourbakhsh *“Introduction to autonomous mobile robots”* MIT press. 2004.

[3] Lentin joseph *“Mastering Ros for Robotics Progrmming” PACKT PUBLISHING 2015.*

[4] Lentin joseph “ Ros Robotics projects” PACKT PUBLISHING 2017.

[5] R.PATRICK GOEBEL “ROS By Example” Volume1 API ROBOT P RODUCTION 2015.

[6] SafdarZaman, Wolfgang Slany, Gerald Steinbauer “ ROS-based Mapping, Localization and Autonomous Navigation using a Pioneer 3-DX Robot and their Relevant Issues“ Graz University of Technology, Austria.

[7] BAKDI Azzeddine “Autonomous mobile robot navigation: Application on RobuTER/ULM“ Master thesis , IGEE , 2015.

[8] Suruz Miah “Design and Implementation of Control Techniques for Differential Drive Mobile Robots: An RFID Approach” Phd Thesis , University of Ottawa 2012.

[9] Xiaolin Hu, Member, IEEE “ Applying Robot-in-the-Loop-Simulation to Mobile Robot Systems” , Georgia State University , 2004.

[10] Rahul Kumar Bhadani, “The CAT Vehicle Testbed: A Simulator with Hardware in the Loop for Autonomous Vehicle Applications”, University of Arizona.

[11] M. Bacic, IEEE member , “On hardware-in-the-loop simulation” , University

of Oxford 2005.

[12] KaiyuZheng , “ROS Navigation Tuning Guide” , September 2, 2016.

[13] SIMATIC Ident RFID systems SIMATIC RF650R/RF680R/RF685R Configuration Manual

[14] https://support.industry.siemens.com/cs/ww/en/pv/6GT2810-2AB04

[15] https://support.industry.siemens.com/cs/ww/en/pv/6GT2812-0GB08

[16] https://support.industry.siemens.com/cs/ww/en/pv/6GT2811-6AA10-0AA0

[17]https://iotandelectronics.wordpress.com/2016/10/07/how-to-calculate-distance-from-the-rssi-value-of-the-ble-beacon/

[18]https://electronics.stackexchange.com/questions/83354/calculate-distance-from-rssi

[19]Indoor Positioning System by “Wade Jarvis”, “Arthur Mason”, “Kevin hornhill” , and”Bobby Zhang”

[20] SIMATIC Ident RFID systems SIMATIC RF600 System Manual.

[21] Pictures from CDTA.

[22] ROBUTER/ULM(Étude de Fonctionnement).

I. APPENDIX A

1.1. An Introduction to Robot Operating System ROS :

Robot Operating System (ROS) is a trending robot application development platform that provides various features such as message passing, distributed computing, code reusing, and so on.

The ROS project was started in 2007 with the name Switchyard by Morgan Quigley as part of the Stanford STAIR robot project. The main development of ROS happened at Willow Garage.

Here are some of the reasons why people choose ROS over other robotic platforms such as Player, YARP, Orocos, MRPT, and so on :

• **High-end capabilities** : ROS comes with ready to use capabilities, for example, **SLAM (**Simultaneous Localization and Mapping) and **AMCL** (Adaptive Monte Carlo Localization) packages in ROS which can be used for performing autonomous navigation in mobile robots.

• Tons of tools : ROS is packed with tons of tools for debugging, visualizing, and performing simulation. The tools such as **rqt_gui** , **RViz** and **Gazebo** are some of the strong open source tools for debugging, visualization, and simulation.

• Inter-platform operability : The ROS message-passing middleware allows communicating between different nodes. These nodes can be programmed in any language that has ROS client libraries. We can write high performance nodes in C++ or C and other nodes in Python or Java. This kind of flexibility is not available in other frameworks.

• Modularity: One of the issues that can occur in most of the standalone robotic applications are, if any of the threads of main code crash, the entire robot application can stop. In ROS, the situation is different, we are writing different nodes for each process and if one node crashes, the system can still work. Also, ROS provides robust methods to resume operation even if any sensors or motors are dead.

• Concurrent resource handling: In ROS, we can access the devices using ROS topics from the ROS drivers. Any number of ROS nodes can subscribe to the same message from the ROS driver and each node can perform different functionalities. It can reduce the complexity in computation and also increase the debug-ability of the entire system.

• Active community: When we choose a library or software framework, especially from an open source community, one of the main factors that needs to be checked before using it is its software support and developer community. There is no guarantee of support from an open source tool. Some tools provide good support and some tools don't. In ROS, the support community is active. The ROS community has a steady growth in developers worldwide.

1.2.Understanding the ROS file system level :

Similar to an operating system, ROS files are also organized on the hard disk in a particular fashion. In this level, we can see how these files are organized on the disk. The following graph shows how ROS files and folder are organized on the disk:

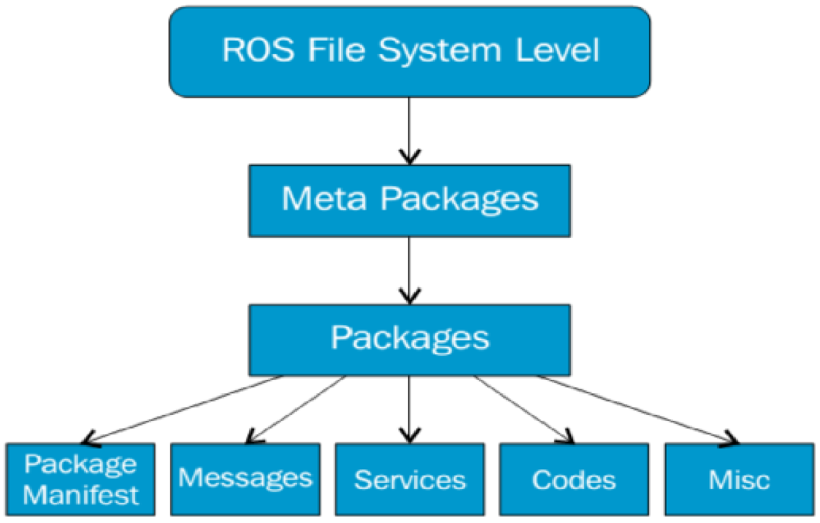


Figure I.1:ROS file system level.

A typical structure of a ROS package is shown here:

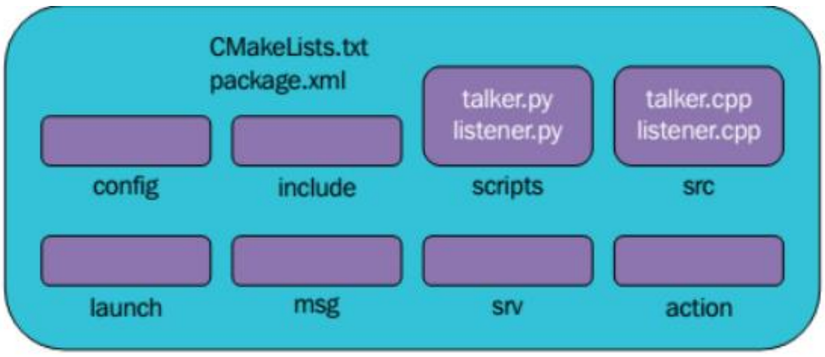


Figure I.2: Structure of a typical ROS package.

Let us now introduce some of ROS’s architecture keywords. ROS uses the concept of nodes, messages, topics, stacks, and packages, below a quick described of this concepts **Node** : A process that performs computation; nodes communicate with each other through messages.

Message: A strictly type of data structure; a node sends a message by publishing it to a topic.

Topic: Channel between two or more nodes; nodes communicate by publishing and/or subscribing to the appropriate topics.

Package: Compilation of nodes that can easily be compiled and ported to other computers, necessary to build a complete ROS-based controller system.

Stack: Groups of ROS packages making easier the process of sharing code with the community.

1.3.Installing ROS Indigo

The following steps are needed for a successful ROS indigo installation. We assume that Ubuntu repository was successfully installed.

C.1. Configure your Ubuntu repositories :

First, you must check that your Ubuntu accepts restricted, universal, and multiversal repositories.

C.2. Setup your sources.list : Setup your computer to accept software from

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

package.ros.org**.**

C.3. Set up your keys : It is important to add the key because with it we can be sure that we are downloading the code from the right place and no body modified it.

```
$sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9F
F1F717815A3895523BAEEB01FA116
```

C.4. Installation:

Before doing something, it is necessary to update all the programs used by ROS. We do it to

avoid incompatibility problems. Type the following command in a shell and wait:

```
$sudo apt-get update
```

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

Then :

C.5.Initialize rosdep : rosdep enables you to easily install system dependencies for source you want to compile and is required to run some core components in ROS.

```
$sudo rosdep init
$rosdep update
```

```
$echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
$source ~/.bashrc
```

C.6. Environment setup :

C.7. Getting rosinstall :

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

D – Installing gazebo : Gazebo can be installed as a standalone application or an integrated application along with ROS. To work with Gazebo and ROS, we don't need to install it separately.

The ROS package that integrates Gazebo with ROS is named gazebo_ros_pkgs , which has created wrappers around a standalone Gazebo. This package provides the necessary interface to simulate a robot in Gazebo using ROS message services.

For The complete Gazebo_ros_pkgs can be installed in ROS Indigo using the following command:

$ sudo apt-get install ros-indigo-gazebo-ros-pkgsros-indigo-gazebo-ros-control

Assuming that the ROS environment is properly set up, we can start roscore before starting Gazebo using the following command:

$ roscore

The following command will run Gazebo using ROS:

$ rosrungazebo_ros gazebo

E- launching Rviz :

First Check for any system dependencies.

$ Rosdep install rviz

Now , you can start Rviz using the following command line :

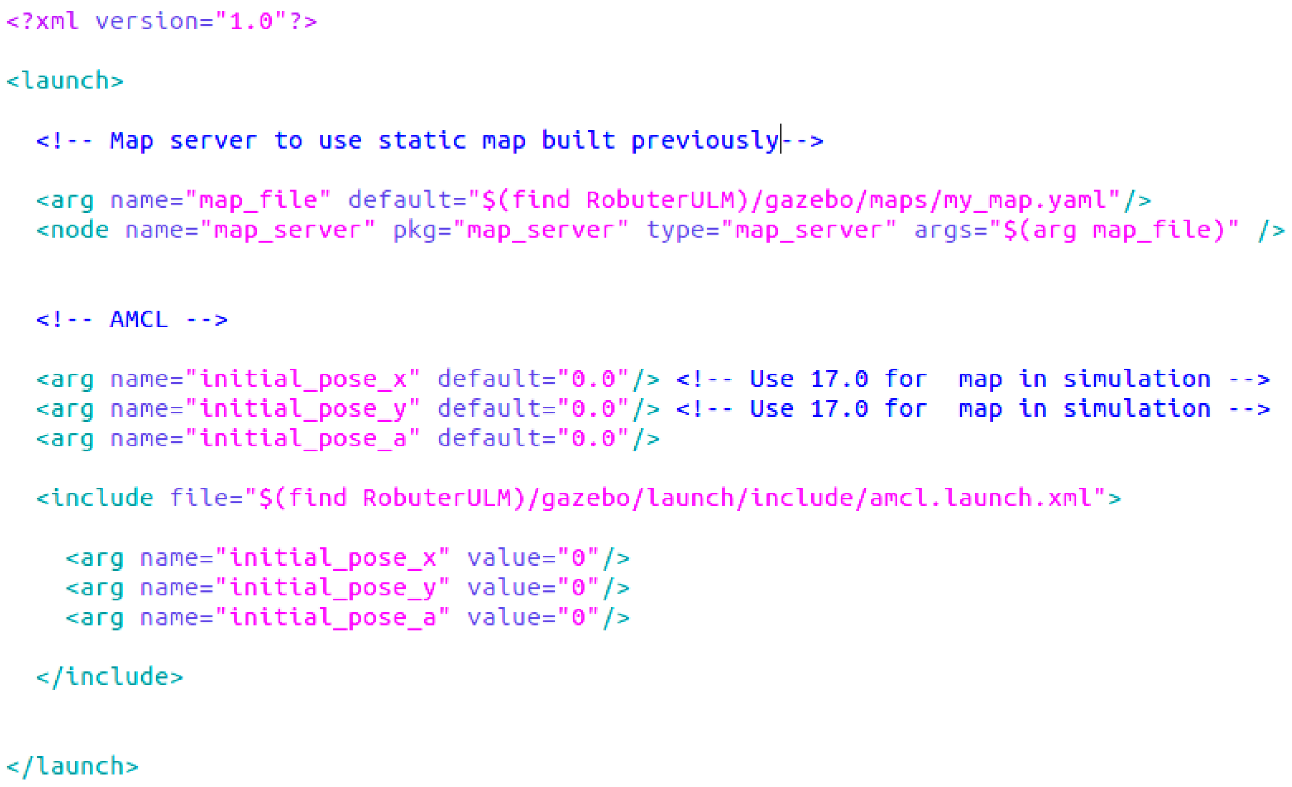
$ rosrunrvizrviz

1.4.Simulation launch and Configuration files

1-Gazebo launch file

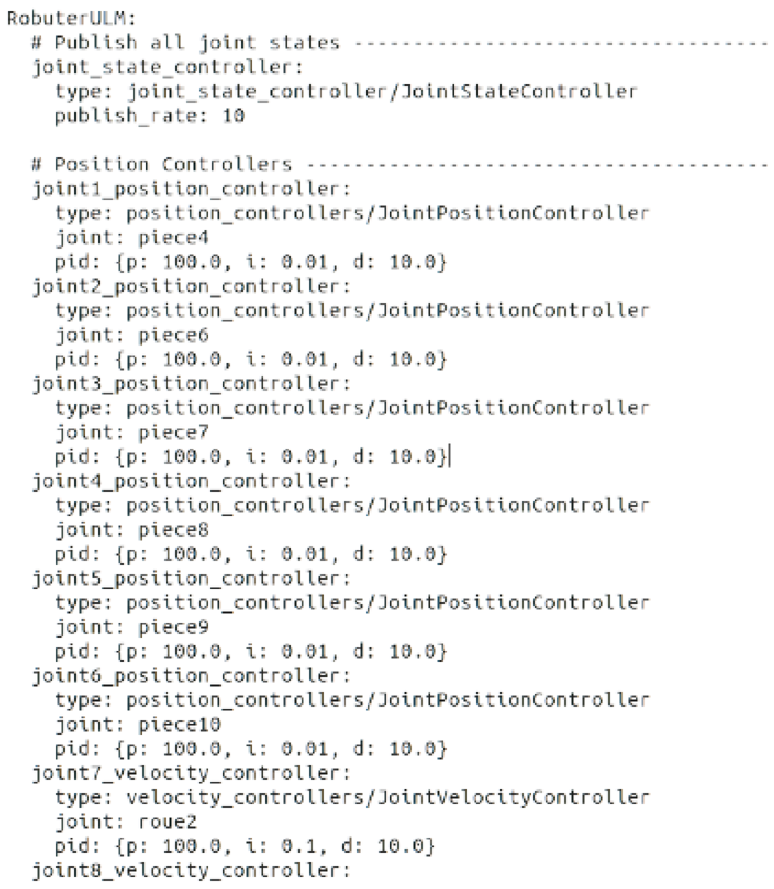
```
<?xml version="1.0"?>

<launch>

        <arg name="robot_name" value="RobuterULM"/>

        <!-- Load the RobuterULM URDF model into the parameter server -->
        <param name="robot_description" textfile="$(find RobuterULM)/description/urdf/RobuterULM.urdf" />


         <!-- Start Gazebo with mybot.world -->

        <include file="$(find RobuterULM)/gazebo/launch/lab.launch"/>


        <node pkg="gazebo_ros" type="spawn_model" name="spawn_$(arg robot_name)"
        args="-x -0.195 -y 1.09 -z 0.215 -unpause -urdf -param robot_description -model RobuterULM" respawn="false" output="screen" >
        </node>

        <!-- run robot_control.launch created previuosly -->

        <include file="$(find RobuterULM)/gazebo/launch/robot_control.launch" />



</launch>
```

2- Amcl launch file



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

<launch>

  <!-- Map server to use static map built previously-->

  <arg name="map_file" default="$(find RobuterULM)/gazebo/maps/my_map.yaml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />


  <!-- AMCL -->

  <arg name="initial_pose_x" default="0.0"/> <!-- Use 17.0 for  map in simulation -->
  <arg name="initial_pose_y" default="0.0"/> <!-- Use 17.0 for  map in simulation -->
  <arg name="initial_pose_a" default="0.0"/>

  <include file="$(find RobuterULM)/gazebo/launch/include/amcl.launch.xml">

    <arg name="initial_pose_x" value="0"/>
    <arg name="initial_pose_y" value="0"/>
    <arg name="initial_pose_a" value="0"/>

  </include>


</launch>
```

3- Robot_control Configuration file :



```
RobuterULM:
  # Publish all joint states -----------------------------------
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 10

  # Position Controllers ---------------------------------------
  joint1_position_controller:
    type: position_controllers/JointPositionController
    joint: piece4
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint2_position_controller:
    type: position_controllers/JointPositionController
    joint: piece6
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint3_position_controller:
    type: position_controllers/JointPositionController
    joint: piece7
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint4_position_controller:
    type: position_controllers/JointPositionController
    joint: piece8
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint5_position_controller:
    type: position_controllers/JointPositionController
    joint: piece9
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint6_position_controller:
    type: position_controllers/JointPositionController
    joint: piece10
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint7_velocity_controller:
    type: velocity_controllers/JointVelocityController
    joint: roue2
    pid: {p: 100.0, i: 0.1, d: 10.0}
  joint8_velocity_controller:
```

4- Robot control launch file :

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

<launch>

  <!-- Load joint controller configurations from YAML file to parameter server -->
  <rosparam file="/home/cherif/catkin_ws/src/RobuterULM/control/config/robot_control.yaml" command="load"/>

  <!-- load the controllers -->
  <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
        output="screen" ns="/RobuterULM" args="joint_state_controller
                                          joint1_position_controller
                                          joint2_position_controller
                                          joint3_position_controller
                                          joint4_position_controller
                                          joint5_position_controller
                                          joint6_position_controller
                                          joint7_velocity_controller
                                          joint8_velocity_controller"/>

</launch>
```

II. APPENDIX B

In this we will talk about the RFID system by describing its components and their characteristics, the configuration that we have used in our work, and the commands that has been implemented in the application.

1.1.Components

The RFID system consists of a reader, antennas, tags, and embedded platform where the application will be executed.

1.1.1. Reader

The reader is the device that provides an interface for using the tags. The reader used in our work in the SIMATICS RF680R which is designed to be used in industrial environment. The reader has two Profinet ports, capable of controlling UFH 4 antennas. it has also 4digital input and 4 digital output

For more information refer to datasheet https://support.industry.siemens.com/cs/ww/en/pv/6GT2811-6AA10-0AA0

1.1.2. Antennas

The antennas are the devices that are responsible of delivering power and communicating with tags. The ones used in this work are RF650A, it is an UFH antenna which can reach 1000mW of radiation power.

For more information refer to datasheet https://support.industry.siemens.com/cs/ww/en/pv/6GT2812-0GB08

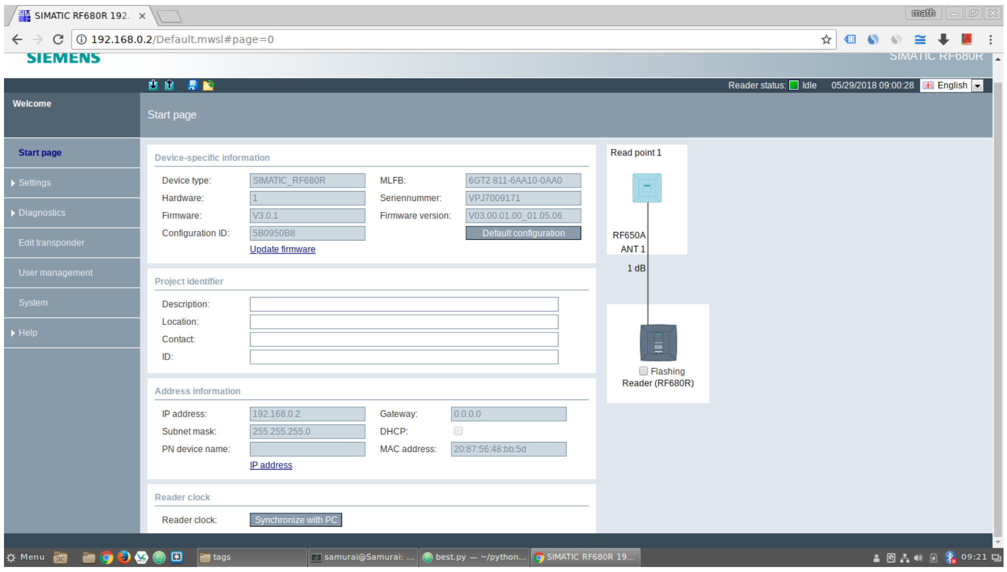
1.1.3. Tags

The tags are the storage elements, the ones used in this work are 6GT2810-2AB04, these are passive tags that has a range of 4m, the memory configuration is 64B user, 32B EPC, and 64B TID
These tags are classified as Class 1 Gen 2 / ISO 18000-6C
For more information refer to datasheet https://support.industry.siemens.com/cs/ww/en/pv/6GT2810-2AB04

Figure II.1RFID tag.

1.2.Configuration

The reader was configured using the WBM interface which is an interface accessible by the browser by typing the IP address of the reader, (192.168.0.2/24).



Reader status: Idle 05/29/2018 09:00:28 English

Device-specific information

Device type:	SIMATIC_RF680R	MLFB:	6GT2 811-6AA10-0AA0
Hardware:	1	Seriennummer:	VPJ7009171
Firmware:	V3.0.1	Firmware version:	V03.00.01.00_01.05.06
Configuration ID:	5B0950B8		

Project identifier

Description:
Location:
Contact:
ID:

Address information

IP address:	192.168.0.2	Gateway:	0.0.0.0
Subnet mask:	255.255.255.0	DHCP:	☐
PN device name:		MAC address:	20:87:56:48:bb:5d

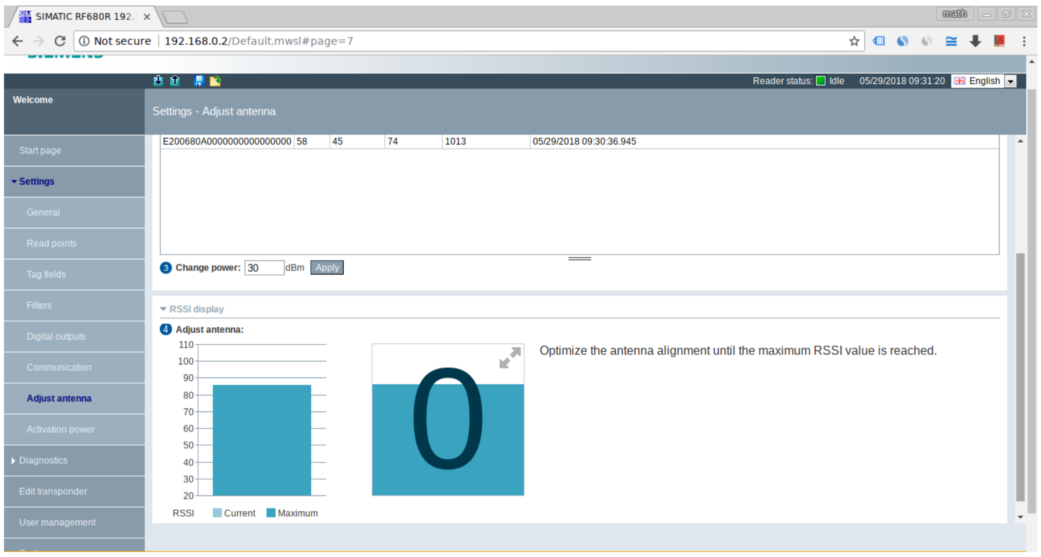
Reader clock

Reader clock:

Figure II.2: WBM (reader web interface).

1.2.1. Hardware configuration

The objective is to identify the hardware to be used with the reader (Antennas, cables, Tags), then configuring the country profile which stands for the operation characteristics approved by the country (frequency , max power …) since Algeria was not in the country approval list we have chosen the most used one “ETSI”.



Reader status: Idle 05/29/2018 09:31:20 English

E200680A0000000000000000	58	45	74	1013	05/29/2018 09:30:36.945

3 Change power: 30 dBm

4 Adjust antenna:

Optimize the antenna alignment until the maximum RSSI value is reached.

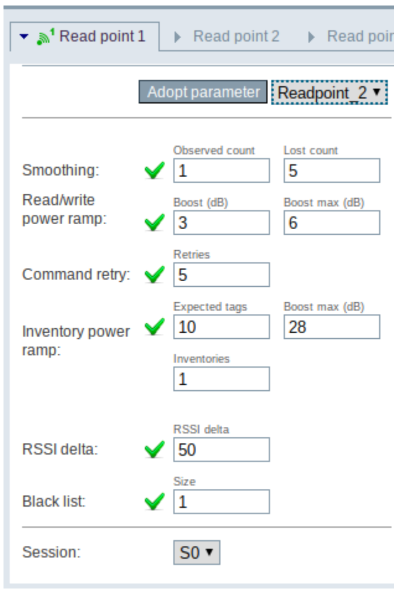
Figure II.3: Antennas adjustment interface.

1.2.2. Software configuration

The objective is to adjust and configure antennas for the highest capture range … then we configure tag one by one by giving them new identifiers because when they come from the factory they all come with the same ID which make it impossible to distinguish and modify then when they all are in the range on the antennas.

1.2.3. Algorithms configuration

After configuring the antennas and tag we proceed to configuring the way the reader scan and modify tags, in our configuration we aimed to get the best performance possible from the reader by increasing the power ramp (value of power to increase when action fail), reducing the blacklist size, increasing the RSSI delta… for more information on tag algorithms please refer to the configuration manual(C79000-G8976-C386-06)



Smoothing:	✓	Observed count: 1	Lost count: 5
Read/write power ramp:	✓	Boost (dB): 3	Boost max (dB): 6
Command retry:	✓	Retries: 5	
Inventory power ramp:	✓	Expected tags: 10	Boost max (dB): 28
		Inventories: 1	
RSSI delta:	✓	RSSI delta: 50	
Black list:	✓	Size: 1	
Session:		S0	

Figure II.4: Operation algorithms configuration page.

1.2.4. Raspberry pi setup

The operating system installed in the raspberry pi is the latest Raspbian Jessie , we installed this specific version because it is the only version which is compatible with ROS Indigo.

Indigo and Jessie are naming version of ROS and Raspbian.

We also install MySQL and MongoDB , to be used for data storage.

1.3.Implemented commands

This is the list of commands that has been implemented in the application.

CMD	comments	CMD	Comments
hostGreetings	Create link.	lockTagBank	
hostGoodbye	Broke link.	getObservedTagIDs	
heartBeat	Test link.	readTagIDs	Read from EPC memory.
getAllSources	Get configuration read points.	readTagField	Read from a predefine field.
getConfiguration	Get the configuration stored.	readTagMemory	Read from a specific location.
getConfigVersion	Get configuration unique id.	writeTagField	Write to a predefine field.
getActiveConfiguration	Get the actual active configuration.	writeTagID	Write to EPC memory.
getTime	Get reader time.	writeTagmemory	Write to a specific location.
setTime	Set reader time.	getParameter	Get reader specific parameter.
getLogFile	Get log file.		

Table II.1: Implemented commands

For more information on commands meaning and usage option please refer to this manual Configuration Manual, 03/2018, C79000-G8976-C386-06