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 Electronics

Final Year Project Report Presented in Partial Fulfilment of the Requirements for the Degree of

MASTER

In **Control Engineering**

Option: **Control Engineering**

Title:

Implementation of motion generation strategy for mobile robots with RFID sensors

Presented by:

- **SEBAA Abderrahmane**

Supervisor:

- **Mme.AKLI Isma**
- **Mme.DERRAGUI Nabila**

Registration Number...../2020

Abstract

Some modern Robotics platforms use Radio Frequency IDentification (RFID) technology in various fields of application (industrial, service ...). RFID systems are composed of antennas, readers and tags. RFID tag consists of a memory storing information. The reader reads/writes information from/on the tags through the antennas. In robotic context, the tags can contain information about the environment such as obstacles, humans, robots.... This master thesis was carried in the Center of Development of Advanced Technologies (CDTA) and the main goal was to integrate a new software and hardware architecture for the differential drive Robotic platform, with Integrating RFID Sensors in the motion strategy.

The Proposed architecture was based on a Single-board Computer (RaspberryPi) embedded platform, a microcontroller based card and Robot Operating System (ROS) for the software implementation and the approach execution. The embedded platform has to control the movement of the mobile robot, and to read sensorial information (RFID, Laser measurement Sensor,...). The navigation module should be introduced with integrating RFID technology. The robot takes decisions of regulating its speed or setting a new destination based on the information contained in the RFID tags it receives. This can be very beneficial in an industrial environments where a mobile robot can carry products to different lines in the production process based on the RFID tag attached to them.

The value added by the architecture is to provide a fully functional and scalable differential drive robotics platform with integrating RFID technology for further research and development projects.

Dedication

I dedicate my FYP work to my family and many friends. A special feeling of gratitude to my loving parents, whose words of encouragement and push for tenacity ring in my ear. I also dedicate this FYP work to my many friends who have supported me throughout the process. I will always appreciate all they have done. I dedicate this work and give special thanks to my uncle Khirddine, his wife and Mima for hosting and supporting me during the time I was working on the project in CDTA.

-SEBAA Abderrahmane-

Acknowledgments

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 supervisors Dr. DERAGUI Nabila, and Dr. AKLI Isma for her guidance, support and help throughout the different phases of the project and to all and every teacher who taught and guide us from primary school to the last year of study.

Contents

Abstract **1**

Dedication **2**

Acknowledgments **3**

1 Problem Description and Literature Review **10**
1.1 Introduction . . . 10
1.2 Problem description . . . 10
1.3 Robotics Embedded Systems . . . 11
1.3.1 RaspberryPi Single-Board Computer . . . 11
1.4 Robotics Software Development . . . 11
1.5 Literature Review . . . 12
1.5.1 The use of raspberryPi in mobile Robotics . . . 12
1.5.2 Application of RFID Sensors in Mobile Robotics . . . 18
1.6 Conclusion . . . 23

2 Overview On the Proposed Platform Architecture and Control of of Differential Drive Robot **24**
2.1 Introduction . . . 24
2.2 RFID System . . . 25
2.2.1 The RFID Reader . . . 25
2.2.2 RFID tags . . . 26
2.3 Control of Differential Drive Mobile Robots . . . 27
2.4 Differential Drive Robot modelling and kinematics . . . 28
2.5 Robot differential Drive Model and Position Control . . . 28
2.6 Robot Odometry . . . 30
2.6.1 Getting Position from Encoder ticks . . . 30
2.7 Go to goal algorithm . . . 31
2.7.1 Global path planning . . . 31
2.7.2 Local path planning . . . 31
2.8 Path Planning with RFID . . . 31
2.9 Real time reading of RFID information . . . 32
2.10 Conclusion . . . 32

3 Platform implementation **33**
3.1 Introduction . . . 33
3.2 General overview of ROS . . . 33

3.2.1 Computation graph model . . . 33
3.2.2 Tools . . . 34
3.2.3 ROS Distributions: . . . 35
3.3 Hardware In The Loop Simulation . . . 35
3.3.1 Robot model and virtual environment . . . 35
3.3.2 Navigation Stack . . . 37
3.3.3 Localization . . . 38
3.3.4 Path Planning . . . 38
3.3.5 Integration of RFID . . . 38
3.3.6 Reader Node . . . 38
3.3.7 Handler Node . . . 39
3.3.8 Speed_regulator Node . . . 40
3.3.9 GUI Node . . . 40
3.3.10 Integration of RFID . . . 45
3.4 Raspberry pi Solution . . . 46
3.4.1 Low Level Design . . . 46
3.4.2 Linking ROS to the low-level controller . . . 48
3.4.3 Publishing Sensor Data and Speed Commands through ROS topics . . . 49
3.4.4 My_Pos_Calc Node . . . 49
3.4.5 My_uni_to_diff node . . . 49
3.4.6 Combine motors node . . . 50
3.5 Conclusion . . . 50

4 validation and results **51**
4.1 Introduction . . . 51
4.2 System Setup . . . 51
4.3 Communication With The RFID Reader . . . 53
4.4 HITL Validation . . . 53
4.5 Hardware Validation . . . 55
4.6 Suggestions For Improvement . . . 56
4.7 Conclusion . . . 56

Appendix A **58**

Appendix B **60**

Appendix C **62**

List of Figures

1.1 Raspberry pi . . . 13
1.2 Agent model with kinect . . . 13
1.3 Differential Kinematic Model . . . 14
1.4 Arduino Control Loop . . . 14
1.5 OBC Part Implementation . . . 14
1.6 Realization of RT Control Application Integrating ROS and RT Tasks on Xenomai 15
1.7 CAD Model and Real Model of Mobile Robot with Omnidirectional Wheels . . . 16
1.8 Block diagram of The Robot Control System . . . 16
1.9 General Description of The Implementation of The Automatic Ablation Process . . . 17
1.10 Transmitter Side Block Diagram . . . 18
1.11 Receiver Side Block Diagram . . . 18
1.12 RFID System . . . 19
1.13 Block Diagram of Robot Navigation . . . 19
1.14 Top and Front View of The Overall Kit . . . 20
1.15 System Architecture . . . 21

2.1 The Raspberry pi Implementation Architecture . . . 24
2.2 The HITL Implementation Architecture . . . 25
2.3 Integration of RFID reader in an IT environment . . . 26
2.4 Integration of RFID reader in an automation environment . . . 26
2.5 RFID tags . . . 27
2.6 Types of Control . . . 27
2.7 Stable Motion Planning for Dynamic Non-holonomic Mobile Robots . . . 28
2.8 Differential Drive Mobile Robot Model . . . 29
2.9 Optical Encoder signal . . . 30

3.1 ROS Master Operation Diagram . . . 34
3.2 Robot TF Tree . . . 36
3.3 Map Building Visualization in RVIZ . . . 37
3.4 Navigation Stack Setup . . . 37
3.5 Home Page of The Graphical User Interface . . . 39
3.6 Home Page of The Graphical User Interface . . . 40
3.7 Home Page of The Graphical User Interface . . . 40
3.8 Reader Interface of Graphical User Interface . . . 41
3.9 Configuration Menu . . . 42
3.10 Tags Menu . . . 43
3.11 Manual Mode Interface . . . 44

3.12 Autonomous Mode Interface 45
3.13 Tag ID Interpretation By ROS Nodes 46
3.14 Differential Drive Mobile Robot 47
3.15 Motor Driver pinout [Sparkfun datasheet] 47
3.16 Logic diagram of L298N motor driver [Sparkfun datasheet] 48
3.17 Optical Encoder Signal 48
3.18 Linking Arduino and ROS 49
3.19 My_Pos_Calc Diagram 49
3.20 My_Uni_To_Diff Diagram 50
3.21 CombineMotors Diagram 50

4.1 Virtual Environment with Gazebo 52
4.2 RVIZ With Loaded Configuration 52
4.3 Antenna reading RFID tag and its information displayed on the GUI 53
4.4 Visulization in RVIZ after moving the robot around manually 54
4.5 Message on the GUI after reading the RFID tag of an object 54
4.6 Robot Reaching Its Goal in Gazebo 55
4.7 GUI message after detecting a movable obstacle 55
8 Ubuntu Mate 62
9 Gazebo 63
10 Qt Designer 63
11 Start page of WBM 64

Introduction

Autonomous robots have captured the attention and focus and most engineers in last few decades, specially the navigation strategies applied for the robots to know their position and to find the best path to a goal destination. In general, the robotics area of research is involves the design, construction and use of robots with the aim to enhance the intelligence, efficiency and cost (computational or material) in those areas. Advancing in this field yields in advanced machines that help the human to perform tasks or do it completely by itself. This is very beneficial, specially in industrial environments where speed and efficiency are highly needed, and also in environments that can be threatening to human safety. The use of industrial robots goes back to 1937, where a crane-like device was built almost entirely using Meccano parts, and powered by a single electric motor. Five axes of movement were possible, including grab and grab rotation. Automation was achieved using punched paper tape to energise solenoids, which would facilitate the movement of the crane's control levers. The robot could stack wooden blocks in pre-programmed patterns. Now, the industrial robot applications vary tremendously, they include painting, welding, assembly, pick and place for printed circuit boards, palletizing packaging and labeling, product inspection and much more. One of the technologies integrated in autonomous robots is the RFID technology. A mobile robot can use RFID technology for localization, navigation or identification of objects surrounding it. The main perception tools for a robot to gather information about its environment have been vision sensors for a long time which provide acceptable results. However, there are some downsides for vision-based navigation systems, such as the lack of vision in some environments, the high complexity of image processing techniques, and the need of objects to be in line of sight of the robot to be identified and taken into consideration. So, the need to overcome these disadvantages pushes the engineering community to find a nav- igation strategy that relies on other than vision sensors or something that complements them, and one of the best modern technologies that make a great candidate for that in the RFID technology. In this project, we look forward to design and implement an RFID-based navigation technique that makes the robot take its movement decisions based on the data obtained from the the RFID tags deployed in its environment. In an industrial plant, RFID provides higher product visibility, eliminates manufacturing errors, enables asset management, potentially can increase production speeds and has other benefits as well. In contrast to simple barcode tracking, multiple RFID tags can be read at the same time as they pass by the reader. This report is organized as follows. Chapter 01 Problem description and literature review where we describe the problem that we look to find a solution to throughout this project, we present a general overview of the mobile robots, talk about the industrial application which is in context of our work and details about the aim of the project. Next in this chapter we make a literature review on articles that address the use of raspberry pi in mobile robots and the use of RFID sensors in robotics and obstacle avoidance behavior. Chapter 02 The proposed solution where we give a general overview of the global architecture,the control theory behind mobile robots, talk about the goal to goal and obstacles avoidance algorithms and the the RFID

technology contribution with these technologies, and lastly in this chapter we talk about the real time reading of RFID information. Chapter 03 Development where we give a general overview of ROS, the low level design, connecting the low level hardware to ROS and describe the architecture of hardware in the loop simulation. Chapter 04 validation and results of our implementation of the described system and obstacle avoidance and path finding algorithms.

Chapter 1

Problem Description and Literature Review

1.1 Introduction

In this chapter we will describe the problem that we look to find a solution to throughout this project, we present a general overview of the mobile robots, talk about the industrial application which is in context of our work and details about the aim of the project. Next in this chapter we make a literature review on articles that address the use of raspberry pi in mobile robots and the use of RFID sensors in robotics and obstacle avoidance behavior

1.2 Problem description

A mobile robot is a machine controlled by software that uses sensors and other technology to identify its surroundings and move around its environment. Mobile robots function using a combination of artificial intelligence (AI) and physical robotic elements[13]. Mobile robots can be classified based on the mechanical system used to perform movements. The main three types are legged, tracked and wheeled robots[13]. However from a software perspective, we can define 2 types of robots, Autonomous and Non-Autonomous. In the field of autonomous robotics, the robot must be able to perform operations and movements without human interaction. An autonomous robot must be able to build a map of its environment, determine itslocation and navigates from point to point while identifying obstacles and changing its behavior according to the nature of the obstacles. Mobile robots are widely used in the industry. An industrial environment is a term used to describe working conditions that may be outside of optimal. Industrial environments are usually more harsh than normal work environments, such as an office. In an industrial environment, people and equipment are exposed to more extreme conditions. An example of industrial environment are factories, where humans can experience extreme heat or cold or be exposed to dangerous materials. So, the use of robots and autonomous mobile robots for transportation is much more convenient, especially since modern factories tend to have large warehouses with heavy equipment used in production. RFID tags placed around the environment can give a lot of information to the robot. In an industrial environment, there are a variety of locations that require the robot to change its behavior. For example decreasing the speed in production lines that contain human operators and obstacles, or for example this line is empty so the robot increases the speed. Or for example the robots should stop at a specific location to move some objects from one place to

another. These scenarios can be implemented by the help of vision sensors. However using vision techniques has some disadvantages, like the high amount of computational power. That can lead to many time delays that are going to affect the overall real-time system. In addition to that, vision does a poor job in the localization of the robot and the obstacles, and in most cases it must be combined with a distance sensor like a laser scanner. Moreover the cost of the overall robotic platform increases, since these components are expensive and the need for cloud technologies to handle the high processing power. On the other hand, integrating RFID technology becomes very handy. Since we can store information about the environment on small tags then extract it using antennas and RFID readers, which doesn't require a lot of computational power. In order to have a complete robotic architecture, the robotic system should be composed of robotic embedded systems and robotics software development. Sensors are also important in a robotic system as they are the robot's tools to perceive the environment, such as RFID antennas that can read RFID tags spread around the environment that can give the robot plenty of information of its surroundings with very low computational cost.

One of the ways of implementing and testing our work is Hardware In The Loop (HITL) which is a technique where real signals from a controller are connected to a test system that simulates reality, tricking the controller into thinking it is in the assembled product. Test and design iteration take place as though the real-world system is being used. You can easily run through thousands of possible scenarios to properly exercise your controller without the cost and time associated with actual physical tests[1]. A great simulator to use in HITL simulation would be Gazebo, which offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. Another way of implementaion would be the raspberry pi, which will run most of our software processes to control a real differential drive robot.

1.3 Robotics Embedded Systems

As stated before, a mobile robot is a physical machine, hence it consists of some hardware elements that determine the mechanical and electrical behavior of the robot. In addition to the hardware part, a software must be present to drive the robot. In this section, we'll talk about the tools embedded in the robot.

1.3.1 RaspberryPi Single-Board Computer

Since we are going to use Robot Operating System, our embedded system must include a computer running Linux operating system (Ubuntu Distribution) The Raspberry Pi is a small Single-Board Computer developed by The Raspberry Pi Foundation, that runs the Linux Operating System , its main distribution is called Raspbian. But in our case we have installed Ubuntu Mate [Appendix C] since ROS works better with ubuntu distributions.

1.4 Robotics Software Development

Developing optimized software for robotic is crucial, if the application requires many sub-parts, the process requires developing many separate blocks of codes that need to work all together. Hence it is necessary to develop codes under some programming structure, that leads to a reusable and understandable code. In our application we choose to use Robot Operating System (ROS), which is robotics middleware (i.e. collection of software frameworks for robot software development). ROS provides services designed for a heterogeneous computer cluster such as hardware abstraction,

low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor data, control, state, planning, actuator, and other messages.[3]

ROS provides functionality for hardware abstraction, device drivers, communication between processes over multiple machines, tools for testing and visualization, and much more. The key feature of ROS is the way the software is run and the way it communicates, allowing designing complex software without knowing how certain hardware works. ROS provides a way to connect a network of processes (nodes) with a central hub. Nodes can run on multiple devices, connect to the hub in various ways. Another feature of the ROS is the reusability of the codes, so that a developer can focus on a certain task without wasting a lot of time in other areas of the project. All these advantages make ROS widely used in the robotics community.

1.5 Literature Review

In this part we introduce the literature review of the various techniques used in the field of mobile robots. We will focus mainly on the use of the raspberryPi and the Integration of RFID technologies in the field.

1.5.1 The use of raspberryPi in mobile Robotics

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a little device that enables developers to explore computing, and to learn how to program in languages like Scratch and Python. It's capable of doing everything you'd expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games. The Raspberry Pi has the ability to interact with the outside world, and has been used in a wide array of digital maker projects, from music machines and parent detectors to weather stations and tweeting birdhouses with infra-red cameras. It can be used for various tasks especially in the field of Robotics and internet of things. Some advantages of this small computer is the cheap price, the ability to connect to the internet easily and the large community support. Which makes it very suitable in small to medium projects and research purposes, especially in the field of robotics.

In this section we are going to mention some recent scientific articles that have used raspberryPi as part of their implementation, and summarize the most important points in each article and the role raspberryPi has taken in each one. The applications of Raspberry Pi varies a lot, so each article discusses a different application.

1.5.1.1 Real-World Modeling of a Pathfinding Robot Using Robot Operating System (ROS)

This paper[11], discussed a hands-on implementation of A* path finding algorithm on a local hardware model that is entirely handcrafted (figure Fig. 1). The main goal was designing control algorithms to mitigate physical dynamics problems in the implementation section and testing the agent against its simulation results to see how accurately it performs using feedback and control algorithms. The prototyped model Included:
-Asynchronous dual DC geared motors with Hall Effect encoders for monitoring odometry.
-The computational system comprises Linux (Ubuntu) based Raspberry Pi 3b board and Arduino UNO controller as an auxiliary controller between analog sensors and the on-board-computer

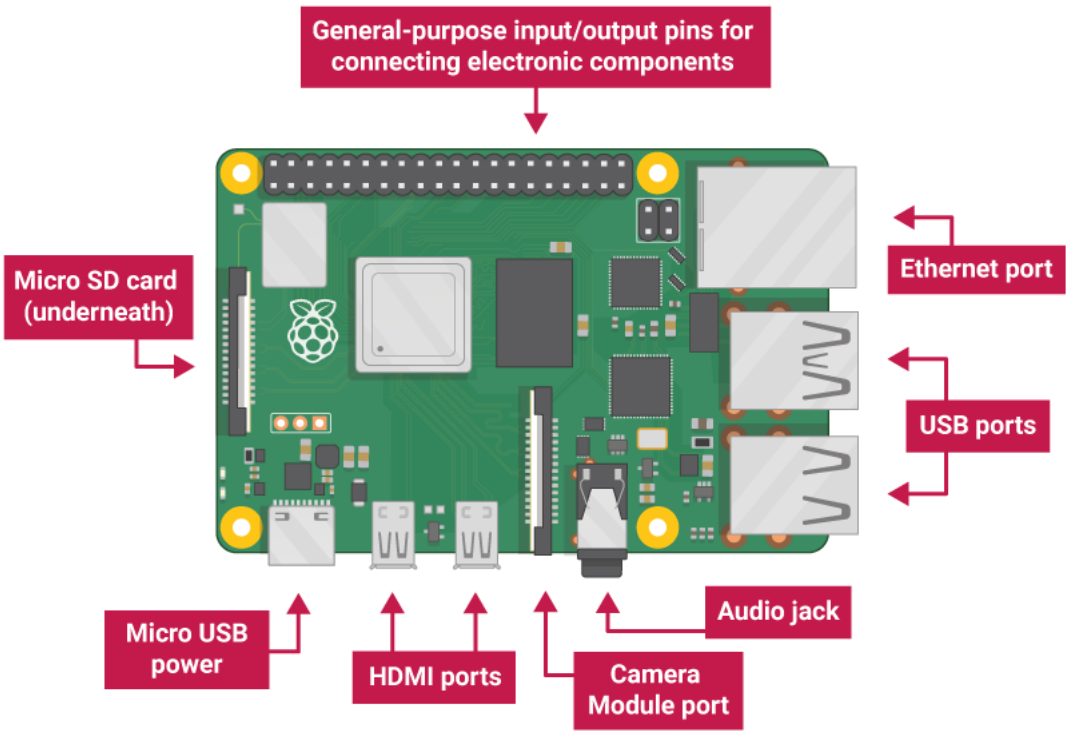

Figure 1.1: Raspberry pi

(OBC).

-Robot Operating System (ROS) environment is used for creating event nodes for controlling and executing offline planning systems.

-Due to time constraints the paper focused on offline planning (A*) only instead of implementing simultaneous localization and mapping (SLAM) and online planning and mapping. Wheeled mo-

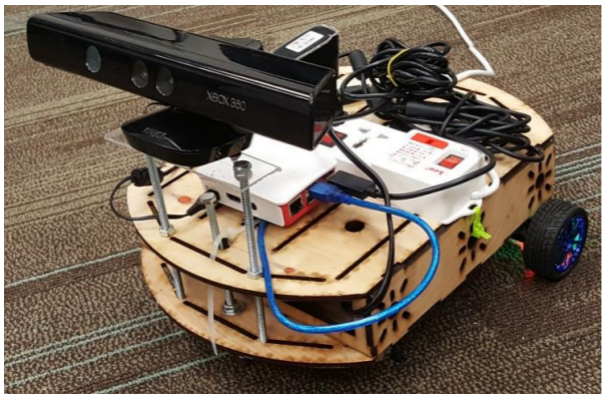

Figure 1.2: Agent model with kinect

bile robots mostly operate over a differential drive mechanism. It consists of two motors attached with wheels on a similar axis. Both motor drives are independent of each other's movement. The common access of both motors is known as their center of curvature. The agent cannot move in the direction along the axis. The motion planning of the robot can be maneuvered using the robot location with angular (ω) as well as linear velocity (V). The velocities of both drives must be synchronized in order to move the agent in the desired path. The DC motors are controlled

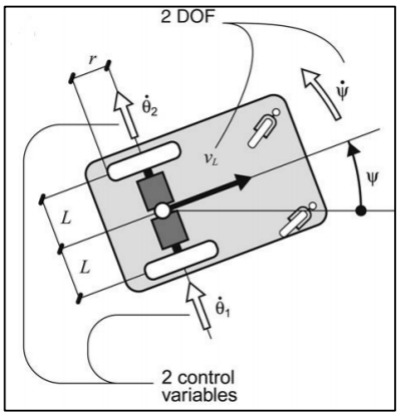

Figure 1.3: Differential Kinematic Model

using H-Bridge that provides switches and a bridge to the external power bank for the motors. The pulse width modulation (PWM) throughputs the velocity/acceleration by an impulse of duty-cycle probed from the controller. Since the control system handles differential drive parameter and H-bridge handlers The OBC (Raspberry Pi) is on the higher level of the system. it runs pathfinding algorithms and returns the required control coefficients to the control unit over USB serial communication node. ROS is used to handle communication nodes, publishing (TX) and subscribing (RX) nodes between the low level controller Arduino and OBC.

The following figure illustrates the control loop implemented on the Arduino, OBC serially sends the number of steps and directions it wants the agent to move. The control block feeds the coefficients to the movement function sitting inside the microcontroller. The function then commands the H-Bridge to make drives moves up to desired steps Due to stochastic drift, the derivative of Δsteps is taken to feed it back to the summation node. After implementing the

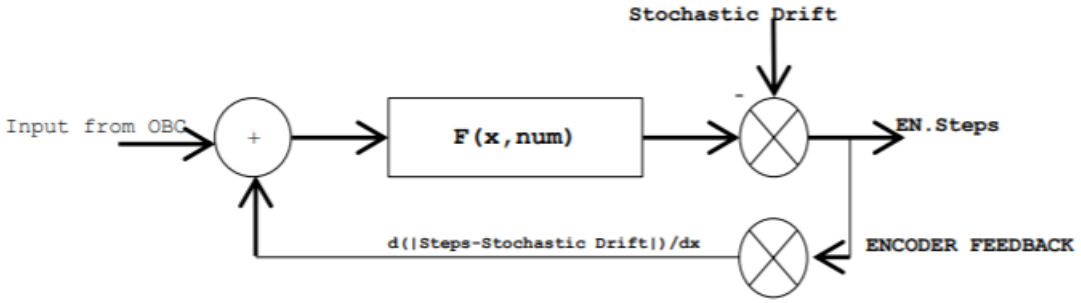

Figure 1.4: Arduino Control Loop

control loop, the ROS part is implemented on the OBC as shown in the figure: The system was

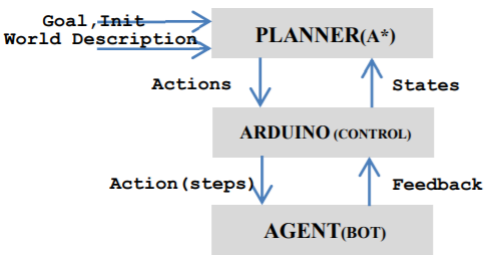

Figure 1.5: OBC Part Implementation

evaluated against the ideal simulation world and have found that the ground moves of the empirical hardware model are eventually satisfying in terms of per-step calculations, moving from one node to another and performing exact actions as required. However, the non-synchronization of motors is prone to error with a simple 1st order feedback loop which is used in the model.

1.5.1.2 Integration of ROS and RT tasks using message pipe mechanism on Xenomai for telepresence robot

In This paper[5], the Raspberry Pi 3, (Serves as the main controller M4K telepresence robot) was used to verify Practicality and flexibility on a solution to make Robot operating system (ROS) operate in real-time. since it does not operate in RT, which is a critical requirement to administer precise control periods and to meet necessary temporal deadlines of priority-based multi-tasking RT systems. The solution discussed was based on Xenomai, which is an RT dual-kernel approach of embedded Linux integrating ROS and real-time (denoted as RT for tasks) tasks through a communication interface termed, cross-domain datagram protocol (XDDP). The method is based on existing open source technologies allowing redistribution and easy integration of ROS to more complicated RT applications. Integration of ROS using XDDP on Xenomai: The method is illustrated in the following figure:

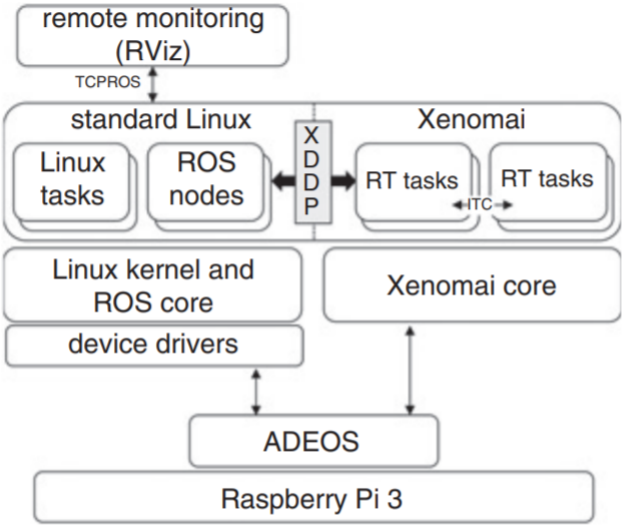

Figure 1.6: Realization of RT Control Application Integrating ROS and RT Tasks on Xenomai

-ROS Core runs on a standard linux kernel, handling ROS nodes. -Xenomai Runs alongside the standard Linux kernel through a hardware abstraction layer, termed adaptive domain environment for operating systems (ADEOSs) to enable both kernels to exist on the same machine. -MSW (mode switching) is turned on when ROS functions run inside an RT Task -XDDP is used instead of inter-task communication mechanisms to overcome MSW.

1.5.1.3 USE OF A RASPBERRY PI TO BUILT A PROTOTYPE WIRELESS CONTROL SYSTEM OF A MOBILE ROBOT

The main aim of this article[4] is to present the design and build wireless control system for the mobile robot with an omnidirectional wheel. The system uses a popular Raspberry Pi microcomputer with dedicated hardware and operator PC. The low level drive's control is supported

by Arduino Mega 2560. The design focused on the way of moving, the transmission medium and the method of controlling a robot. The advantage in using omnidirectional wheels is to achieve an off-road platform, while reducing the resistance occurring in the system (relative to the track drive) and simplifying the Structure (in the case of a drive with a pair of steered wheels). However, Robots using omnidirectional wheels are less efficient in the field and have worse traction on flat surfaces.

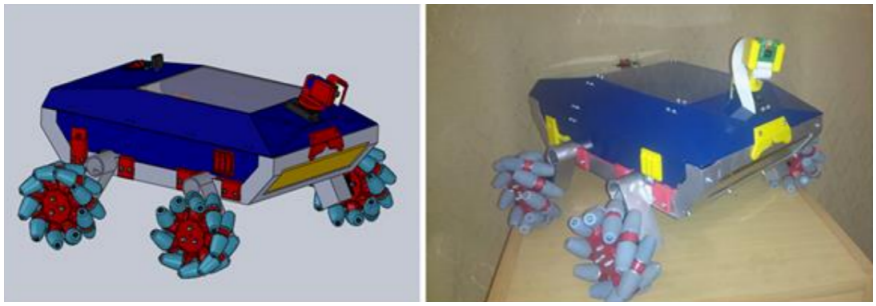

Figure 1.7: CAD Model and Real Model of Mobile Robot with Omnidirectional Wheels

The developed system consist of wireless control system and video transmission based on Raspberry PI, HD camera (image source), Arduino Mega 2560 as a microcontroller implements driver control system and Windows PC with wireless network adapter (as operator PC)(Robotics forum Forbot). System is shown in the following block diagram:

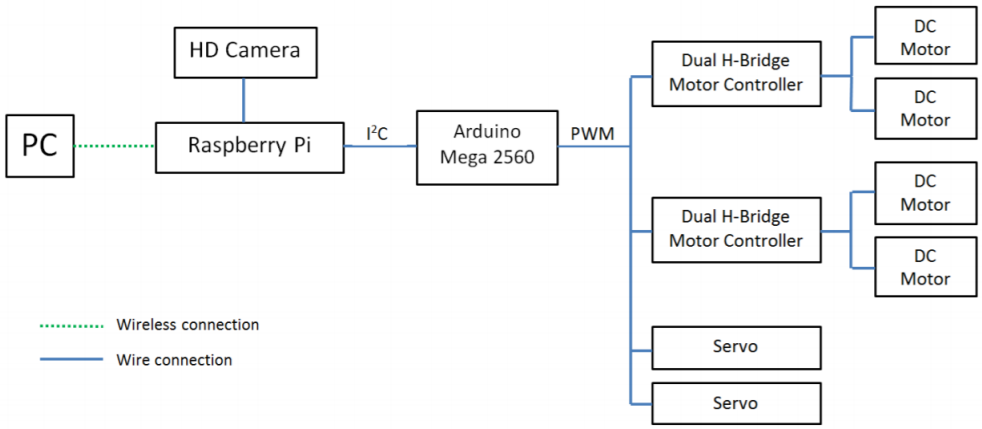

Figure 1.8: Block diagram of The Robot Control System

This solution allows for quick and cheap start prototype of the mobile robot To check its functional and driving parameters. Tests of the control system have shown some delays, for examples Streaming has a 1.5 s delay, and control data transmission has 1s delay. Transmission range in open space is 55 m, in a confined space 40 m.

1.5.1.4 Automatic Laser Interstitial Thermal Therapy for Robot-Assisted Surgery: Implementation

In this paper[10], An implementation of the Laser interstitial thermal therapy is presented, which include the thermal damage calculation, the thermal control at the edges, robot arm positioning and the laser ablation. Thermal sensors are installed at the border between healthy and tumorous tissues to prevent side effect. The form of the deformation caused by the laser ablation

is assumed to be as a sphere. The raspberryPi is used this time to simulate the laser ablation using the power distribution. The following figure describes the whole process of the system:

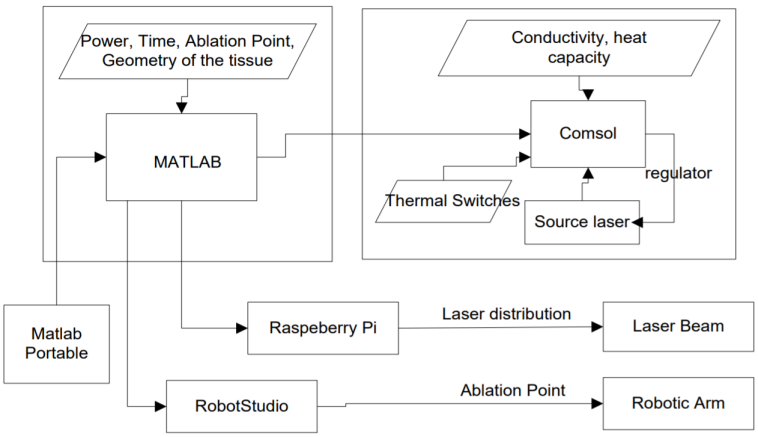

Figure 1.9: General Description of The Implementation of The Automatic Ablation Process

Comsole module: For a specific point, this procedure will check if the temperature at the edges exceed the temperature limit, and return the information to Matlab. It will also return the power distribution that has to be executed by the Led to have a safe laser ablation. RobotStudio module: This module will move the 6-axis robotic arm to the point for the laser ablation. Matlab Portable: all these applications can be run from Matlab Portable App on any Phone. The description was divided into 3 main sections: Hexagonal Close Packed (HCP) Structure Building stages of the HCP structure Construction of the topological diagram The implementation procedure was as following: 1. Modeling in COMSOL Multiphysics: to receive as input the laser power and the time, and then provide an estimated damage volume and an estimated sphere radius. 2. Bang-bang Controller: Events interface to control the heating process, by either allowing or stopping the source function depending on a temperature limit. 3. Matlab, Comsol simulation and robotic arm: Included the matlab codes to access comsol simulation, Matlab codes to control the Robotic Arm and Matlab codes to control the laser LED through a Raspberry PI B+.

1.5.1.5 Cloud Robotics in industry using raspberry Pi

Cloud Robotics is a field that integrates cloud technologies in robotics. The goal of this paper is to construct a robot with sensors and an alert system, which works within the control of the cloud. The robot to work on ROS- Robot Operating System.[15] -The system developed in this paper was proposed to overcome the disadvantages of the wired and manual systems that are heavily used in industry. A Raspberry-pi controller is used to control the movement of a locomotive Cloud Robot. An android device or computer using ROS controls the directions. The robot system consists of:
Sensors: camera, Temperature sensor, gas sensor, IR sensor.
Outputs: danger indication lamp, cooling fan and speaker.
Wi-Fi is used to send the data from the controller to Mobile and vice versa. The monitored data will be sent to the cloud. The following figures illustrates the entire system:

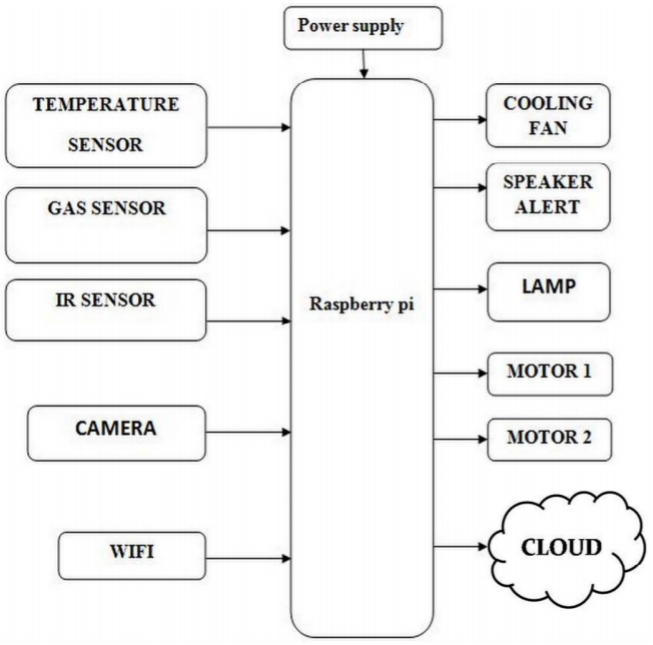

Figure 1.10: Transmitter Side Block Diagram

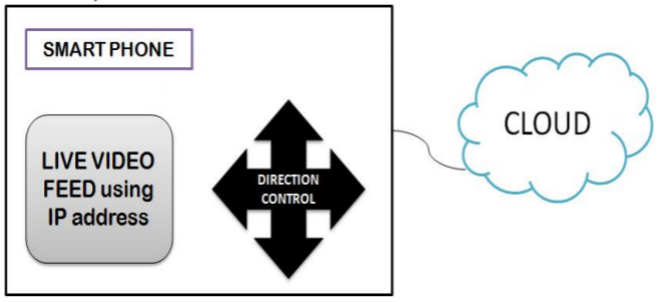

Figure 1.11: Receiver Side Block Diagram

The Robot that can move and monitor the entire industry from the inside. The mobile or computer at the receiver station is connected by a unique IP address. Therefore, the person at the receiver side controls the movement of the robot by operating ROS. The live video feed obtained by the camera is streamed to the receiver and viewed on the screen. -If Temperature level increases above the threshold value then the controller switches on the cooling fan. -If Gas level increases above the threshold value then the controller invokes the speaker announcement. -IR sensor signals if a person is trying to cross a restricted area and switches on the lamp. -All the data obtained in mobile is uploaded to the cloud. -The uploaded information can be viewed, saved from anywhere by entering the IP address.

1.5.2 Application of RFID Sensors in Mobile Robotics

An RFID device typically consists of radio frequency (RF) tags, a reader with one or more antennas, and software to process the tag readings. The reader interrogates the tags, receiving their ID code and other information stored in their memory. Tags can be either passive or active. Passive tags are activated by the electromagnetic field generated by the RFID antenna. Active tags, instead, are powered by an on-board battery (Finkenzeller, 2003).
Applications of RFID include inventory management, industry automation, ID badges and ac-

cess control, equipment and personnel tracking. Compared to conventional identification systems, such as barcodes, RFID tags offer several advantages, since they do not require direct line-of-sight and multiple tags can be detected simultaneously. Owing to these properties, RFID has recently found its way in the mobile robotics field, promising to contribute new solutions to data association problems in basic navigation tasks, such as localization, mapping, goal reaching, and item detection[9].

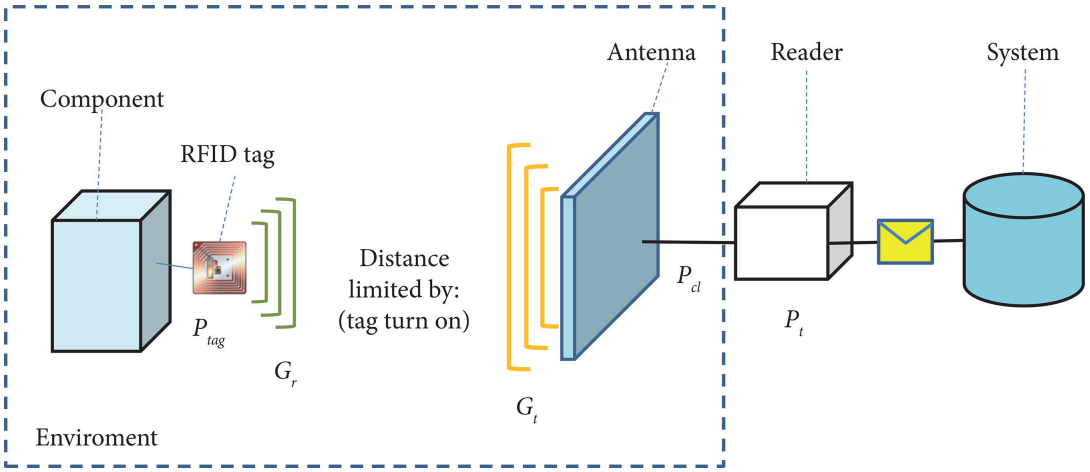

Figure 1.12: RFID System

1.5.2.1 Robot Navigation System with RFID and Ultrasonic Sensors

This paper[16] proposed a new navigation method for indoor mobile robots using RFID and ultrasonic sensors. The need for such navigation methods instead of using Global Navigation Satellite System (GNSS) is that GNSS is limited to open areas with satellite signals. The robot system is composed of a radio frequency identification (RFID) tag sensor and Ultrasonic sensors. The robot's environment has RFID tags scattered through it used as landmarks and the robot has a topological relation map of these tags. The robot moves automatically through its environment and scans for tags until it detects one. The robot then refers to the topological map for the next movement. A basic RFID system consists of an antenna or coil, a transceiver (with decoder) and a transponder (RF tag). The paper then goes on explaining the hardware implementation, it mentioned that L293D motor driver is used. The motor driver is mounted on a good quality, single sided non-PTH PCB. The pins of L293D motor driver IC are connected to connectors for easy access to the driver IC's pin functions. The following block diagram for robot navigation is provided:

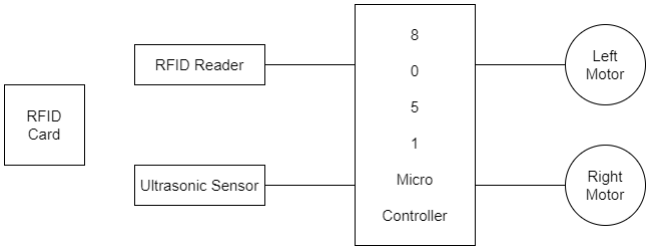

Figure 1.13: Block Diagram of Robot Navigation

So in summary, the proposed system was the following: The robot is consisting of the mechanical part, a computer, a RFID reader and an antenna, and ultrasonic sensors. The mechanical part is a

platform with wheels and motors which is controlled by the microcontroller. The RFID interrogator is connected to the computer via RS-232 serial port. The ultrasonic sensors are attached to the sides of the robot and used to measure the distance to walls. Since the area where tags can be detected at intersections is quite large, the robot has to use ultrasonic sensors to determine when to turn without collision to the wall. And the sensors will keep the robot out of collision when the hallway is not straight. The computer is in charge of processing the data from the RFID interrogator and ultrasonic sensors via serial ports and sending orders to the microcontroller to impact on the movement of the robot. Usually, localization of the robot and motion control strategy are the main problems that require solving in skilled navigation in mobile robots, and if the environment is unknown, then map building while moving also needs to be solved. These problems are solved in conjunction due to their interdependence. The implementation in this paper introduces a promising strategy to navigate a mobile robot within an unknown or uncertain indoor environment.

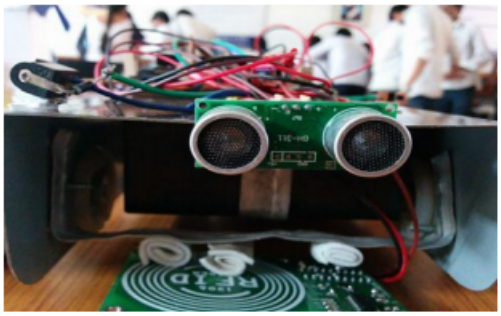

FIG.8.FRONT VIEW OF ULTRASONIC SENSORS

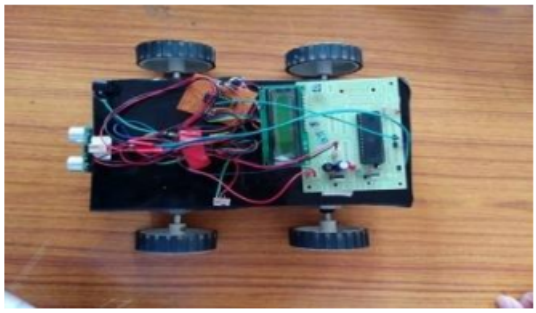

FIG.9. TOP VIEW OF OVERALL KIT

Figure 1.14: Top and Front View of The Overall Kit

1.5.2.2 An Autonomous Robot Navigation

This paper[8] presents a design of an autonomous vehicle for transportation of light weight equipment is presented. The robot will navigate its way around the RFID based floor without the help of a human. The objective of this project is to use data collected from RFID tags, which are embedded on the ground and use it to navigate a robot. The robot would be able to find specific locations and execute specific tasks based on the information that it collects from the tags. The vehicle is equipped with IR Transmitter and IR Receiver which is used to find the obstacle in the path of the Vehicle, destination position information is obtained by using GPS receiver, for dynamic destination changes a GSM modem is equipped. Controlling commands for Motor Driver like turn left, turn right, speed up and speed down etc., interfaced by using a low cost LPC2148 microcontroller into RFID. Then by using moving control commands which are read from already stacked RFID tags on the tracks the autonomous mobile robot will successfully reach the destination. In this paper the navigation task is subdivided into hurdle avoidance and goal seeking tasks. With the help of two front IR sensors Hurdle avoidance is achieved. The range data collected from these sensors are fed into the microcontroller. Goal seeking behavior involves the data from GPS, GPS receiver which is processed by another microcontroller. In this project for the demo purpose we are giving the goal as left, right, forward commands because in a room environment there will be small changes in the GPS location coordinates that robots may not consider as different locations . The main microcontroller fetches the desired data and generates motion commands for the robot. For selecting start and goal stations for robots inside the campus a GSM modem is interfaced to the main controller. For the system architecture, a four wheeled

car type vehicle robot is used. A SIM300D GSM modem is used for communication with GSM modem. An M89 GPS receiver is used to get the information of the robot.

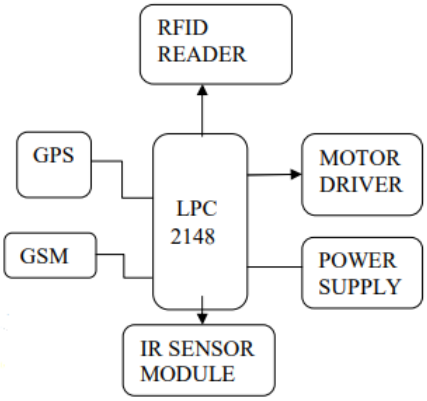

Figure 1.15: System Architecture

A compass is also needed so that the robot can calculate the angle between the direction it's facing and the direction towards the destination point. In addition, an RFID grid is used with tags placed separately in a way where they are in the range of the RFID reader. When the robot is first placed anywhere on the edge of the grid, facing the center, themicrocontroller obtains tag information from the reader. This tag information is the identification number found on the tag. The microcontroller then matches the retrieved tag information to a pre-assigned list of where this particular tag will be found on the grid. The tag value from the pre-assigned list is the information that will be sent to the NAV microcontroller. The converted tag information will now be labeled as the previous tag info and put into the navigation algorithm memory for use later in the process. The controller will analyze the current position values of the vehicle and compare it to destination position value and make the motors rotate such that it reaches the Destination. When going to the destination if there is any obstacle it will be observed by using IR Transmitter and Receiver and making the vehicle rotate and move to destination position. Once the vehicle reaches the destination the Buzzer will switch on.

1.5.2.3 RFID Sensor Modeling by Using an Autonomous Mobile Robot

Even though RFID technology has been available for more than sixty years. The ability of manufacturing the RFID devices and standardization in industries have given rise to a wide application of RFID technology only in the start of the 21st century. An RFID device consists of transponders installed in the environment, a reader with one or more antennas and a software for data processing. The RFID tags have a microchip and a small antenna, and they connect with the reader through the antennas and the software interprets the information stored in their memory. Effective reading, cheapness, and the line of sight with reader being unnecessary are some of the advantages of RFID tags, there could also be no need for internal power supply if passive tags are used which use the energy of the waves coming from the reader's antennas. Recently, the mobile robotics community showed interest in RFID technology and started investigating its potential application in critical navigation tasks. For example, RFID tags are employed as artificial landmarks for mobile robot navigation based on topological maps (Kubitz et al. 1997). In (Tsukiyama, 2005) the robot follows paths using ultrasonic rangefinders until an RFID tag is found and then

acts according to a topological map, and other examples like (Gueaieb & Miah, 2008) and (Choi et al., 2011). However, these approaches have requirements that cannot be always met in real environments, and due to their peculiarities, no sensor model is presented. On the other hand, there are undesirable effects that produce false readings, which makes modelling RFID sensors and localizing passive tags not that straight forward such as the RFID systems sensitivity to interference and reflections from other objects and the position of the tag relative to the receivers. Different algorithms that use different approaches were developed to model RFID system, recent ones consider tag detection event and received signal strength (RSSI) value. Many sensor models were based on probabilistic approach, a novel one was proposed in (Joho et al., 2009) where RSSI information and tag detection event are both considered to achieve a higher modelling accuracy. A sensor model is also illustrated in (Milella et al., 2008) which differs from the previous one in that they use fuzzy set theory instead of probabilistic approach.
In a paper titeled RFID Sensor Modeling by using Autonomous Mobile Robot[6], recent advances in fuzzy logic-based RFID modelling using an autonomous robot was presented. They followed the principle of the work by (Joho et al, 2009) but their approach was based on a fuzzy reasoning framework to automatically learn the model of the RFID device. They also put into consideration relative orientation between tag and antenna. Fuzzy C-Means (FCM) algorithm is applied to automatically cluster sample data into classes and also to obtain initial data memberships. This information is used to initialize an ANFIS neural network, which is trained to learn the RFID sensor model, which is defined as combination of an RSSI model and a Tag Detection Model. To build the sensor model, data is acquired and used to construct and learn the model using Adaptive Neuro-Fuzzy Inference System (ANFIS). The data is recorded by moving the robot equipped with antennas manually up and down in a corridor like environment after deploying a number of tags at different positions, the robot continuously records the relative distance and relative orientation of the antennas with respect to the tags and RSSI value and ID for each tag detection. A theodolite station computes the true tag locations whereas the robot positions are estimated using laser data and self-localization algorithm called Mixture-Monte Carlo Localization, hence, the relative position between tags and robot are known. So, the approach presented here for developing an RFID sensor combines an RSSI model and a tag detection model. The main contribution of their work concerns the supervised learning of the model to characterize the relationship between tags and antenna. They used FCM and ANFIS networks to learn the Fuzzy Inference Systems describing both SSM and TDM and the reliability of the model was proven by experimental tests.

1.5.2.4 Object Recognition Using a 3D RFID System

Robots nowadays can perform several tasks such as cleaning rooms and bringing objects to the human, like a mobile robot, let's say a command to bring the human's mobile phone to him is sent, the robot can scan the room all objects in the room using vision and sonar, etc. Then it tries to compare the objects with the target mobile phone. Once it finds a match, it grabs the target object and brings it to the user. However, if the target object were hidden behind other objects, the robot would not be able to detect it using vision or sonar, here the RFID technology can be very useful, as each object can have a built-in tag that allows the robot to distinguish it from other objects, and it can detect it even behind obstacles. The robot can then avoid the obstacles or move them using vision or sonar to get the target object to the human. This paper[12] proposed an advanced RFID system based on 3D tag. The proposed RFID system can identify the object, and estimate the object's position and orientation. Because of these characteristics, the robot with the automated systems can recognize objects easily and rapidly. Naturally, this

recognition mechanism can also simplify other robot processes such as localization, navigation, and manipulation. A robot can be informed of the existence of an object when its tag is detected by the antenna of the RFID system, however, the robot has to depend on other sensors. There are objects with new type of tags called 3D tag, which provides the position and orientation of the object. This allows the robot to easily estimate the pose of the object and can directly approach it without unnecessary scanning and sensing of other objects. The robot knows the size and shape and other characteristics of the object, which makes it simple to compare and match the real target and model. For the materialization of this concept, the 3D tag and the 3D RFID system were developed. The proposed RFID system is a part of the recognition module which is one of four modules that compose DRP I (Dynamically Reconfigurable Personal Robot I). The main function of the recognition module together with the sensor module is to recognize, and judge from the existing state of objects. This module also has vision and voice recognition, which are all synchronized and can work quickly thanks to the confirmation of the target from the RFID system. The proposed RFID system is composed of an antenna, which is swept by an actuator, and reader to detect the 3D tag, which is composed of several passive tags. These passive tags need to obtain their required power from electromagnetic wave coming from the antenna through electromagnetic induction, therefor, the tags must be within the range of the detectable angle and distance in the magnetic field, otherwise, they cannot be detected. The 3D tag is comprised of several tags called tag units, which are attached to six edges of a cube that is covered with radio shielding material. This shield limits the pitch angle for the detection of the unit tag and not the roll and yaw angles. That means when the antenna detects a tag units, it is facing one or two sides of the cube, for example, if it detects the tag unit on the top front, that means that it is facing the top or front or both. There are twenty four possible positions for the cube, and by detecting two of the unit tags attached to its six edges, it's position can be concluded. This RFID system can determine the existence, position and orientation of an object, which will considerably reduce the dependence of the robot on other sensors like vision systems required for object recognition.

1.6 Conclusion

After explaining the aim of our work, explaining robotics and providing some interesting findings in the fields of robotics and RFID technology, we will give a general overview of our system, explain the control and modeling of a differential drive mobile robot, describe path planning and the RFID system.

Chapter 2

Overview On the Proposed Platform Architecture and Control of of Differential Drive Robot

2.1 Introduction

In this chapter, we are going to have an overview about the overall architecture of our robot platform, we will mainly focus on the theoretical background behind our work. As stated in the previous chapter, Robot Operating System (ROS) will be used as the main framework of software development,development; hence our code will be under ROS philosophy and following the ROS conventions. The Following Figures illustrate our system.

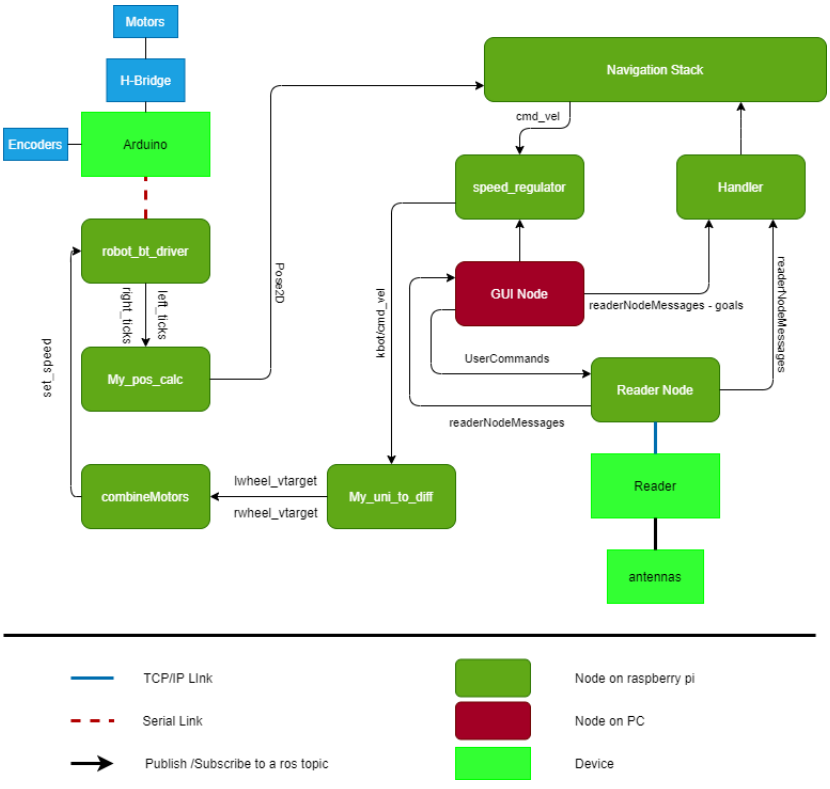

Figure 2.1: The Raspberry pi Implementation Architecture

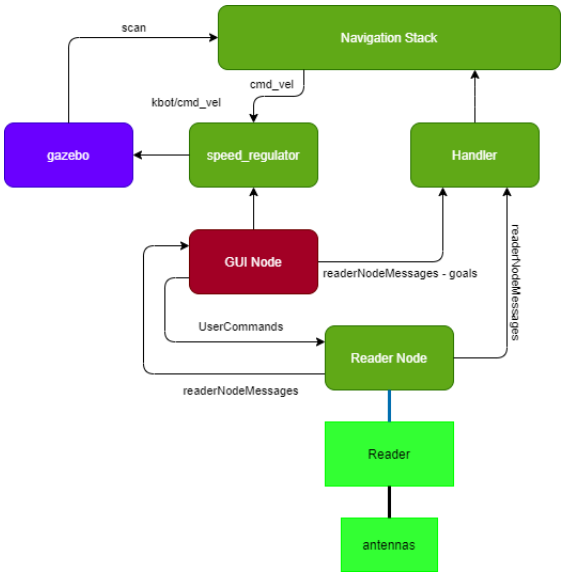

Figure 2.2: The HITL Implementation Architecture

The proposed solutions(HITL and Raspberry pi) are in multiple parts: An RFID reader connected to one or more antennas that reads the transponders data and sends it over TCP/IP link to a processor (raspberry pi, PC, ...) in XML format. In the raspberry pi architecture, the raspberry pi with the reader and the physical robot, it is connected to the Arduino through a usb serial link and to the RFID reader and a PC in local area network. Most of the ROS nodes of our system are running on raspberry pi. Arduino, which receives the output it should deliver to the motors and sends the sensors data through the usb serial link with the raspberry pi. In the HITL architecture, some of the ROS nodes running on raspberry pi are ran on the PC on which the simulation runs and robot is a simple 3D model of a differential drive robot running in a gazebo simulation. The system is summarized in figures 2.1 and 2.2 and is explained in detail in Chapter 3. In the following sections, we will talk about the RFID system, control of differential drive robot and path planning algorithms.

2.2 RFID System

2.2.1 The RFID Reader

2.2.1.1 Description

In order to detect and read the RFID tags in the environment, we need antennas connected to an RFID reader, and the one we used is the Ultra High Frequency (UHF) reader SIMATIC RF680R, which is intended for use in logistics and automation. The RF680R reader is intended for use in automation environment, for example on a production line but is equally suitable for applications in logistics. To meet these requirements, the reader is equipped with a high transmit power and degree of protection (IP65). The RF680R reader is either integrated without problems in SIMATIC S7 automation system via an integrated PROFINET connector or via the RS-422 interface and the ASM 456 communications module via PROFIBUS. Suitable programming blocks are available. The connection to PC environments is via Ethernet using TCP/IP and the XML protocol. A second Ethernet interface (both M12) can be used for diagnostics during operation so that the connection to the higher-level system does not need to be interrupted. The WBM (Web Based Management) allows commissioning, configuration and diagnostics of all three devices

using an Internet browser. This makes additional updates and installation of configuration and diagnostics software unnecessary[14].

2.2.1.2 Characteristics

The SIMATIC RF680R UHF reader has the following characteristics:

Antennas	4 x external antenna connectors
Transmit power	2000Mw
Digital input/outputs	4 x digital inputs and 4 x digital outputs
RS-422 interface	1 x plug M12 8-pin
PROFIBUS connection via CM	ASM 456 (115.2 kbps)
Ethernet interface	2 x Industrial Ethernet, M12 (TCP/IP with XML protocol or PROFINET)
Max. transmission speed	100 Mbps
Degree of protection	IP65
Configuration/ diagnostics options	WBM (browser) STEP 7 (S7)
Interfaces to PC/ controller	XML interface SIMATIC interface

The RF680R reader does not support IRT and cannot function as IRT conductor. The reader can be configured as a client in MRP rings. Network diagnostics via SNMP is supported by the reader.

2.2.1.3 Integration

The reader can be integrated in an IT environment as shown in the following figure:

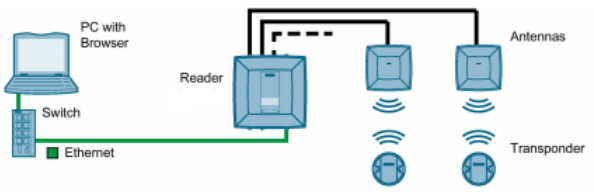

Figure 2.3: Integration of RFID reader in an IT environment

Or in an automation environment as shown in the following figure:

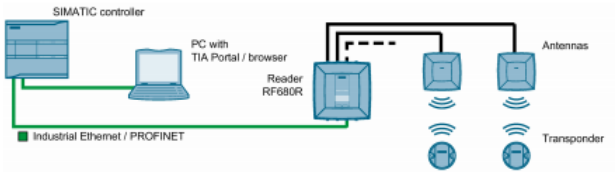

Figure 2.4: Integration of RFID reader in an automation environment

The first integration is the one we worked with in our project.

2.2.2 RFID tags

The RFID tags are labels that can be affixed to any object, they consist of three pieces: a micro chip (an integrated circuit which stores and processes information and modulates and demodulates radio-frequency signals), an antenna for receiving and transmitting the signal and a substrate. The

tag has a non-volatile memory in which the information is stored. The RFID tag includes either fixed or programmable logic for processing the transmission and sensor data, respectively.

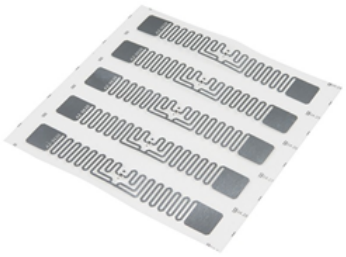

Figure 2.5: RFID tags

RFID tags can be either passive, active or battery-assisted passive. An active tag has an on-board battery and periodically transmits its ID signal. A battery-assisted passive has a small battery on board and is activated when in the presence of an RFID reader. A passive tag is cheaper and smaller because it has no battery; instead, the tag uses the radio energy transmitted by the reader. However, to operate a passive tag, it must be illuminated with a power level roughly a thousand times stronger than an active tag for signal transmission. That makes a difference in interference and in exposure to radiation. Tags may either be read-only, having a factory-assigned serial number that is used as a key into a database, or may be read/write, where object-specific data can be written into the tag by the system user. Field programmable tags may be write-once, read-multiple; "blank" tags may be written with an electronic product code by the user. The RFID tag receives the message and then responds with its identification and other information. This may be only a unique tag serial number, or may be product-related information such as a stock number, lot or batch number, production date, or other specific information. Since tags have individual serial numbers, the RFID system design can discriminate among several tags that might be within the range of the RFID reader and read them simultaneously. The RFID tags used in our project are the Simatic RF630L Smart-label variant.

2.3 Control of Differential Drive Mobile Robots

This section concerns the theoretical background on which our work was based. The robotics field is highly based on Control theory, which is influencing a system to follow certain behaviour,and system is anything that changes over time. In control Engineering, there are two methods of controlling a system, open loop control or closed-loop (feedback) control. Both methods are illustrated in the following figures

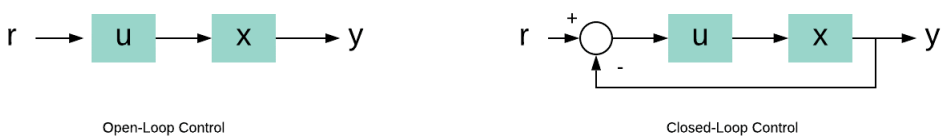

Figure 2.6: Types of Control

Where

- x: State = Representation of what the system is currently doing.
- r: Reference = What we want the system to do.
- y: Output = Measurement of (some aspects of the) system.
- u: Input = Control signal.
- Feedback = Mapping from outputs to inputs.

Before handling any system, we need to have a mathematical description of it, obtaining this description is called modelling, hence since our goal is to control a differential drive robot, we must model this type of robots, which will be explained in the following section.

2.4 Differential Drive Robot modelling and kinematics

One of the crucial steps in controlling a robot is to understand its Kinematics and dynamics. Robot kinematics refers to the study of the geometric relationship model of motion without considering the forces that affect the motion. Robot dynamics involves including all the forces in the modelling.

For any mobile robot, there are 6 degrees of freedom (6DOF), which are x, y, z , roll, pitch and yaw. As shown in the following figure:

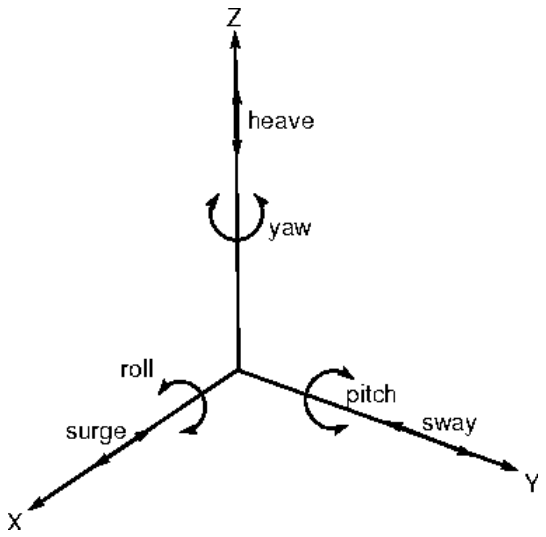

Figure 2.7: Stable Motion Planning for Dynamic Non-holonomic Mobile Robots

Where: (x, y, z): position in 3D plane (Roll, pitch, yaw): side wise rotation, forward backward rotation, and heading respectively.

Our robot is a differential drive robot so we are mostly interested in only the position along the x, y plan and the orientation yaw.

2.5 Robot differential Drive Model and Position Control

This model can be used in the prediction of a differential drive robot motion using changes in the left and right wheels speed, hence we can use it to determine the path and estimating the position of the robot. This method of describing motion is referred to as a kinematics approach, which ignores the causes of motion and focuses on the effects.

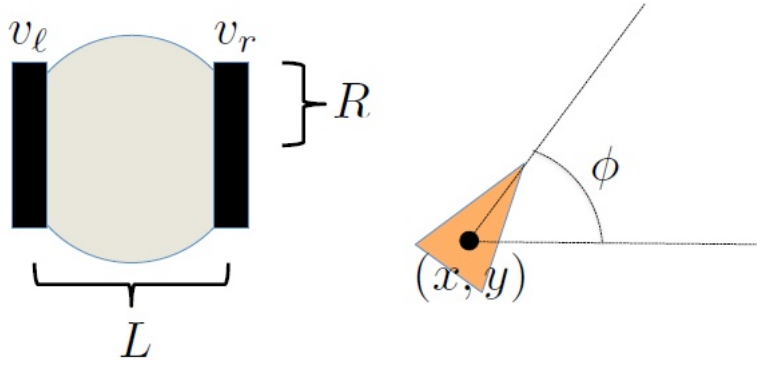

Figure 2.8: Differential Drive Mobile Robot Model

The final is model is described as following:

$$\begin{cases} \dot{x} = \frac{R}{2}(v_r + v_l)\cos\phi \\ \dot{y} = \frac{R}{2}(v_r + v_l)\sin\phi \\ \dot{\phi} = \frac{R}{L}(v_r - v_l) \end{cases} \tag{2.1}$$

where:

R: Radius of the wheels.

L:Distance between the two wheels.

$v_r and v_l$: Right wheel and left wheel velocities respectively.

θ: The heading of the robot.

$\dot{x}$, $\dot{y}$, $\dot{\theta}$: Changes of x, y and θ respectively.

This model is the model used for the implementation of the controller, the final inputs to the robot must be in terms of right and left wheel velocities, However it's not convenient for the design of motion algorithms, the best way is to deal with only the position and orientation ie. linear and angular velocities, and then finding a relationship to convert to the differential drive model. In this case, The Unicycle Model is the right option for design. The model is described as follows:

$$\begin{cases} \dot{x} = v\cos\phi \\ \dot{y} = v\sin\phi \\ \dot{\phi} = \omega \end{cases} \tag{2.2}$$

where:

v: Linear velocity.

ω:Angular velocity.

Now, we need to convert back to the differential drive model since the robot must receive left and right wheel velocities, since it can't deal with angular and linear velocities. To do that, we use the following equations:

$$\begin{cases} v_r = \frac{2v+\omega L}{2R} \\ v_l = \frac{2v-\omega L}{2R} \end{cases} \tag{2.3}$$

2.6 Robot Odometry

Odometry stands for the use of external sensors to estimate the state of the robot(x, y, tehta) relative to some initial conditions, an example of these sensors are Compass, accelerometer, gyroscope and wheel Encoders. In our platform we have mainly used wheel encoders to estimate the position of the robot, however more advanced techniques can be used such sensor fusion using both gyroscope and Encoders,

In our case we have used 2 optical encoders for each motor, then we have used the ticks of the Encoders(change in the state of the signal given by the encoder) on The differential Drive model. The signals generated by the optical encoders is illustrated in the following figure:

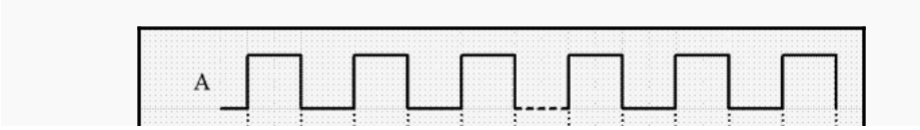

Figure 2.9: Optical Encoder signal

The encoder we used is only a one channel encoder, hence it can't give information about the direction of the rotation, we overcome this problem by software, in the arduino code we have set a flag that stores the direction of the motor according to the command given. We will discuss more details about that in the following chapter.

2.6.1 Getting Position from Encoder ticks

This is straightforward, using encoder ticks as we can get a direct relationship between the ticks and the distance moved by each wheel, hence the path of the robot. The relationship is defined as follows:

$$D = \frac{2\pi R \times \Delta ticks}{N} \tag{2.4}$$

Where:

Δticks: The number of ticks in a given sample of time.

N: The number of ticks per rotation.

Then we need to define the distance traveled by the center between the two wheels given the distance traveled by the two wheels, this is given as:

$$D_c = \frac{D_l + D_r}{2} \tag{2.5}$$

For a small sample time and by ignoring the friction, we can define a relationship between the current and next position as:

$$\begin{cases} x' = x + D_c \cos\phi \\ y' = y + D_c \sin\phi \\ \phi' = \phi + \frac{D_r - D_l}{L} \end{cases} \tag{2.6}$$

Cruise Controller: the Cruise-controller is a PID controller implemented to control the speed of each wheel ie; Vl and Vr. **Behavior-based Robotics:** The technique at which our robot platform was developed, is based on the behavior based approach which is about developing useful controllers and switching between them depending on the environmental changes.

2.7 Go to goal algorithm

According to Al-Taharwa[ref him], path planning is an optimization problem according to definition that, in a given mobile robot and a description of an environment, a plan is needed between start and end point to create a path that should be free of collision and satisfies certain optimization criteria such as shortest path. This is true but only if the purpose of the solution that we are looking for is the shortest path, which is not always the case since sometimes the shortest path may require complex algorithms making the calculation to generate output longer, and thus the result wouldn't be the most efficient. So, various approaches have been introduced which are according to the environment, sensor type etc. Path planning algorithms can be categorized into static dynamic according to whether the robot's environment contains static or moving obstacles, and global or local according to whether the information about the environment is already known to the robot or not.

2.7.1 Global path planning

In global path planning, the robot must already possess information of the environment before planning its path to the goal destination. In this approach the algorithm generates a complete path from the start point to the destination point before the robot starts its motion.

2.7.2 Local path planning

In local path planning, the robot moves in an unknown or dynamic environment and the path planning algorithm has to respond to the unpredictable changes in the environment. In other words, local path planning is a real time obstacle avoidance algorithm using data acquired by the robot's sensors regarding contingency measures that affect its navigation safety. A robot using a local path planning would go in a straight line towards its goal until it detects an obstacle. The robot then deviates from the straight line to avoid the detected obstacle while updating important information such as the distance from its current position and its goal position. To ensure that the robot reaches its goal accurately, it must always know the position of the target point.

2.8 Path Planning with RFID

Instead of depending on the visual sensors data to make path planning decisions, the robot can obtain information about the environment and the obstacles around it from RFID tags attached to them. The tags can provide information about the location of the obstacles or instruction for the robot on how it can avoid them and get to its destination. One of the advantages of RFID sensors over the visual sensors is that they don't need the tags to be in sight to read them, they can read them even behind obstacles or in the absence of light in the environment which is crucial for a safe navigation of the robot.

2.9 Real time reading of RFID information

The antennas of the RFID reader can read in real time any RFID tag within its detection range. In order to achieve that with the Simatic RF685F reader and send the detected tag to the application in an xml format, we need to send it a triggerSource command with "Start" as the value of triggerMode parameter and the name of the antenna in the sourceName parameter. The triggerSource command looks as shown in appendix B.

2.10 Conclusion

After explaining the RFID system, the control of the differential drive robot, the algorithms used for path planning and having a general idea of the system's architecture, we will dive into each block in our system, understand the low level design and have a general overview of ROS in the next chapter.

Chapter 3

Platform implementation

3.1 Introduction

In this chapter we will go through a general overview of ROS where we will explain important concepts and tools related to it, specially the ones we used in our work. We will define HITL, explain ROS navigation stack, the graphical user interface that we made, the hardware implementation and every ROS node in our system and the Arduino code.

3.2 General overview of 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. On the seventh of November, 2007, the first commit of ROS code was made to SourceForge by Willow Garage, which was started in January of the same year. However, the vision behind the project and its early days were at Stanford before 2007 by two PHD students named Eric Berger and Keenan Wyrobek who were then invited to work at Willow Garage by its founder Scott Hassan who shared the same vision of a Linux for robotics.

3.2.1 Computation graph model

The processes that ROS runs are represented as nodes, which are connected by what's called topics. Each node can publish to a topic or subscribe to it to receive the messages published to it. A node can also provide or call a service. Before running any node though, a process called ROS Master must be running on its own terminal as it is responsible for registering nodes to itself, setting up node-to-node communication for topics, and controlling parameter server updates. It does that by setting up peer-to-peer communication between all node processes after they register themselves with it. The ROS architecture is decentralized, meaning that nodes running on different computers on a network can work properly no matter on which computer the ROS master is, as long as the networking part is done correctly. [2]

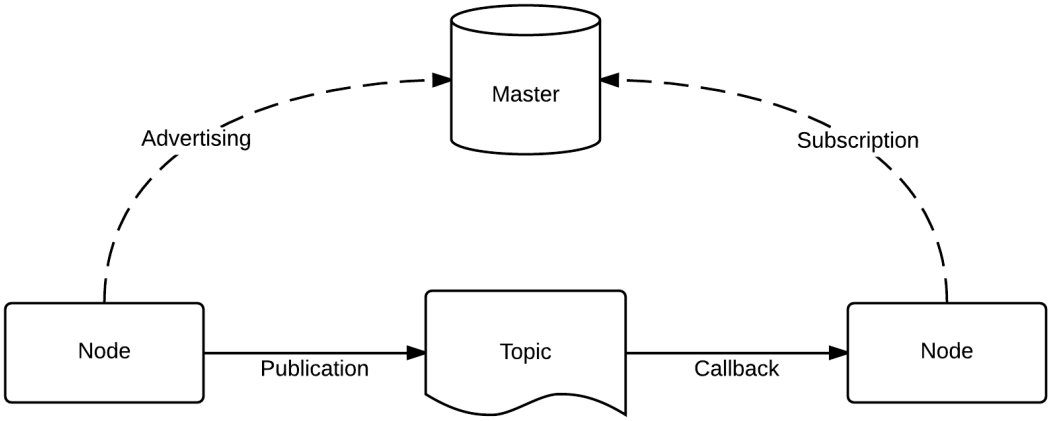

Figure 3.1: ROS Master Operation Diagram

Node: A node represents a single process, it has a unique name and communicates with other nodes through messages. Its logic can be written using either C++ or python.
Topic: A topic is a channel through which the nodes communicate via messages. It can be subscribed to or published into by the nodes.
Message: A message is what nodes publish to topics to communicate, each message has a type, like string or odometry. A message type that requires explanation here is the Twist message.
Twist message type: geometry_msgs ROS package is a ROS package that includes many message types that are used for Robot and kinematics. ROS Twist message is ROS message type that encodes motion in 6 degrees of freedom — 3 as linear motion and 3 as angular as following:
geometry_msgs/Vector3 linear: constituted of 3 float64 variables for linear velocities, x,y and z.
geometry_msgs/Vector3 angular: constituted of 3 float64 variables for linear velocities, x,y and z.
Service: A service represents an action that a node can take which will have a single result. It is advertised by a node.
Package: A package is a compilation of nodes that can easily be compiled and ported to other computers.

3.2.2 Tools

Rviz: rviz is a three-dimensional visualizer used to visualize robots, the environments they work in, and sensor data. It is a highly configurable tool, with many different types of visualizations and plugins.
Catkin: catkin is the ROS build system, having replaced rosbuild as of ROS Groovy. catkin is based on CMake, and is similarly cross-platform, open source, and language-independent.
Rosbash: The rosbash package provides a suite of tools which augment the functionality of the bash shell. These tools include rosls, roscd, and roscp, which replicate the functionalities of ls, cd, and cp respectively.
Roslaunch: roslaunch is a tool used to launch multiple ROS nodes both locally and remotely, as well as setting parameters on the ROS parameter server. roslaunch configuration files, which are written using XML can easily automate a complex startup and configuration process into a single command. roslaunch scripts can include other roslaunch scripts, launch nodes on specific machines, and even restart processes which die during execution.

3.2.3 ROS Distributions:

Since its release, ROS have had several distributions, here are some of the popular ones:
Indigo Igloo, released on July 22nd 2014
Kinetic Kame released on May 23rd 2016
Melodic Morenia, which is the one we used in our project, released on May 23rd 2018
Noetic Ninjemys, last ROS1 release, released on May 23rd 2020

3.3 Hardware In The Loop Simulation

Hardware in the loop (HITL) testing is a technique where real signals from a controller are connected to a test system that simulates reality, tricking the controller into thinking it is in the assembled product. Test and design iteration take place as though the real-world system is being used. You can easily run through thousands of possible scenarios to properly exercise your controller without the cost and time associated with actual physical tests.
The way we used HITL is by reading real RFID tags using the SIMATIC RF680R RFID reader and feed the red information to our system in a PC where the simulation runs in gazebo. In order to implement our work in a hardware in the loop simulation, we made a 3D model of the robuter ulm, an environment and tweaked the ROS navigation stack parameters to suit our robot. Then we built a map of the environment using slam gmaping and a laser scanner. Finally we programmed the decision making of the robot based on the RFID tags it receives. In this section we will explain these steps.

3.3.1 Robot model and virtual environment

The robot model can be described in one of the following file formats: sdf, urdf or xacro. The one we used is xacro, where the robot parts are described in link tags and are attached to each other using joint tags, where a joint attaches a child link to a parent link. One advantage of using xacro is that we can describe repetitive links and joints such as wheels in macro tags while defining parameters that change depending on the link such as the joint origin. We can then just call it and set the parameters as needed. The link in our robot are:
base_footprint: A dummy link modeled as a tiny box with heavy weight that represents the center of the robot
Base_link: represents the base of the robot to which other links are attached. It is modeled as a box.
Caster_wheel: a wheel modeled as a sphere to keep the balance of the robot.
Right_wheel and left_wheel: The two controlled wheels of the robot, they are modeled as a cylinder.
Sensor_laser: a laser sensor used to scan the environment, it is modeled as a small box.
The way these links are linked is shown is the following figure:

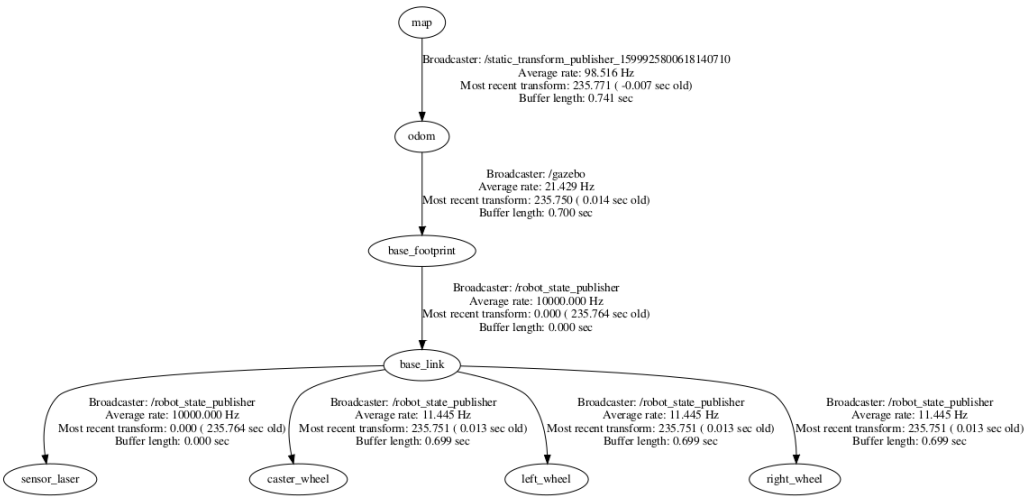

Figure 3.2: Robot TF Tree

This is called the transform tree, which is generated using tf package. We will explain tf in the next section. The visual and collision of the links are described in the <visual> and <collision> tags respectively using either a simple shape such as a box or a cylinder or by linking the location of a mesh file like sdl or dae. In our case, we used simple shapes as mentioned earlier.

Gazebo building editor is a tool in gazebo that allows us to easily build an environment for our robot, we used this tool to build a simple closed room environment in which we will spawn our robot later. In order to control the robot in the simulation and receive its odometry and laser sensor's data, we added diffdrive_plugin and gazebo_ros_head_hokuyo_controller plugins to our xacro file. These are gazebo plugins that give our URDF models greater functionality and can tie in ROS messages and service calls for sensor output and motor input. Diffdrive_plugin is a model plugin which is inserted in the URDF inside the ¡robot¿ element. It is wrapped with the ¡gazebo¿ pill, to indicate information passed to Gazebo. Upon loading the robot model within Gazebo, the diffdrive_plugin code will be given a reference to the model itself, allowing it to manipulate it. Also, it will be give a reference to the SDF element of itself, in order to read the plugin parameters passed to it. Specifying sensor plugins is slightly different. Sensors in Gazebo are meant to be attached to links, so the ¡gazebo¿ element describing that sensor must be given a reference to that link. Upon loading the robot model within Gazebo, the gazebo_ros_head_hokuyo_controller code will be given a reference to the sensor, providing access to its API. Also, it will be give a reference to the SDF element of itself, in order to read the plugin parameters passed to it.

TF: TF is a package that lets the user keep track of multiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time.

Mapping: In order for the robot to do global path planning, it needs a map of the environment it is in. So we built it using gmapping ros package. The way it works is by first launching the slam_gmapping node and providing the topic where the laser_scanner publishes its data as a param. Then we guide manually the robot in the environment while monitoring the map that is being built in rviz. Here is how it should look:

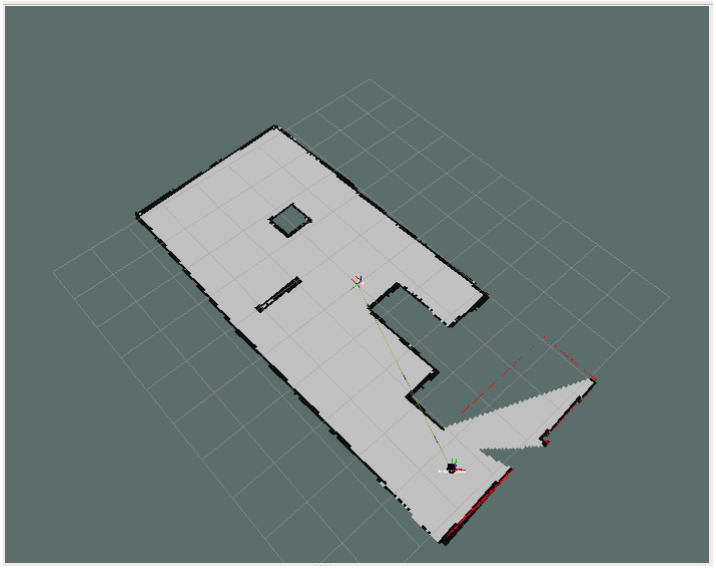

Figure 3.3: Map Building Visualization in RVIZ

Once we are satisfied with the result, we run the following command to save the map.

```
rosrun map_server map_saver -f mymap
```

This command generates two files: mymap.pgm and map.yaml that we can import using the command:

```
rosrun map_server map_server mymap.yaml
```

Or just add the map server node to a launch file as we did, where we added it to the move_base.launch file.

3.3.2 Navigation Stack

The Navigation Stack is fairly simple on a conceptual level. It takes in information from odometry and sensor streams and outputs velocity commands to send to a mobile base. Use of the Navigation Stack on an arbitrary robot, however, is a bit more complicated. As a pre-requisite for navigation stack use, the robot must be running ROS, have a tf transform tree in place, and publish sensor data using the correct ROS Message types. Also, the Navigation Stack needs to be configured for the shape and dynamics of a robot to perform at a high level. The configuration files for our robot can be found in the appendix.

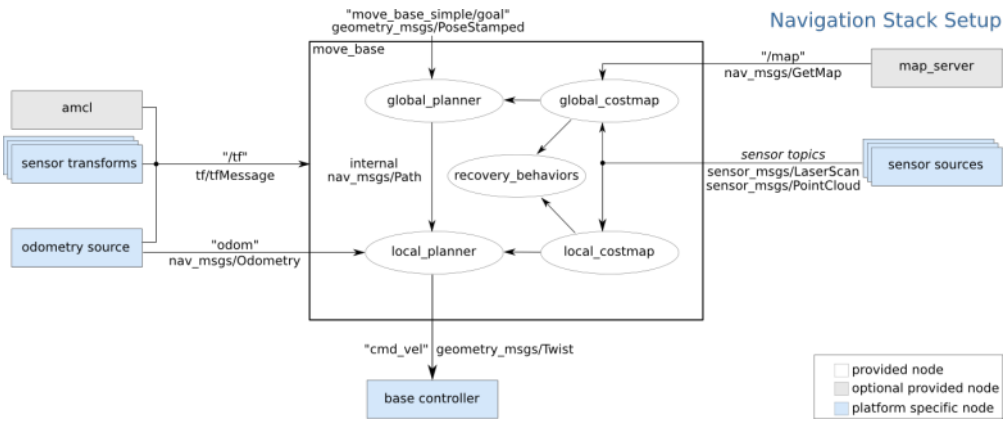

Figure 3.4: Navigation Stack Setup

The navigation stack assumes that the robot is configured in a particular manner in order to run. The diagram above shows an overview of this configuration. The white components are required components that are already implemented, the gray components are optional components that are already implemented, and the blue components must be created for each robot platform. The navigation stack has pre-requisites which are:

- The robot runs with ROS
- The robot be publishing information about the relationships between coordinate frames using tf.
- Sensor information which in our case is the laser scanner information published in the laser_scanner topic.
- Odometry information published using tf and the nav_msgs/Odometry message.
- The navigation stack can send velocity commands using a geometry_msgs/Twist message on the "cmd_vel" topic.

3.3.3 Localization

The robot uses Adaptive Monte Carlo Localization (amcl) to localize it self in the map. Amcl is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox), which uses a particle filter to track the pose of a robot against a known map. amcl takes in a laser-based map, laser scans, and transform messages, and outputs pose estimates.

3.3.4 Path Planning

Navigation stack uses global planner and local planner for its navigation, the global planner finds the shortest path to the destination whereas the local planner executes it. The global planner uses D* by default which can be changed and the local planner uses the Dynamic Window Approach and trajectory rollout by default.

3.3.5 Integration of RFID

The robot takes decisions about the about the next goal it goes to, stops or change its speed based on the RFID tags the RFID reader reads and the source antenna it reads it from. The process of choosing which destination the robot should go or whether it should stop is handled by the handler.py node in the r_gui_h package. The handler node is subscribed to the /readerNodeMessages topic and will send goals to the move_base node or cancel them based on the messages it receives from the reader node.
The FYPv3 node is also subscribed to the readerNodeMessages and will display information about the object to which the received tag is attached to on the Graphical User Interface's (GUI) consoles.

3.3.6 Reader Node

This node handles the communication with the RFID reader, it is subscribed to the "UserCommands" where the commands required to be sent to the reader are published by the GUI node. The node has access to xml files stored on the resources/cmd forlder on the r_gui_h package. Each

file contains the xml structure of a reader command with placeholders in each parameter. Once a command is published by the GUI node in the userCommands topic, the reader node receives it and reads the necessary xml file, replaces the placeholders with the appropriate values then sends the command to the reader and waits for a reply. Once it receives a reply, it publishes it to the readerNodeMessages topic. Another thing worth mentioning here is the way this node deals with tag event reports when the reader is continuously triggered. The node in this case should continuously be ready to receive reports from the reader and at the same time listen for the user command that stops the reader triggering. Thus, we used multithreading where in the main thread the node receives the reader reports and on another thread, it subscribes to "UserCommands2" topic where the "stop trigger" command is published.

3.3.7 Handler Node

The handler node sends a goal(coordinates on the map) to the ros navigation stack or makes the robot stop based on the RFID tag it read and the antenna it read it from. The way it does that is by using client-server interaction on ROS. The ActionClient and ActionServer 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.

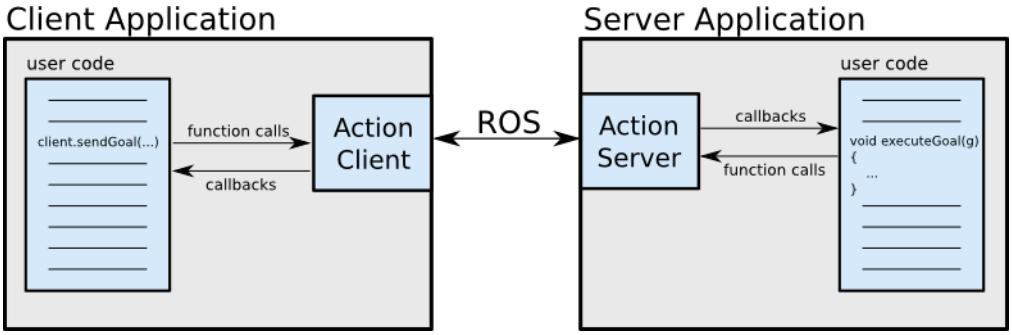

Figure 3.5: Home Page of The Graphical User Interface

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:
Goal: To accomplish tasks using actions, we introduce the notion of a goal that can be sent to an ActionServer by an ActionClient. In the case of moving the base, the goal would be a PoseStamped message that contains information about where the robot should move to in the world.
Feedback: Feedback provides server implementers a way to tell an ActionClient about the incremental progress of a goal. For moving the base, this might be the robot's current pose along the path.
Result: A result is sent from the ActionServer to the ActionClient upon completion of the goal. This is different than feedback, since it is sent exactly once. This is extremely useful when the purpose of the action is to provide some sort of information. For move base, the result isn't very important, but it might contain the final pose of the robot.
The handler node sends a goal if an RFID tag is read from the back_antenna and represented a movable, passive, light weight and small object. It cancels all goals if an RFID tag is read from other than the back_antenna and represented a human or a movable obstacle. If it's a movable obstacle, a message pops on the GUI console asking for the obstacle to be removed. The handler

node will not send any goals until "all clear" button on the GUI is clicked.

3.3.8 Speed_regulator Node

The speed_regulator node takes the linear and angular velocities published on the cmd_vel topic by either move_base node or GUI node and multiplies them by a number to increase or decrease the speed of the robot. It then publishes the result as a Twist message on the kbot/cmd_vel topic to which the kbot is subscribed.

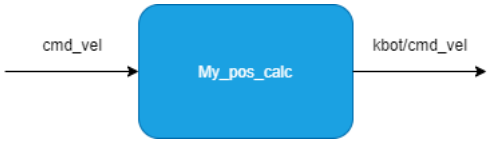

Figure 3.6: Home Page of The Graphical User Interface

3.3.9 GUI Node

In order to make sending and receiving data to and from the simatic rfid reader and the arduino uno easier for the user, a gui was made using pyqt5 framework. As the gui's window is opened, the user is faced with home page with 3 buttons, Reader button which leads to the reader interface that allows the user to communicate with the reader and send commands in xml format as shown in the reader documentation[1], Manual button which leads to the manual interface that allows the user to control the robot manually, Autonomous button which leads to the autonomous interface.

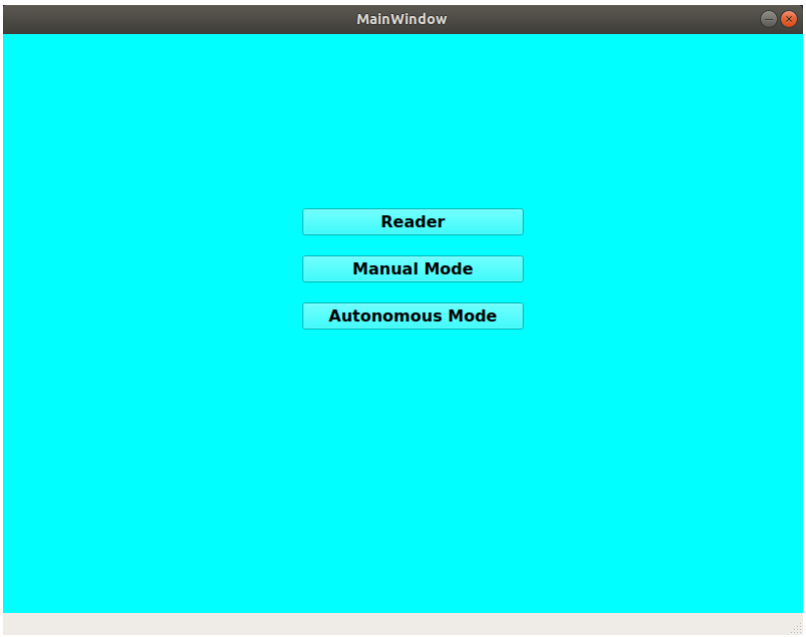

Figure 3.7: Home Page of The Graphical User Interface

3.3.9.1 Reader Interface

The reader interface is displayed after the user presses the reader button in the home screen, the user then is presented with a menu of buttons that would send a command to the simatic rfid reader or display a menu with more commands.

In the reader interface the user can also simulate the receiving of a tag by pressing the simulate tag sending where the gui node takes the value that the user enters in the plainTextEdit above the button and publishes it to the tags topic, and because a tag id is a 24 digits hexadecimal number, then if the user enters a non hexadecimal number or the number of digits in the hexadecimal number isn't 24, a message box pops up asking him/her to enter a correct tag id. A console is also displayed in the reader interface which displays messages sent from the rfid reader or the arduino.

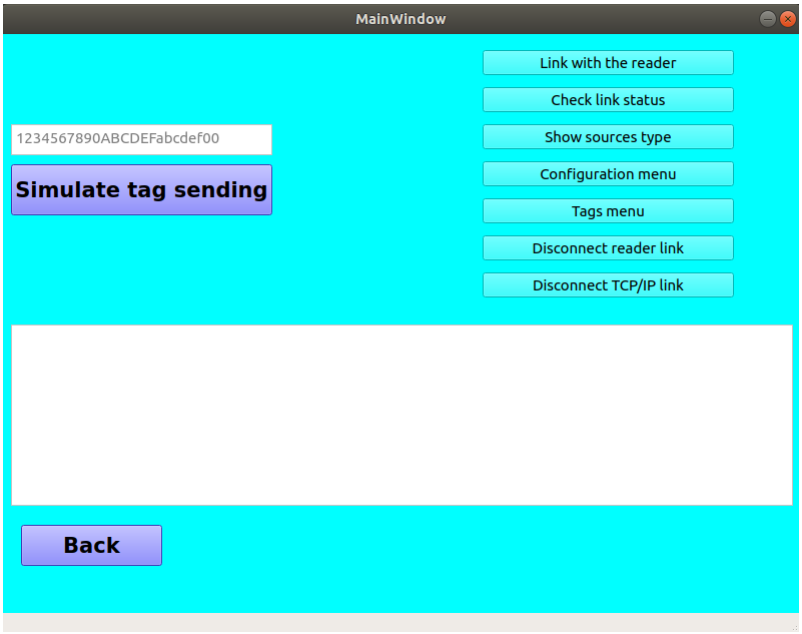

Figure 3.8: Reader Interface of Graphical User Interface

Each command sent by the user application is replied to by the reader with a reply frame. If the command was executed successfully, the reply frame has the value "0" in the "ResultCode" parameter. If other values are returned in this parameter, it means that the command was not executed successfully. In this case, the return value corresponds to the error code. In our project, it's not the gui ros node that sends the xml format of the command to the reader via the TCP/IP link, it's the reader node that does that after the gui node sends the name of the command to the it via "UserCommands ros" topic. The commands in the commands menu are as follow:

Link with the reader: The "link with the reader" button sends a hostGreetings command, which is necessary at the start of all communication with the reader. If commands are sent without being preceded by the "hostGreetings" frame, the reader replies with the error message "ERROR_INVALID_READER_STATUS".

Check link status: The "check link status" button sends a heartbeat command, which checks whether or not the connection is interrupted (e.g. wire break) or whether the reader is out of operation. After this command is sent, the reader blocks connection requests of new clients for 30 seconds by sending "heartbeat" command periodically within a period of 30 seconds, which allows us to make sure that no other unwanted user applications access the reader.

Show sources type: The "show sources type" button sends getAllSources command, which queries the names of all configured read points of the reader.

Configuration Menu: The "configuration menu" button takes the user to the configuration menu interface, which contain a list of commands related to the reader configuration.

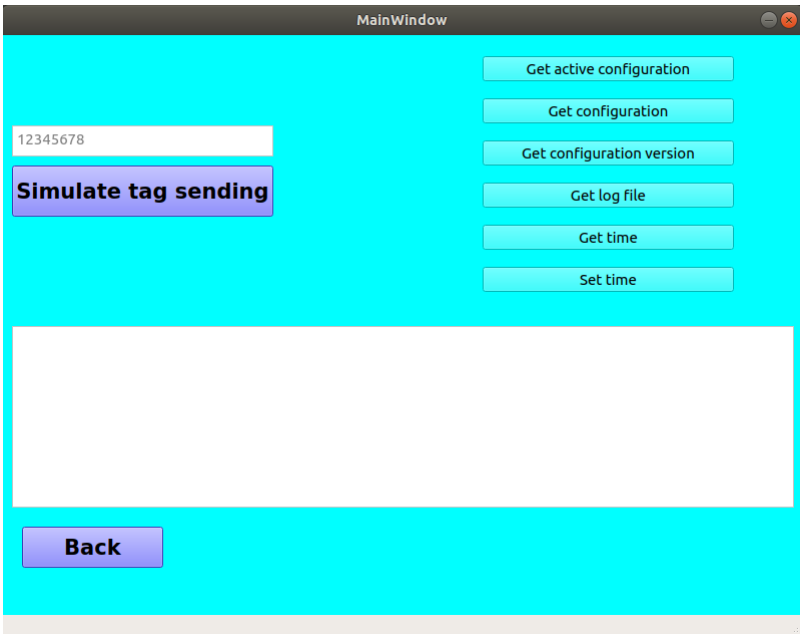

Figure 3.9: Configuration Menu

Get active configuration: The "get active configuration" button sends getActiveConfiguration command, which requests a configuration file from the reader containing the active parameters that the reader is currently working with.
Get configuration: The "get configuration" button sends getConfiguration command, which requests the configuration stored on the reader.
Get configuration version: The "get configuration version" button sends getconfigVersion command, which requests the version of the configuration stored on the reader.
Get log file: The "get log file" button sends getLogfile command, which requests the log from the reader. The reply frame of the reader can take up to 20 seconds.
Get time: The "get time" button sends getTime command, which requests the current time stamp of the internal reader clock.
Set time: The "set time" button sends setTime command, which sets the internal reader clock.
Tags menu: The "tags menu" button takes the user to the tags menu interface that contains a list of commands related to the RFID tags.

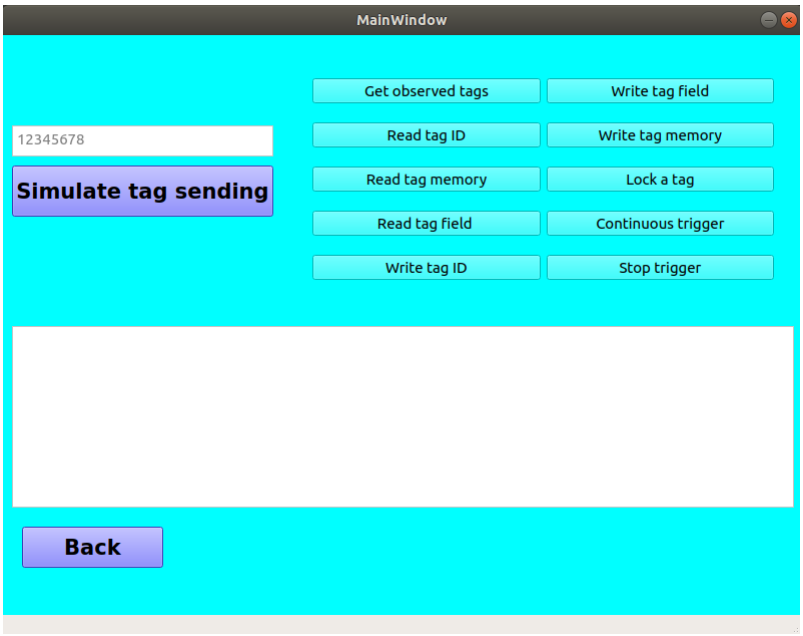

Figure 3.10: Tags Menu

Get observed tags: The "get observed tags" button sends the getObservedTagIDs command, which returns all identified transponders from a selected read point. If no transponder was identified with the "Observed" status, a positive reply without transponder data is returned.
Read tag ID: The "read tag ID" button sends the readTagIDs command, the selected read point takes an inventory and returns all identified transponders in the reply frame. If no transponder was identified, a positive reply without transponder data is returned. This command remains active during the entire duration specified in the command frame.
Read tag memory: The "read tag memory" button sends the readTagMemory command, which reads data from the requested transponder.
Read tag field: The "read tag field" button sends the readTagField command, which reads data from the selected transponder. The address of the data area is specified by the name of a tag field which is specified using the WBM.
Write tag ID: The "write tag ID" button sends the writeTagID command, which writes a new EPC-ID to the transponder. There must be only one transponder in the antenna field to ensure clear identification when writing the ID, otherwise, a negative reply is returned.
Write tag field: The "write tag field" button sends the writeTagField command, which writes data to the selected transponder. The address of the data area is specified by the name of a tag field which is specified using WBM.
Write tag memory: The "write tag memory" button sends the writeTagMemory command, which writes data to the requested transponder. If no EPC-ID is made available or detected, the command is executed with all the transponders recognized from the read point.
Lock a tag: The "lock a tag" button sends the lockTagBank command, which locks the memory area of the selected transponder. If no EPC-ID is made available or detected, the command is executed with all the transponders recognized from the read point.
Continuous trigger: The "continuous trigger" button sends the triggerSource command with the "Start" parameter in the value_triggerMode tag, which makes the selected read point continuously triggered until a stop command is sent. The reader then will send a tagEventReport whenever it detects an RFID tag with the tagID of the detected tag.

Stop trigger: The "stop trigger" button sends the triggerSource command with the "Stop" parameter in the value-triggerMode tag, which stops the triggering of the selected read point and will no longer send tagEventReport when an RFID tag is detected.
Disconnect reader link: The "disconnect reader link" button sends the hostGoodbye command, which ends the communication with the reader and terminates the TCP/IP connection.
Disconnect TCP/IP link: The "disconnect TCP/IP link" button hardly disconnects the TCP/IP link between the client and the reader.

3.3.9.2 Manual Mode

In the manual mode interface, the user is faced with 4 buttons: forward, backward, left and right which allows the user to drive the robot forward or backward and turn it left or right respectively. Any messages sent by the reader or the Arduino are displayed in the console below the 4 driving buttons.

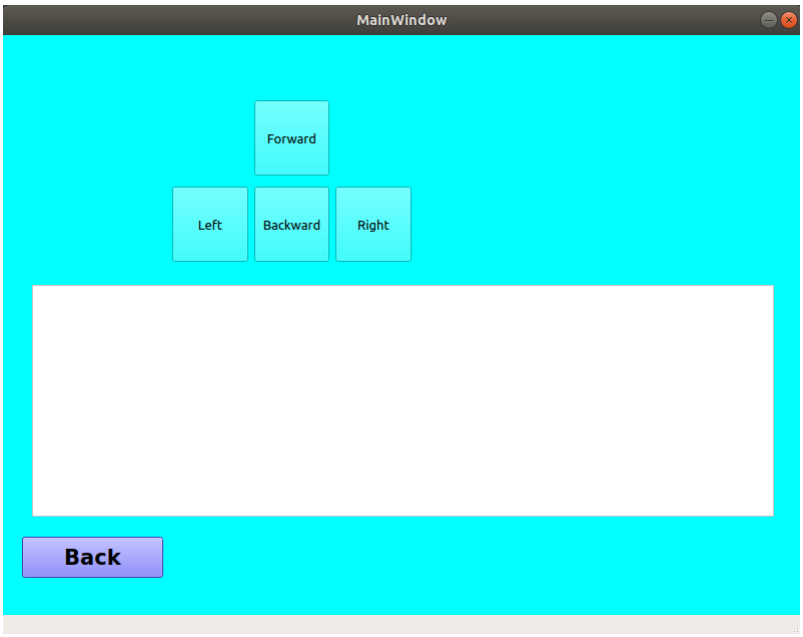

Figure 3.11: Manual Mode Interface

When the user clicks one of the driving buttons, a twist message is published on the cmd_vel ros topic, which would cause either a robot with a raspberry pi with a node subscribed to the cmd_vel topic or a differential drive robot in a gazebo simulation to move accordingly.

3.3.9.3 Autonomous Mode

In the autonomous mode interface, the user is faced with a text field, a go button and a console that displays messages from the reader and the Arduino. The user can enter the coordinates of the goal destination of the robot and press go, the coordinates are then published to the ros topic to which the handler node is subscribed. The handler node will handle the process of moving the robot from its current position to the desired destination submitted by the user.

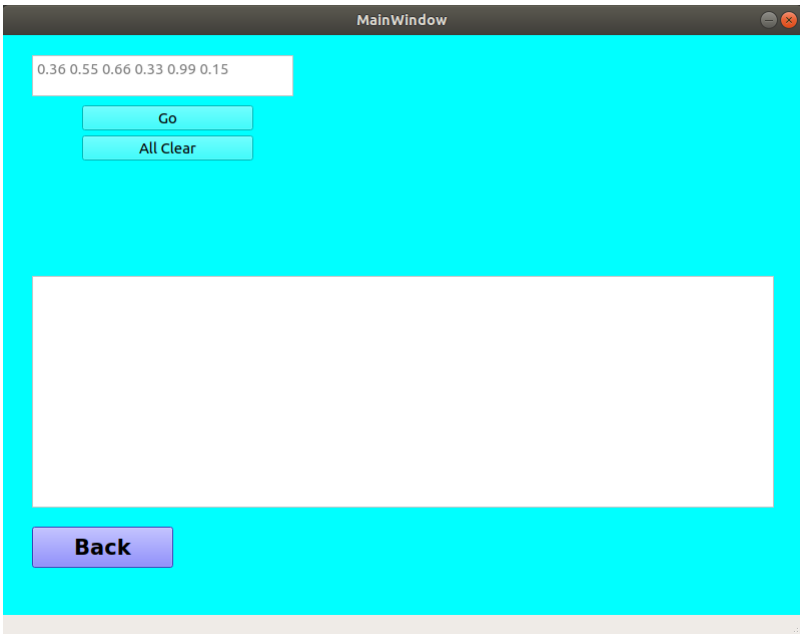

Figure 3.12: Autonomous Mode Interface

3.3.10 Integration of RFID

The robot takes decisions about the about the next goal it goes to, stops or change its speed based on the RFID tags the RFID reader reads and the source antenna it reads it from. The process of choosing which destination the robot should go or whether it should stop is handled by the handler.py node in the r_gui_h package. The handler node is subscribed to the /readerNodeMessages topic and will send goals to the move_base node or cancel them based on the messages it receives from the reader node. The GUI node is also subscribed to the /readerNodeMessages and will display information about the object to which the received tag is attached to on the gui's consoles. The tag id is interpreted as shown in the following figure:

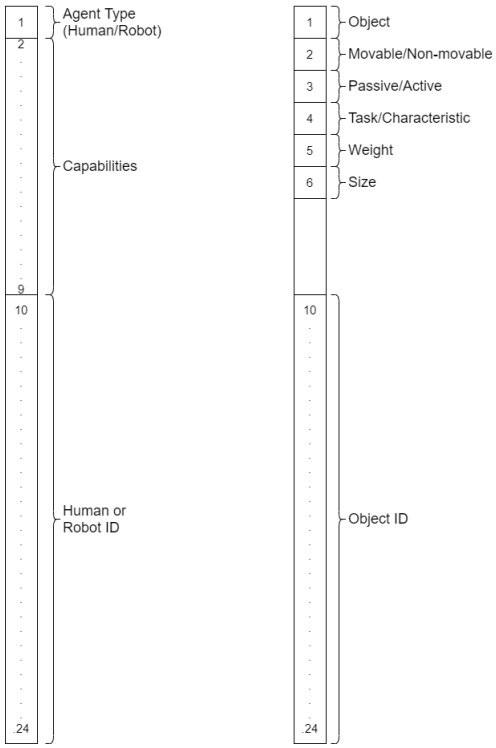

Figure 3.13: Tag ID Interpretation By ROS Nodes

Agent type: Which can be either a human or a robot.
Object: Indicates that it's not a human or a robot.
Capabilities: A human or robot agent can possess one or more capabilities such as mobility, manipulation, welding, assembly ... etc.
Tasks/Characteristics: A simple object has a task, which is the task that it is used for, or a characteristic whether it is a wall, floor ... etc.
Movable or non-movable: The hexadecimal digit assigned to it determines whether the object is movable or non-movable.
Active: Movable and active objects are tools that change the attributes of any object. Non-movable and active objects are machines that change the attributes of objects.
Passive: Movable and passive objects are the ones to be manufactured and transported. The non-movable and passive objects are workspaces, walls, doors ... etc.
Notice that the first hexadecimal digit in the tag id isn't shown in the figure as it is already used in the handler node to indicate stopping or restarting motion for the robot.

3.4 Raspberry pi Solution

3.4.1 Low Level Design

In the design process, we have used the following robot chassis, it represents a standards, low cost Differential Drive Mobile robot.

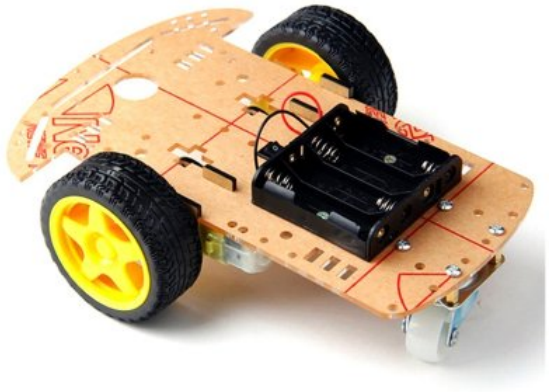

Figure 3.14: Differential Drive Mobile Robot

For the Electronics Design we designed our own circuit to have control over DC motors and get state feedback about the current position of the robot, since we are doing closed loop control.

3.4.1.1 Arduino Microcontroller

The microcontroller we used for our robot platform is the Arduino, which is an open-source Electronics Board based on Atmega microcontroller. Arduino boards are able to deal with input output using the Arduino programming language which is based on C/C++.

3.4.1.2 RaspberryPi Single-Board Computer

Since we are going to use Robot Operating System, our embedded system must include a computer running Linux operating system (Ubuntu Distribution) The Raspberry Pi is a small Single-Board Computer developed by The Raspberry Pi Foundation, that runs the Linux Operating System , its main distribution is called Raspbian. But in our case we have installed Ubuntu Mat since ROS works better with ubuntu distributions.

3.4.1.3 Controlling DC motors

The Speed and The direction of the DC motors were controlled using the L298N dual motor Driver. The Direction can be controlled using voltage levels High or Low, whereas the speed is controlled using PWM (Pulse Width Modulated) signals. The following schematic shows the Motor Driver pinout:

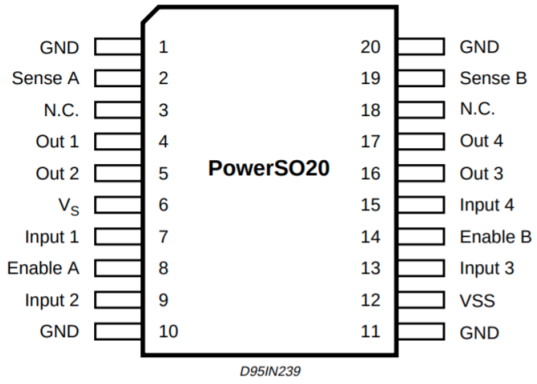

Figure 3.15: Motor Driver pinout [Sparkfun datasheet]

The Output pins are linked to the right and left motor, the input pins (2 for each motor) are used for direction control, whereas the enable pins are used for speed control by receiving PWM signals from the Arduino Microcontroller. The operation of the motor driver is according to the following logic diagram:

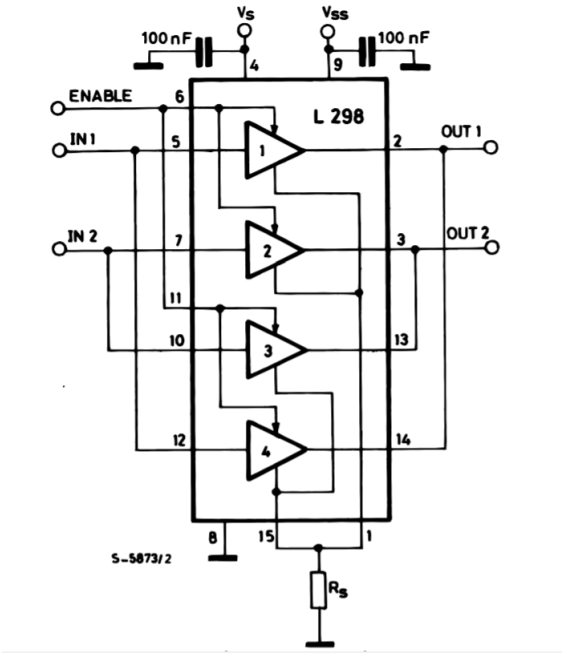

Figure 3.16: Logic diagram of L298N motor driver [Sparkfun datasheet]

3.4.1.4 Getting Feedback data from Encoders

In order to know the current position of the robot we need a sensor, in our case we have used 2 optical encoders for each motor, then we have used the ticks of the Encoders on The differential Drive model. The signals generated by the optical encoders is illustrated in the following figure:

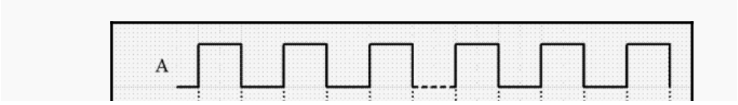

Figure 3.17: Optical Encoder Signal

The encoder we used is only a one channel encoder, hence it can't give information about the direction of the rotation, we overcome this problem by software, in the arduino code we have set a flag that stores the direction of the motor according to the command given.

3.4.2 Linking ROS to the low-level controller

Using one sensor on the arduino and sending its data via serial port to the computer(in our case the raspberryPi) is a very easy task, especially using pyserial python library and the built-in Serial class in the arduino, However when we have more than one sensor, and the data needs to be sent and received at the same time the task becomes difficult to handle. First, we have used a library called the messenger handler library in the arduino to give the serial data a good structure, i.e the data needs to be structured in one string and then be sent to ROS , where it gets decoded and sent through different ROS topics, the same applies for the data sent from ROS. Another problem

we have faced is that the serial communication protocol can't send and receive data at the same time. If we try to receive data from ROS while the port is busy, we get an error and the port will be disconnected. Hence to overcome this problem we had to use the Multithreading concept to be able to send and receive simultaneously.

The whole process of linking the low-level controller (Arduino) with the raspberry Pi, including decoding the data, sending, receiving and publishing was implemented into the communication node. The node was initially developed by Lentin Joseph it was for communication using bluetooth[7]. But we have modified it to use serial communication via USB, by the help of the threading and pyserial python libraries.

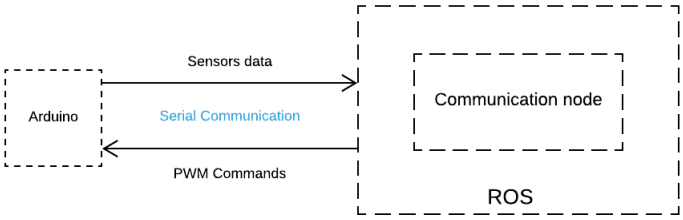

Figure 3.18: Linking Arduino and ROS

3.4.3 Publishing Sensor Data and Speed Commands through ROS topics

The robot_bt_driver node is not responsible only for linking arduino to ROS, but also for publishing sensor data and speed commands through ROS topics. The string sent to ros from arduino takes the following structure: 'name1 x y name2' For example x and y can be replaced with left motor PWM, right motor PWM, or for encoder data, like left motor ticks, right motor ticks.

3.4.4 My_Pos_Calc Node

This node calculates the pose of the robot using the encoders data. It subscribes to right_ticks and left_ticks topics, calculates the pose of the robot using the equations in section 2.6.1 and publishes it to a topic called Pose2D.

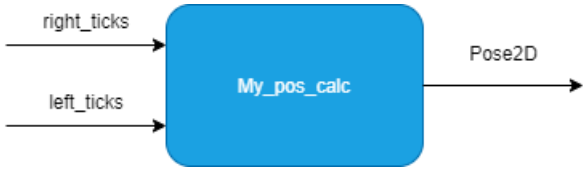

Figure 3.19: My_Pos_Calc Diagram

3.4.5 My_uni_to_diff node

When controlling the robot manually or using move base, the messages published are Twist messages, which contain the desired linear and angular velocities. However, what we need to send to Arduino is the desired speed for the right and left motors, and this where this node comes in. it subscribes to the kbotcmd_vel topic, receives the twist messages and converts the linear and angular velocities to left motor speed and right motor speed using the equation (2.2). Finally, it publishes them to lwheel_vtarget and rwheel_vtarget topics respectively.

Figure 3.20: My_Uni_To_Diff Diagram

3.4.6 Combine motors node

This node simply subscribes to rwheel_vtarget and lwheel_vtarget topics, combines the desired right motor and left motor speeds that it received from these topics in one string and publishes it to the set_speed topic.

Figure 3.21: CombineMotors Diagram

3.5 Conclusion

After explaining our system in detail in this chapter, we describe the results obtained from testing our system in a HITL simulation and on a real differential drive robot. We will discuss the results and provide suggestions to enhance system in the next chapter.

Chapter 4

validation and results

4.1 Introduction

In this chapter we will talk about the validation of our system, both HITL and the Hardware implementation. We will discuss the results, problems that we faced and give suggestions in how to make the system better. We will start by the setup of our system where we mention the nodes and launch files that we start as well as the tools launch. Then, we will talk about the communication with the RFID reader. After that we will talk about the HITL simulation and the scenarios that we realized as demonstration of our work. Finally we will talk about the hardware implementation and how far we got in it.

4.2 System Setup

We start by launching the necessary nodes and tools in our work, we will go through them one by one while providing the commands used to launch them and the results obtained from them. Be aware that we source devel/setup.sh in every new terminal we open which sets environment variables used by ROS and by the Gazebo simulator.
First we launch our kbot_base_rviz_gazebo.launch file using the following command:

```
roslaunch kbot_description kbot_base_rviz_gazebo.launch
```

This will start rviz and gazebo and load the environment we built and our kbot robot model into the gazebo world.

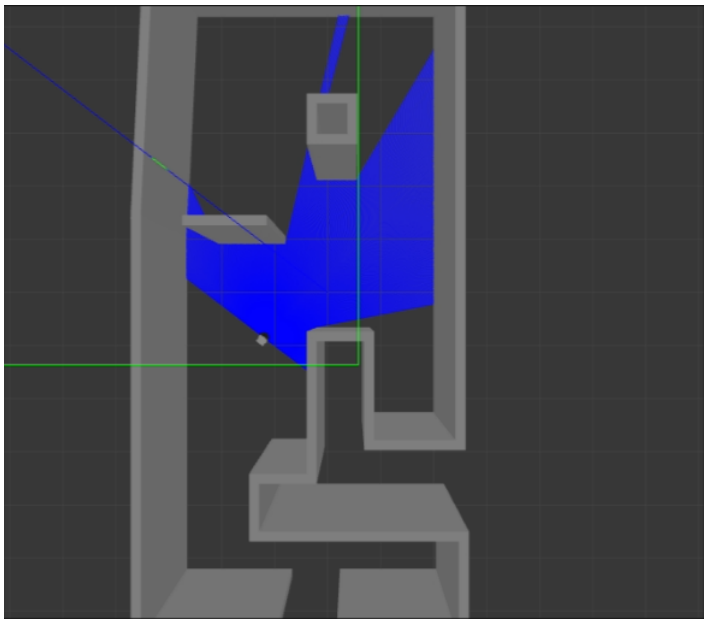

Figure 4.1: Virtual Environment with Gazebo

Next we launch the move_base.launch file using the following command:

```
roslaunch kbot_simple_control move_base.launch
```

This will launch the move_base node and load the configuration files that it need. We have added our handler and speed_regulator nodes to this launch files so that we can launch them using a single command instead of running them one at a time. We then load our saved rviz configuration which lets us visualize the robot model, the map, the laser scan and the pose array.

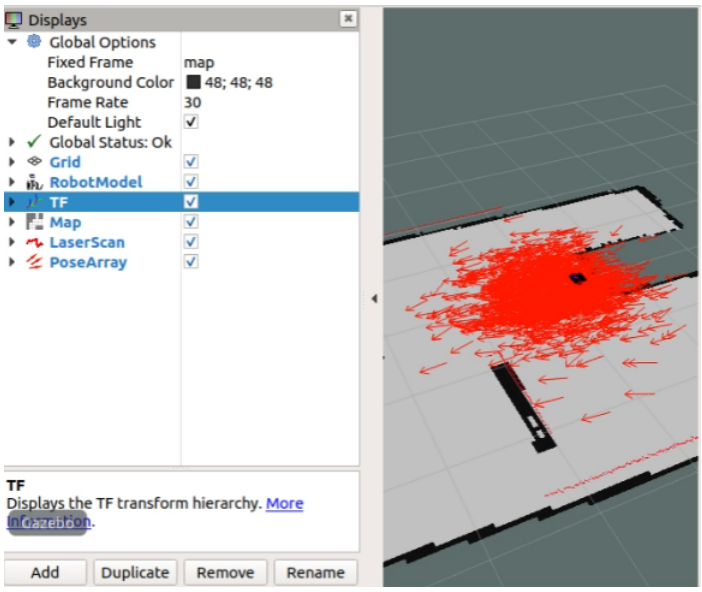

Figure 4.2: RVIZ With Loaded Configuration

After that, we run the readerNode using the following command:

```
rosrun r_gui_h readerNode.py
```

Then we launch the GUI using the following command:

```
rosrun r_gui_h GUI.py
```

4.3 Communication With The RFID Reader

We link with the RFID reader in order to be able to send commands to it, and we do that by clicking on the link with the reader button on the reader interface in the GUI. Usually, the first time we try to link with the reader, after that, it links successfully.
Now, we need to set the reader so that it sends the RFID tags information once it reads them in real time, and we do that using the Continuous trigger button in the Tags menu. If we want to stop this behavior, we can simply click on the Stop trigger button in the Tags menu.
Refer to Appendix B to see the XML commands and replies sent to and from the reader.
We wrote the tag IDs that we need in several RFID tags using the WBM(see Appendix C).

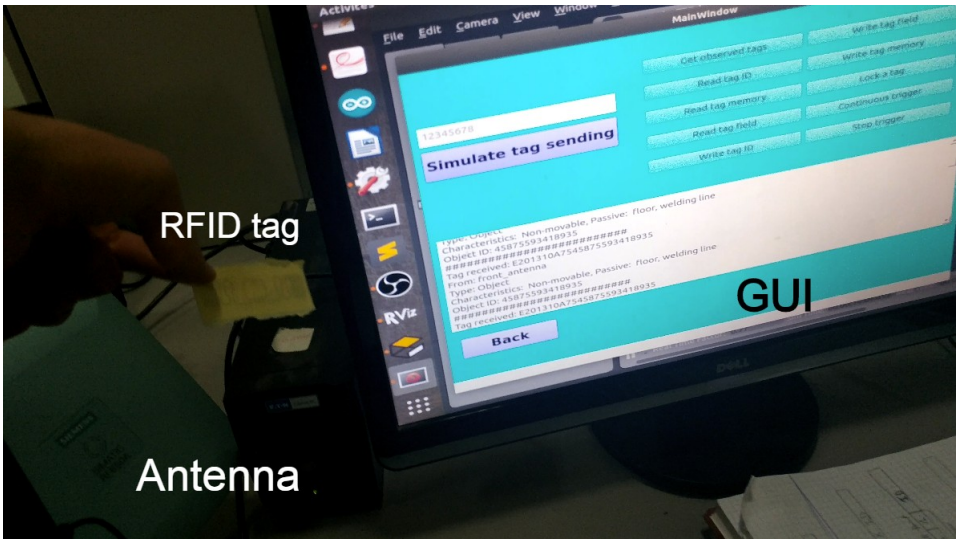

Figure 4.3: Antenna reading RFID tag and its information displayed on the GUI

4.4 HITL Validation

Now that everything is set, we can test our system with simple scenarios where our robot goes to a goal, stops or alter its speed based on the tags it reads. First, we move the robot in the environment manually a little bit so that the amcl algorithm can have a better localization of the robot.

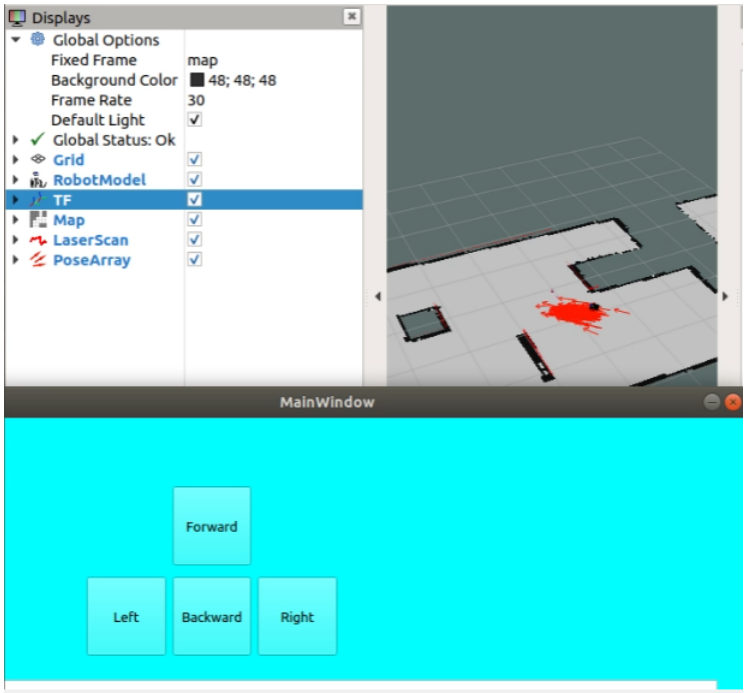

Figure 4.4: Visulization in RVIZ after moving the robot around manually

Next, we pass an RFID tag of a small, light, movable object that requires polishing. Our system behaves as expected; the interpretation of the RFID tag is displayed on the GUI, and the robot moves to the intended location on the gazebo environment.

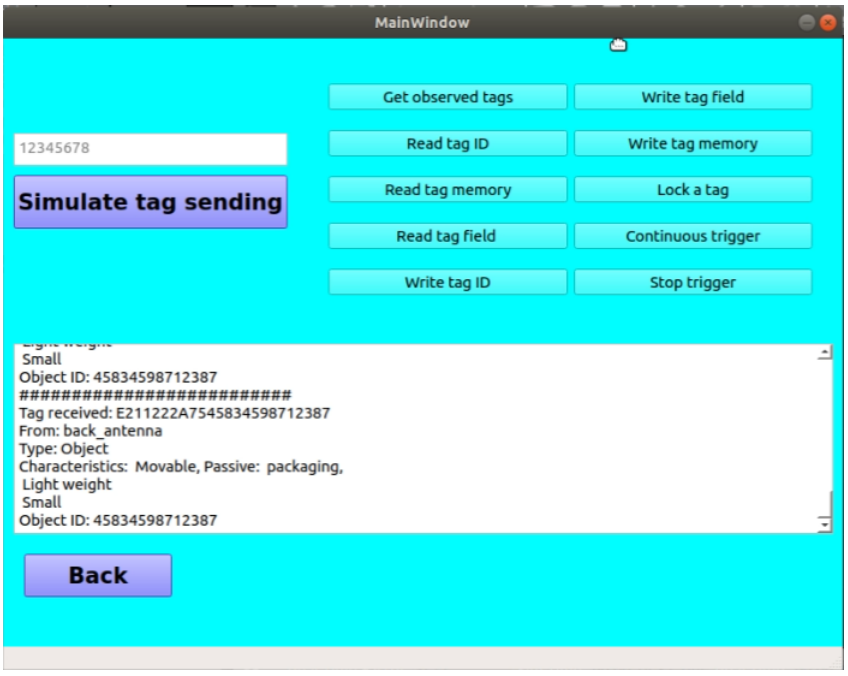

Figure 4.5: Message on the GUI after reading the RFID tag of an object

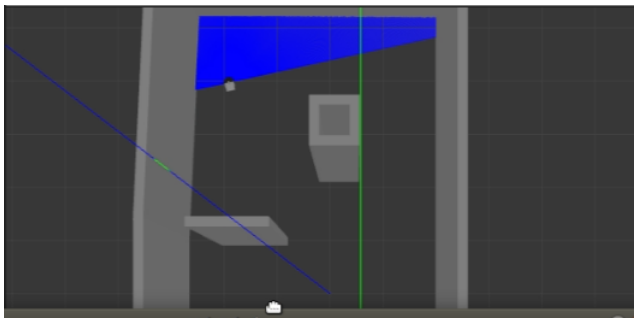

Figure 4.6: Robot Reaching Its Goal in Gazebo

As the robot moves toward its goal, we pass an RFID tag of a wall in the packaging line. The robot slows down as expected. Once it detects an object from other than the packaging line, it resumes motion with normal speed.

Passing an RFID tag of a small, light, movable object in the range of the front, left or right antenna makes the robot stop and send a message to the human operator to move the obstacle as shown in the following figure:

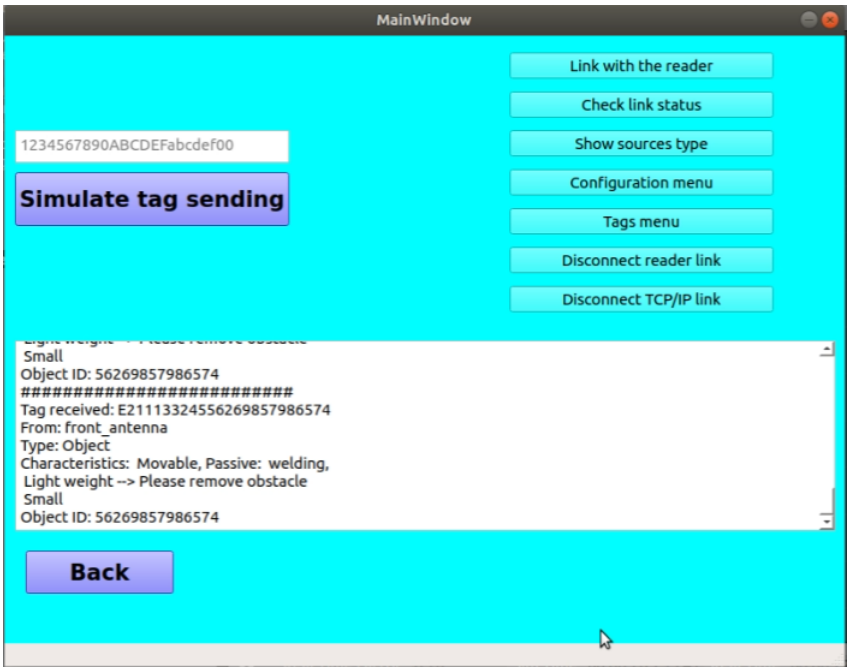

Figure 4.7: GUI message after detecting a movable obstacle

Once the obstacle is cleared, we can press the "All Clear" button in the autonomous interface for the robot to resume motion toward its goal.

4.5 Hardware Validation

As for the real robot, we managed to move it using Twist messages generated by our manual mode interface, meaning that the my_uni_to_diff , combineMotors and robot_bt_driver when sending data to the arduino worked fine. The robot_bt_driver also received the encoders data, however, the data was very noisy, hence, the system couldn't calculate the robot's position using the encoders data and we couldn't use move_base to control the robot.

4.6 Suggestions For Improvement

In order to obtain greater performance for our system and achieve better results, some improvement can be made.
Adding a better 3D model of the robot using mesh files can give a better look for the robot and the simulation overall. Adjusting the inertia to more realistic values can make the simulation more realistic.

As for the hardware part, our main problem was in determining the position of the robot due to the bad signal from the encoders. Using better encoders and using noise filters can help solve this problem, and make our real robot perform the same as in the simulation. Unfortunatly, the implementation of the raspberry solution on the real robot RobuTER was not accomplished due to the COVID-19 pandemic.

4.7 Conclusion

In this chapter we demonstrated the results of simulating our system in a gazebo environment and on a real robot. Next, we will provide an overall conclusion of our work.

Conclusion and perspectives

Our purpose in this project was to make an autonomous differential drive robot that takes decisions about its motion based on the RFID tags that it reads in its environment. The robot is expected to communicate with the RFID reader successfully, interpret the RFID tags that it receives and then stops, alter its speed or find the a path to its goal and go to it autonomously.
First, we talked about robotics and did literature review of some papers in the topics of RFID, raspberry pi and robotics. We then presented the overall architecture of our system and talked about the control of a differential drive robot. Next, we talked about our system in detail and how far we have got implementing it.

We worked on implementing our system in a HITL simulation and on a real small differential drive robot with raspberry pi. In the HITL simulation, we began by creating a simple differential drive robot model with one caster wheel and a laser scanner. We also made a simple closed room environment for the robot to navigate in. The robot is controlled by the messages sent from the ROS navigation stack, which we tuned to work properly with our robot model, or manually using a GUI we developed to simplify the communication with the robot and the RFID reader. As for the raspberry pi implementation, we introduced our low level control, the Arduino code and the additional ROS nodes needed to communicate between the raspberry pi and the Arduino and interpret thier messages. These two implementation could potentially run together in order to control a real robot and view the expected behavior on the HITL simulation.
As for the RFID part, we used the SIEMENS SIMATIC RF680R RFID reader that had four SIEMENS SIMATIC 650A antennas attached to it that read UHF RFID tags which content we wrote using the WBM.

To test our system, we set some objectives in the HITL simulation which were:
- Establish connection with the RFID reader and receive data from it.
- Use the GUI to communicate with the RFID reader and control the robot.
- The robot should be able to go to a position in the map autonomously.
- Recognize the objects and obstacles in its environment based on their RFID tags.
- Alter its motion and send messages to the human operator according to the RFID tags it reads.

The results in the HITL simulation were satisfactory. As for implementing our work on a real robot, then connecting a raspberry pi running ROS with an Arduino and controlling the robot manually using our GUI was as far as we went.

Working on this projects showed us how powerful ROS is in robotics and that it deserved its popularity within the robotics community. It also showed us how useful the RFID is in the industry and the great potential it has in it.

Appendix A

Intoduction

In this appendix we will explain some of the configuration files that are necessary for our move_base node to work properly.

costmap_common_params

```
obstacle_range: 2.5
raytrace_range: 3.0
footprint: [[-0.08, -0.08], [-0.08, 0.08], [0.08, 0.08], [0.08, -0.08]]
#robot_radius: ir_of_robot
inflation_radius: 0.20

observation_sources: laser_scan_sensor

laser_scan_sensor: {sensor_frame: sensor_laser, data_type: LaserScan,
                    topic: /scan, marking: true, clearing: true}
```

Here we set either the footprint of the robot or the radius of the robot if it is circular. In our case, we specified the footprint, the center of the robot is assumed to be at (0.0, 0.0) and both clockwise and counterclockwise specifications are supported. We declare the points that define its boundaries in meters. We'll also set the inflation radius for the costmap. The inflation radius should be set to the maximum distance from obstacles at which a cost should be incurred. In our case, setting the inflation radius at 0.20 meters means that the robot will treat all paths that stay 0.20 meters or more away from obstacles as having equal obstacle cost.

global_costmap_params

```
global_costmap:
  global_frame: map
  robot_base_frame: base_footprint
  update_frequency: 3.0
  static_map: true
```

The "global_frame" parameter defines what coordinate frame the costmap should run in, in this case, we'll choose the /map frame. The "robot_base_frame" parameter defines the coordinate frame the costmap should reference for the base of the robot. The "update_frequency" parameter determines the frequency, in Hz, at which the costmap will run its update loop. The "static_map"

parameter determines whether or not the costmap should initialize itself based on a map served by the map_server.

local_costmap_params

```
local_costmap:
  global_frame: odom
  robot_base_frame: base_link
  update_frequency: 5.0
  publish_frequency: 2.0
  static_map: false
  rolling_window: true
  width: 6.0
  height: 6.0
  resolution: 0.05
```

The "global‘_frame", "robot_base_frame", "update_frequency", and "static_map" parameters are the same as described in the Global Configuration section above. The "publish_frequency" parameter determines the rate, in Hz, at which the costmap will publish visualization information. Setting the "rolling_window" parameter to true means that the costmap will remain centered around the robot as the robot moves through the world. The "width," "height," and "resolution" parameters set the width (meters), height (meters), and resolution (meters/cell) of the costmap.

base_local_planner_params

```
TrajectoryPlannerROS:
  max_vel_x: 0.45
  min_vel_x: 0.1
  max_vel_theta: 1.0
  min_in_place_vel_theta: 0.4

  acc_lim_theta: 3.2
  acc_lim_x: 2.5
  acc_lim_y: 2.5

  holonomic_robot: true
```

The base_local_planner is responsible for computing velocity commands to send to the mobile base of the robot given a high-level plan. The first section of parameters above define the velocity limits of the robot. The second section defines the acceleration limits of the robot.

Appendix B

Introduction

Link With The Reader

Link Command

```
<frame>
<cmd>
 <id> value_id </id>
 <hostGreetings>
 <readerType> value readerType </readerType> //opt
 <supportedVersions>
 <version> value_version </version>
 <version> value_version </version> // opt
 ...
 </supportedVersions>
 </hostGreetings>
</cmd>
</frame>
```

Link Expected Reply

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

Continuous Trigger

Continuous Trigger Command

```
<frame>
<cmd>
 <id> value_id </id>
 <triggerSource>
 <sourceName> value_sourceName </sourceName>
 <triggerMode> value_triggerMode </triggerMode> // opt
 </triggerSource>
</cmd>
</frame>
```

Setting value_triggerMode to Start triggers the Reader continuously. Setting it to Stop stops the trigger.

Continuous Trigger Expected Reply

```
<frame>
<reply>
 <id> value_id </id>
 <resultCode> 0 </resultCode>
 <triggerSource/>
</reply>
</frame>
```

A result code of 0 means that there is no error. A result code other than 0 refers to an error number that can be found in the manual.

Appendix C

Introduction

In this appendix we will describe some of the tools we used in our work.

Ubuntu Mate

Ubuntu MATE is a stable, easy-to-use operating system with a configurable desktop environment. It is ideal for those who want the most out of their computers and prefer a traditional desktop metaphor. With modest hardware requirements it is suitable for modern workstations, single board computers and older hardware alike. Ubuntu MATE makes modern computers fast and old computers usable.

Figure 8: Ubuntu Mate

Gazebo

Gazebo is an open-source 3D robotics simulator. Gazebo was a component in the Player Project from 2004 through 2011. Gazebo integrated the ODE physics engine, OpenGL rendering, and support code for sensor simulation and actuator control. In 2011, Gazebo became an independent project supported by Willow Garage. In 2012, Open Source Robotics Foundation (OSRF) became the steward of the Gazebo project. OSRF changed its name to Open Robotics in 2018.

Gazebo can use multiple high-performance physics engines, such as ODE, Bullet, etc (the default is ODE). It provides realistic rendering of environments including high-quality lighting, shadows, and textures. It can model sensors that "see" the simulated environment, such as laser range finders, cameras (including wide-angle), Kinect style sensors, etc.

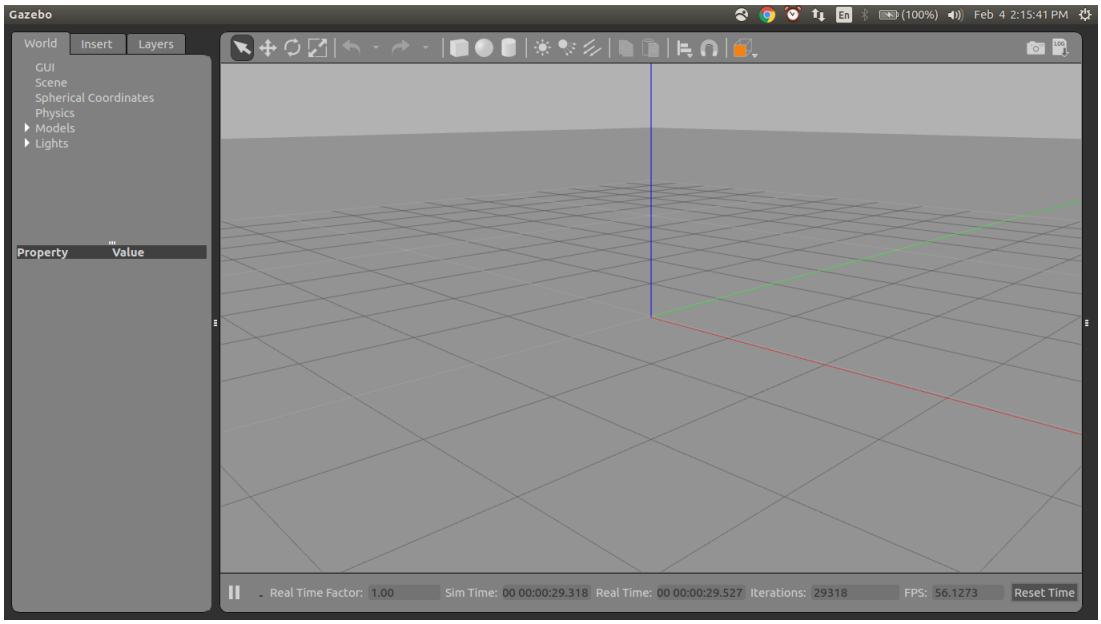

Figure 9: Gazebo

PyQt5

PyQt5 is a comprehensive set of Python bindings for Qt v5. It is implemented as more than 35 extension modules and enables Python to be used as an alternative application development language to C++ on all supported platforms including iOS and Android.
To easily design the interface, Qt designer may be used.

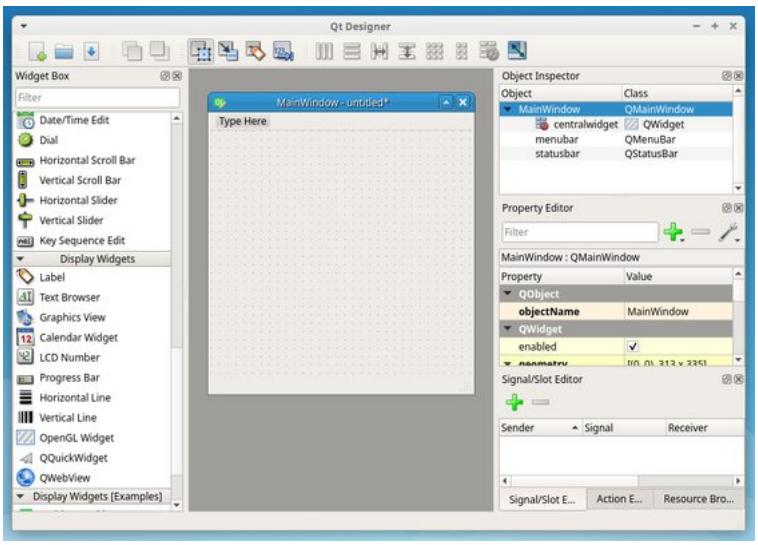

Figure 10: Qt Designer

WBM (Web Based Management)

The RFID reader can be configured using the WBM, which is an interface accessible using the web browser by entering the reader's IP address in the address bar.

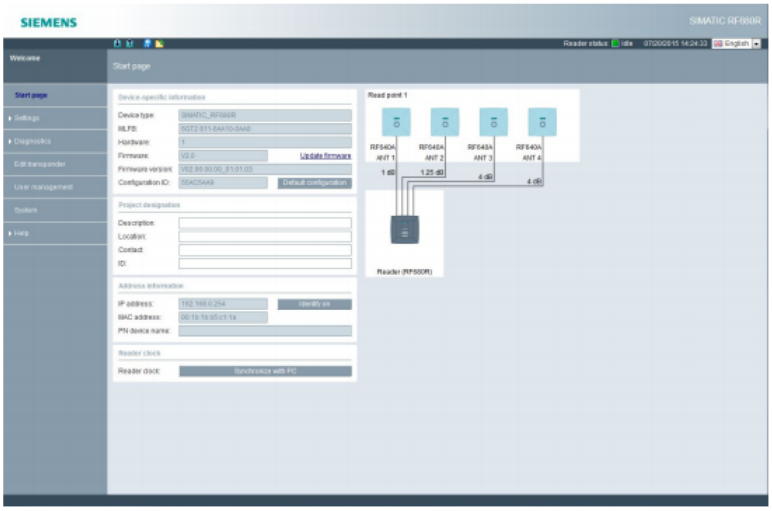

Figure 11: Start page of WBM

Bibliography

[1] URL: https://www.ni.com/en-lb/innovations/white-papers/17/what-is-hardware-in-the-loop-.html#section-1003763995.

[2] URL: http://wiki.ros.org/.

[3] ADNAN ADEMOVIC. *An Introduction to Robot Operating System: The Ultimate Robot Application Framework*. URL: https://www.toptal.com/robotics/introduction-to-robot-operating-system.

[4] L. Baranowski, J. Panasiuk, and M. Siwek. "USE OF A RASPBERRY PI TO BUILT A PROTOTYPE WIRELESS CONTROL SYSTEM OF A MOBILE ROBOT". In: (2017).

[5] Byoungwook Choi et al. "Integration of ROS and real-time tasks using message pipe mechanism on Xenomai for a telepresence robot". In: (2018).

[6] Grazia Cicirelli, Annalisa Milella, and Donato Di Paola. "RFID Sensor Modeling by Using an Autonomous Mobile Robot". In: (2011).

[7] Lentin Joseph. *Robot Operating System for Absolute Beginners Robotics Programming Made Easy*.

[8] K.D.V.S.Anil kumar and M.suman. "An Autonomous Robot Navigation". In: *International Journal of Computer Trends and Technology (IJCTT)* (2013).

[9] Annalisa Milella, Donato Di Paola, and Grazia Cicirelli. "RFID Technology for Mobile Robot Surveillance". In: (2010).

[10] Mhamed Nour and Ahmed Lakhssassi. "Automatic Laser Interstitial Thermal Therapy for Robot-Assisted Surgery: Implementation". In: (2018).

[11] Sayyed Jaffar Ali Raza et al. "Real-World Modeling of a Pathfinding Robot Using Robot Operating System (ROS)". In: (2018).

[12] Se-gon Roh and Hyouk Ryeol Choi. "Object Recognition Using a 3D RFID System". In: (2005).

[13] Margaret Rouse. *Mobile Robotics*. URL: https://internetofthingsagenda.techtarget.com/definition/mobile-robot-mobile-robotics.

[14] *SIEMENS SIMATIC Ident RFID systems SIMATIC RF650R/RF680R/RF685R Configuration Manual*.

[15] Santhosh Krishna B v et al. "Cloud robotics in industry using Raspberry Pi". In: (2016).

[16] A.Seshanka Venkatesh et al. "Robot Navigation System with RFID and Ultrasonic Sensors". In: (x).