Bachelor Degree Project

UNIVERSITY
OF SKÖVDE

Autonomous Robotics Platforms

Bachelor Degree Project in Manufacturing Engineering
Bachelor Level 30 ECTS
Spring term 2019

Néstor Morales de la Fuente
Manuel Serrano Durán

Supervisor: Bernard Schmidt
Examiner: Richard Senington

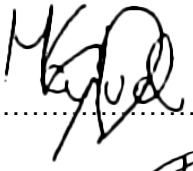
Certify of Authenticity

Submitted by Néstor Morales de la Fuente and Manuel Serrano Durán to the University of Skövde as a Bachelor degree thesis at the School of Technology and Society. I certify that all material in this thesis project which is not my own work has been identified using Vancouver as the reference method.

Skovde, June 19th 2019

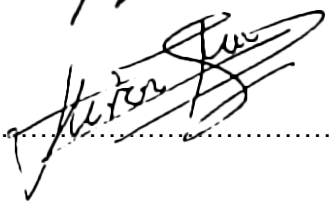
……

Place and date



……

Signature



……

Signature

Abstract

Nowadays, it is of crucial importance for the manufacturing industry to be prepared for the application and understanding of autonomous mobile robots. Given this fact, educational institutions have to provide knowledge and experience to students. These autonomous mobile robots are made and controlled using different platforms and programming languages.

Currently, the University of Skövde wishes to expand the range of tools to be available for building and experimenting with autonomous robots. For this purpose, this thesis project has two main goals.

The first goal is finding the best available robotic platform for small scale, self-built, programmable robots. Such a platform has to include all the hardware necessary for later implementation with the software. The platform is evaluated in this thesis following different criteria.

The second goal is to build a robot using the chosen platform. The robot has to perform a certain task taking advantage of its specific hardware. The development of the task has been achieved using the Robot Operative System (ROS).

This thesis provides step by step instructions of how to build the platform and perform the task.

Table of Contents

List of Figures

List of Abbreviations

ROS	Robot Operating System
IFR	International Federation of Robotics
AI	Artificial Intelligence
AGV	Automated Guided Vehicle
CCD	Charged-Coupled Device
CMOS	Complementary Metal-Oxide Semiconductor
GPIO	General Purpose Input Output
SBC	Single Board Computer
SRAM	Static Random Access Memory
OpenCV	Open Source Computer Vision
HSV	Hue Saturation Value
URDF	Universal Robotic Description Format
XML	Extensible Markup Language

1. Introduction

1.1. Brief History of Robotics

The meaning of the word “Robot”, as it is currently known, has been developed in just under the past 100 years. The first use of the term robot was introduced in 1921 in a science fiction play called “Rossum's Universal Robots” by the Czech writer Karel Capek [1]. In Karel’s play, an inventor creates artificial workers made of flesh that could replace humans at any job, leading into a rebellion by the oppressed workers. Following this play, a number of important novels and films also included the concept of robot-like Asimov's famous “I, Robot” [2]. However, this time the term robot implied a human-like creature made of mechanical parts instead of flesh, differing from Karel’s play.

There is a difference between these terms used in fiction and reality. It does not matter what appearance the robot has as long as it gets the job done. In a factory, it is not important if a robot arm resembles a human one. In this example, the main goal for the robot would be to move boxes from one conveyor to another, regardless of what shape it has.

The robot definition provided by Oxford is “a machine that can perform a complicated series of tasks automatically” [3]. There is a big difference between a robot that sweeps the floor and one exploring Mars. What both have in common is that they can do their jobs autonomously, with little or no human interaction. However, nowadays the field of Collaborative Robots is becoming popular. These robots are designed to interact with humans in the same environment, in opposition with those designed to develop their task autonomously or with guidance.

1.2. History of Robotics in the Industry

After the Second Global War, in America, there was a great industrial push, which led to a huge development of its economy, in part helped by the rapid advancement in electronic technology - servos, digital logic, solid state electronics, etc. In this context, the first industry robot arose thanks to the chance meeting between George Devol and Joseph Engelberger in 1956.

George C. Devol, Jr (February 20, 1912 — August 11, 2011) patented in 1946 a device used for controlling machines, which led him towards the automation field and patent another invention in 1954. This new appliance called unimation, as he described it “makes available for the first time a more or less general purpose machine that has universal application to a vast diversity of applications where cyclic control is desired.” [4]

Joseph F. Engelberger (July 26, 1925 — December 1, 2015) born in New York, grew with an early fascination for the science fiction genre, especially inspired by Isaac Asimov. This enthusiasm made him choose physics at Columbia University, and later he served in the U.S. Navy and worked as a nuclear physicist in the aerospace industry.

Devol and Engelberger met at a party in 1956 and they talked about robotics, automation, Asimov, and Devol's last patent, which Engelberger translated into "robot".
After the meeting, Devol and Engelberger formed an association that led to the birth of the first industrial robot. The team of engineers that previously worked for Devol targeted their efforts on the development of Unimate. The first one was made in 1961 and was delivered to General Motors in Trenton, New Jersey, where it was in charge of unloading high-temperature components from a die casting machine — a very unpopular job for manual labour. It took until 1975 to make a profit, but the company then became the world leader in robotics, with 1983 annual sales of $70 million and 25 per cent of the world market share [5].

Since then, the uprising of robotics is almost exponential. As a graphical example of this incredible growth, the chart in Figure 1 represents the annual supply of industrial robots worldwide between 2008 and 2016 and their forecast between 2017 and 2020, according to the International Federation of Robotics (IFR) [6]. In addition, in Figure 2 can be found the revenue of the robotics field in 2017 and the expected in 2025 in billions of U.S. dollars [7].

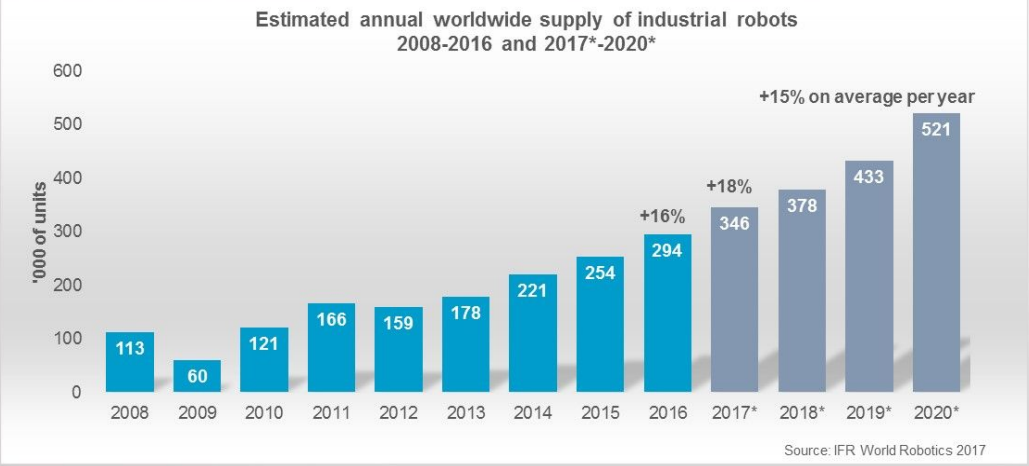


Figure 1: Estimated annual worldwide supply of industrial robots [6]

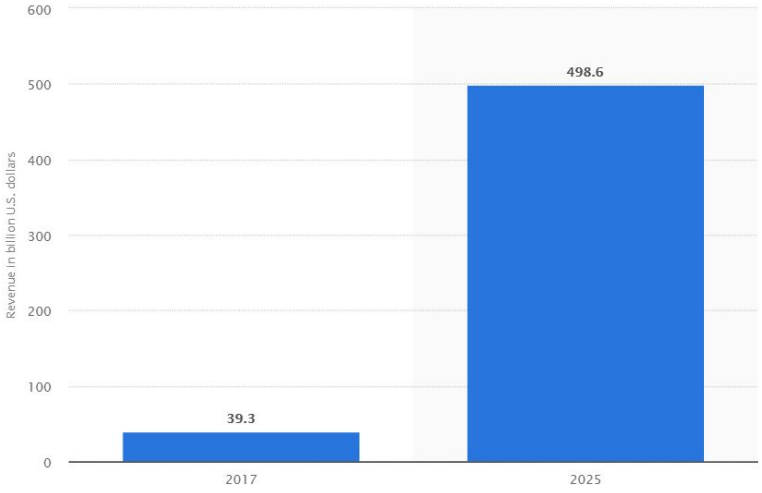


Figure 2: Revenue in the robotics field in 2017 and estimated in 2025 [7]

1.3. Background

Autonomous robots are already being used in industry. In the near future, it is expected that their implementation will become even more advanced. These autonomous robots have different levels of autonomy, sensors, actuators etc. Industry demands skilled engineers that know how to operate and work with autonomous robots.

Currently, students are not being taught what they need to know about autonomous robots. There is a tremendous need to implement this knowledge that will allow students to be competent in the future. This knowledge does not necessarily mean that students should know how to build an autonomous robot from scratch but to work with them and understand how they work.

1.4. Problem description

The University of Skövde is looking forward to having autonomous robots for the students to learn how to design, build and program.

In this thesis, the aim is to develop a small autonomous robot that performs a task. The development of the project has to be documented for possible future implementation in the University.

It is imperative to choose between all the accessible options that can be found on the Internet and to do so the different platforms must be analyzed.

Nowadays there is barely any limitation on what is available to purchase. There is a vast community behind robotics development with different software and programs already implemented.

Another field to take into account is what platform should be used to program the robot. Currently, there is a wide range of middleware to program different devices, but in the autonomous robots field the most famous one is Robot Operating System (ROS).

Finally, the main obstacle in the project is to learn ROS, a completely new middleware for the authors running in Linux. Once this is achieved, the next step is to use it to control the robot and perform the task described. Also, a simulation of the robot and the environment has to be designed in an extension of ROS named Gazebo.

1.5. Aims and objectives

The aim of this thesis is to select the most suitable educational robotic platform and control it using a middleware.

To reach the aforesaid aim, the objectives of the thesis are:

- Selection of the best robotic platform and kit to fulfil the requirements described.
- Build the robot kit chosen and carry out a certain task.
- Develop an online simulation in which the code can be tested.

1.6. Extent and delimitations

The first assignment to do will be the choice of the platform. To do so, different criteria will be analysed in order to select the most suitable one. Also, after the choice of the platform, the selection of the robot kit will be carried out following the same method. The next step is to program the robot to perform a task. It will be designed to take advantage of the DC motors and camera, mainly. This application will be sufficient to conclude the feasibility of the system and framework. Lastly, a simulation will be built to test the code starting with simple functionalities and following with more complex ones.

1.7. Sustainability

Sustainability is often defined as the quality of causing less damage possible to the environment, so the activity can be continued for a longer period of time. It is divided, according to the World Summit on Social Development on 2005 [8], into three fundamental pillars: Economic, Social and Environment development.

In an economic perspective, this project is developed under the minimum budget possible. Its economic impact is minimal. ROS, being an open source middleware, is completely free to install and use. Additionally, the robot can be reprogrammed as many times as possible, giving great flexibility for future students to modify it at their liking without needing a new robot. Highly skilled students translate into a more efficient economy.

Regarding social sustainability, this project was developed for educational purposes. It is meant to be used as support to create a new course in autonomous mobile robots. Thereafter it contributes to healthy social development.

From an environmental standpoint, this project has a small ecological impact. It has been developed to be implemented at the University of Skövde. Each material is meant to be used in different courses, thus having a long lifespan and diminishing its ecological impact. Furthermore, always that is feasible, eco-friendly materials are used in the robot. For instance, the robot uses rechargeable batteries as its main power source.

1.8. Thesis structure

This thesis is going to be divided into 9 chapters, added to an appendix and the bibliography. The first Chapter presents the project and its main goals. In Chapter 2 is developed the research done for this project, the literature review. After that, in Chapter 3, the methodology followed is described. The selection of the hardware, the robotic platform and the robot kit, will be discussed in Chapter 4. Chapter 5 consists of a presentation of the middleware used, ROS. The implementation of the robot tasks and its programming will be

performed in Chapter 6. Chapter 7 presents Gazebo and an example simulation in order to ease the learning. Lastly, the discussion is provided in Chapter 8 and the conclusions in Chapter 9.

2. Literature Review

2.1. Mobile Robots in Industry

Nowadays, most of the industrial robots purchased (especially in the automation field) are robotic arms or manipulators. These kinds of robots have reduced mobility since they are screwed to a fixed point in the production line, where they move with great speed and precision. Their characteristics make them perfect for developing repetitive tasks such as welding, assembly or painting.

However, despite being so popular, robotic arms have a disadvantage: the lack of mobility, since it depends on where the base is screwed, unless that base is mobile. There are other types of robots like collaborative robots and mobile robots among others. Mobile robots can change their location to achieve their goals. This type of robot also has its drawbacks: lack of stability and strength. The selection of different robots in the industry varies depending on the needs of the process that wants to be automated.

The most popular mobile robot in the industry is the Automated Guided Vehicle shown in Figure 2 [9].

Figure 3: Automated Guided Vehicle in a factory [9]

AGVs tend to work in a modified environment. This means that there is some kind of system leading its way, such as lines in the floor, induction loops, beacons, etc. Without this system the robot would not know where to go, it needs the guidance to complete tasks. AGVs work perfectly in these unchanging areas completing tasks such as transportation.

However, there are some major drawbacks. For example, in traditional AGVs it is really costly to change routes and any modification in the working area can lead to the task being uncompleted.

If the working environment is constantly changing a robot that can respond appropriately to these situations is needed. This implies not depending exclusively on an external system that guides the robot or a fixed program. There is a need for an autonomous mobile robot.

So, as it has been explained, nowadays mobile robots need characteristics such as autonomy, intelligence, perception, navigation and mapping among others. These concepts are developed further below [10].

2.2. Autonomy

There are two main definitions of autonomy: *weak* autonomy and *strong* autonomy. Autonomy is measured by the level of self-government, weak being low and high being strong. Self-government means that the machine is able to determine its course of action by its own reasoning process, rather than following a fixed, hardwired sequence of instructions [10].

For example, a robotic arm can complete a simple task perfectly such as welding without human interaction. However, it lacks the power of self-development and adapting to new environments and uncertainties. On the other hand, a rover on Mars, like the one shown in Figure 3 [11], is greatly different. A rover is defined by NASA as “a vehicle for exploring the surface of a planet or moon” [12]. This robot has to complete a more intricate task and adapt to a changing environment with uncertainties. High autonomy is needed in order to achieve its goals. The rover has to plan a route, react to strong winds, find its way through a storm, etcetera.

Figure 4: Mars Rover Curiosity [11]

Strong autonomy has three major steps: perception, learning from experience and changing behaviour. To explain these concepts the Mars rover robot will be again used.

Perception implies interpreting the world, taking into account all the possible phenomena that can occur, like the wind or a rock in the way. The interpretation has to be followed by some kind of planning to achieve the task, such as choosing a path.

Regarding learning from experience, the robot has to learn and recognize patterns. It should have a memory of the past to be able to predict the future, it needs experience. This learning cannot be possible without some level of intelligence.

The third and last step is changing behaviour. If the Mars robot hits a rock it needs to change its path the next time, in order to not make the same mistake. The robot will not succeed if it hits the rock again and makes the same move as before.

Therefore autonomous mobile robots must have high degree of autonomy in order to complete the given tasks in a changing environment. However, the autonomy needed for certain tasks is not the same. AGVs in a factory carrying pallets might have a sufficient level of autonomy by just following lines and avoiding obstacles. Other robots such as the Mars rover need a much higher level of autonomy given that it has to be able to work in an environment that has no human interaction. Choosing how much autonomy a robot needs to have is an important part of its development because it might change the selection of hardware and software.

2.3. Intelligence

The word intelligence is often said without really pondering on what it means. Intelligence refers to behaviour, the ability to acquire and apply knowledge and skills [13]. However, the way in which it is interpreted varies depending on the background and education of the individual.

When evaluating the intelligence of something it is necessary to take two things into consideration: the background and skills of the individual as well as the properties of the object.

It is similar to a chess board. The learning of different moves and strategies is progressive. If a beginner plays against a professional player he will find it difficult to predict the moves of the opponent. The beginner will consider the professional player to be intelligent because he does not understand his thinking process, it is really hard to predict. However, if the situation is analyzed from the professional player point of view, he understands the situation and does not think that the beginner is intelligent when he plays [10].

Therefore, intelligence strongly depends on our background and understanding. One person considered a process to be intelligent because he could not predict it while the other considered it intelligent because of its lack of understanding.

Humans consider themselves intelligent and therefore machines that can do what men do are considered intelligent as well. Returning to the chess example, for humans it is a really hard game to play but, nowadays, it is relatively easy to build a chess-playing machine. Since 1956, MANIAC, a computer developed by Los Alamos Scientific Laboratory became the first machine to beat a human at chess [14].

From this moment on more competitive chess playing machines have been developed. Nowadays, a Google project by the name of AlphaZero used AI to beat in 2018 the strongest chess engine at the time, Stockfish.

What humans can find ordinary and not difficult at all, like walking, can become a great challenge to develop in a machine. For instance, it is more complex to develop a machine able to walk like humans do than one that plays chess.

The key is to associate the concept of intelligence with human behaviour, even the ordinary tasks of our daily lives.

Nowadays, more and more robots are being designed to have "intelligence" in order to understand the environment, learn and change their behaviour. It is usually called Artificial Intelligence, or AI to abbreviate.

2.3.1. Artificial Intelligence

The term Artificial Intelligence is referred to computers which have the ability to imitate aspects of human thought. It is a whole field where software systems are developed with the ability to learn, reason, percept, solve problems and have linguistic intelligence. Usually, the terms of Artificial Intelligence and Machine Learning can be confused, but they are not the same thing. Machine Learning is an application of AI, based on machines having access to data to learn for themselves how to behave. This leads to the concept of Neural Networks, computers designed to classify information as a human does. They work on some probability, but the more data they analyse, the more accurate the machine gets. A typical example is the classification of spam emails. Firstly, the computer gets a pile of spam emails and it starts recognising and establishing patterns between them. Once this step is done, it is ready to work. At the beginning, it may not be much accurate, but as the time passes and it learns what the distinctive elements of a spam email are, it becomes almost infallible. AI modern applications in the field of image recognition are crucial when applying it to autonomous robots. A wide range of autonomous mobile robots uses this aspect to locate objects and interact with the environment.

2.4. Perception

Perception is an extremely important part of any automated system. This means acquiring knowledge of its environment through sensors and filter the data gathered. For more detailed information about sensing refer to H.R. Everett's comprehensive book Sensors for Mobile Robots [15].

Sensors are used to detect simple values such as the temperature of an electronic component, humidity in the air or distance to objects. Some other sensors used in mobile robots are more sophisticated, such as cameras that can stream several photos per second. This data, provided by sensors, has to be processed [16].

Sensor classification

Sensors are classified using two important functional axes: proprioceptive / exteroceptive and passive / active [16].

Proprioceptive measure *internal* system values, such as the temperature of a battery, voltage, or rotation of a wheel.

Exteroceptive measure *external* environment values. Autonomous robots use this data to interpret the environment and interact with it. These sensors include light intensity, cameras or proximity sensors.

Passive sensors measure environmental energy entering the sensors. Examples of these sensors are contact switches, bumpers, inclinometers and CCD/CMOS cameras. Active sensors, on the other hand, emit energy to test the environment reaction. Examples of these sensors are ultrasonic sensors, optical triangulation, optical encoders, capacitive encoders and Doppler radars. However, these sensors have a drawback: they affect the environment. One autonomous robot sensor can influence another robot's measurements.

Table 1, based on Introduction to Autonomous Robots [16], provides a classification of mobile robots sensors and their application. The top sensors are more rudimentary and bottom ones have a higher complexity. Vision sensors are now becoming more popular with their capability to detect objects or human faces using artificial intelligence.

Sensor characteristics

Some sensors provide extremely precise values in a controlled work environment but are filled with errors when they encounter real-world situations. Other sensors have a narrow spectrum in which they can provide accurate values. It is critical to know these characteristics when analysing sensors to choose the most suitable one. Those are explained below:

- **Dynamic range** is used to delimit the input range in which sensors can work adequately. This range is defined by an upper and lower limit. Because the range can be complex the unit of measurement used for dynamic range is decibels. The bigger the dynamic range (dBs) the more extend the sensor can measure. It is important to know the range of an autonomous robot sensor to know how it will react in case it encounters measurements that are superior or inferior to the range. A proximity sensor, for example, might be exposed to an object placed closer than the inferior range, providing invalid data.

- **Resolution** is the minimum difference between two values. Typically, the lower limit of the dynamic range is equivalent to its resolution. This resolution is important in conversions from Alternating Current to Direct Current. An example might be a voltage sensor with a range from 0 to 5V. The conversion is done using 8 bits (255 values). In this case, the resolution of the sensor is the result of dividing the range (5V) between the number of values (255) which is equal to 20 mV.

- **Sampling frequency** is defined as the number of samples per second. For instance, if a sensor has a sampling frequency of 2 Hz, samples are provided every 0.5 seconds. Autonomous robots are normally restricted by the frequency in which they can detect objects.

- **Linearity** is the relationship between the inputs and outputs of a sensor. As the book *Introduction to autonomous robots* [16] states: "A linear response indicates that if two inputs x and y result in the two outputs $f(x)$ and $f(y)$, then for any values a and b, $(ax + by) = af(x) + af(b)$. This means that a plot of the sensor's input/output response is simply a straight line."

Table 1: Classification of sensors used in mobile robotics applications [16]

General classification (typical use)	Sensor Sensor System	PC or EC	A or P
Tactile sensors (detection of physical contact or closeness; security switches)	Contact switchers, bumpers Optical barriers Non-contact proximity sensors	EC EC EC	P A A
Wheel/motor sensors (wheel/motor speed and position)	Brush encoders Potentiometers Synchros, resolvers Optical encoders Magnetic encoders Inductive encoders Capacitive encoders	PC PC PC PC PC PC PC	P P A A A A A
Heading sensors (orientation of the robot in relation to a fixed reference frame)	Compass Gyroscopes Inclinometers	EC PC EC	P P A/P
Acceleration sensor	Accelerometer	PC	P
Ground beacons (localization in a fixed reference frame)	Active optical or RF beacons Active ultrasonic beacons Reflective beacons	EC EC EC	A A A
Active ranging (reflectivity, time-of-flight, and geometric triangulation)	Reflectivity sensors Ultrasonic sensor Laser rangefinder Optical triangulation (1D) Structured light (2D)	EC EC EC EC EC	A A A A A
Motion/speed sensors (speed relative to fixed or moving objects)	Doppler radar Doppler sound	EC EC	A A
Vision sensors (visual ranging, whole-image analysis, segmentation, object recognition)	CCD/CMOS camera(s) Visual ranging packages Object tracking packages	EC	P

P, passive; A, active; P/A, passive/active; PC, proprioceptive; EC, exteroceptive.

2.5. Navigation

At the beginning, the application of a mobile robot was limited exclusively to manufacturing industries, but currently it is commonly used in the fields of entertainment, medicine, mining, rescuing, education, military, space, agriculture and many more. To perform the navigation, the robot is equipped with specific equipment to model the environment and localize its position, control the movement and detect and avoid obstacles.

Safe path planning, choosing the shortest way possible and avoiding obstacles from the initial position to the goal is the most important function of any navigational technique. Because of that, it is easily understandable that this topic is the most researched one in the mobile robots field [17].

Navigation mainly encompasses the ability of the robot to determine its own position and act based on its knowledge and sensor values to reach its goal position as efficiently and reliably as possible, avoiding dangerous or unsafe situations and collisions. This is one of the main problems for autonomous robots, given that their target is to coordinate movements and carry out tasks without any human assistance. To be able to perform the tasks autonomously, the robot has to be able to assess its state and environment to make coherent control decisions.

Fundamentally, it consists of three main competencies: self-localisation, path-planning and map-building-interpretation.

Localization is understood for the robot as establishing its position within a frame of reference.

Path-planning can be described in the robotics field as, given a starting position of the robot, the desired goal and a geometric description of the robot and the environment, finding a path through which the robot can move to the goal avoiding all the possible obstacles. Planning is an extension of localization. Usually, planning is seen as the opposite of reacting, because robots often are only based on one of them, excluding the other. However, mobile robots are set up in a complementary system of both, each being crucial to the other's success. During the execution of the path initially planned, the robot will probably face unexpected events, such as humans or road-blocking obstacles. In that case, the robot would not be able to reach its goal without reacting, so it must incorporate the information from the sensors gained during the execution of the task in order to perform small necessary changes to the path. The concept of planning and reacting to working together is called integrated planning and execution [10].

Mapping is the ability to describe any location within a frame of reference, and after that, being able to interpret it.

Most of today's industrial robots operate with very little cognition, as the environment is assumed to be static and highly structured. Meanwhile, in mobile robotics, the environment is usually non-static and in many cases unstructured, so they use a combination of data from different sources to plan the most effective path to arrive at a goal [16].

2.6. Locomotion

One of the basic abilities of autonomous robots is locomotion, the skill of moving from one place to another. Nowadays, as research advances, there are more and more possible ways for robots to move, either walking, jumping, running, rolling, flying, etc. so it is a crucial aspect in the mobile robot design. However, most of the autonomous robots, especially the industrial ones, work with wheels, as they are really simple and provide an excellent efficiency on flat ground. These wheels can also be part of a railway system, when the surface that the robot covers is not regular, so the rolling friction is minimized.

Basically, there are four types of wheels, as shown in Figure 5 [16].

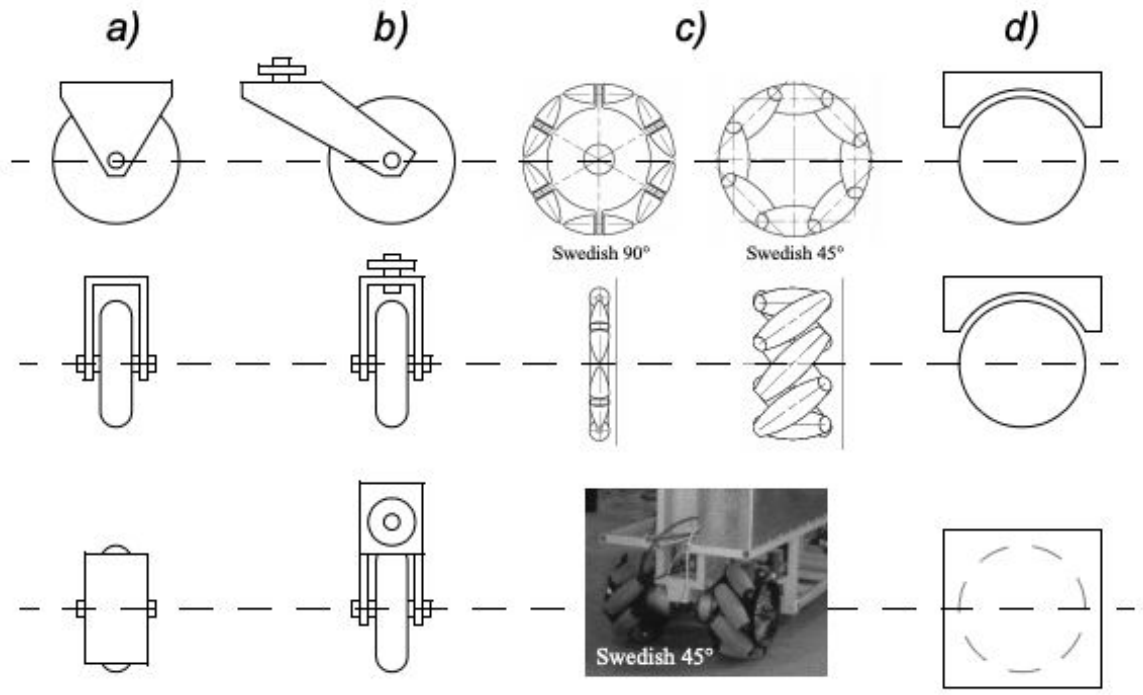


Figure 5: The four basic wheel types. (a) Standard wheel. (b) Castor wheel. (c) Swedish wheel. (d) Ball or spherical wheel [16].

The first two wheels, standard and castor have only two degrees of freedom, as they have a primary axis that allows them to move forward and backwards. The main difference between these two wheels are the centre of rotation since in the standard one it passes through the contact patch with the ground, meanwhile, the castor wheel applies a force in the robot chassis during steering.

The Swedish and spherical wheels both have three degrees. What is special about the Swedish one is that it can move in another direction than forward and backwards, usually perpendicularly -totally or partially- to the conventional direction. The spherical wheel is purely omnidirectional and it is often used as a support, apart from helping to steer, although its realization is technically difficult.

The wheel has always been the most popular locomotion mechanism in autonomous robots because of its versatility, furthermore, the balance problem present in legged-robots is avoided, as three wheels are enough to maintain the equilibrium. Instead of that, the research covers traction, stability and control.

3. Method

The purpose of this section is to explain the methodology used. This project is extremely practical with many iterations and changes in its structure and, given this fact, Scrum was used.

Scrum is an iterative and incremental project management methodology or framework, mainly used nowadays for software development and, according to the 12th annual State of Agile report [18], up to 70% of software teams use Scrum or a Scrum hybrid. This methodology was mainly designed for those projects in which a new tested product has to be delivered every 2-4 weeks. For this reason, Scrum is not limited exclusively to software development, as it works perfectly on those projects that have to be developed under complexity and ambiguity of the task, which is the case of this project. Those terms are used because this framework is based on the recognition that the customer may change his mind about some requirements and that unpredictable challenges will arise.

Scrum is designed for teams, working as a unit, who divide the work into short actions that should be completed within timeboxed operations, usually from one week to a month. These divisions are called sprints. A track of the development of the project should be done in the daily scrums, 15-minutes meetings used to stake out the goals. It is also important to maintain the product in a potentially shippable state once the sprint is finished, as the main difference between this methodology and Waterfall methodology.

Following there is explained the process necessary to apply Scrum:

- **Product Backlog:** This comprises the work that has to be done, organized according to priority, either because they are more important or more urgent, and it should be discussed and agreed by the whole team. This would be the problem description.

- **Sprint Planning:** Once the Product Backlog is done, and according to it, the team starts with the higher priority items and the approach of how to achieve the goal. It should not be much detailed, due to unforeseen events, and it should be collaborative so everyone can apport. This is the Gant Schedule.

- **The Sprint:** Periods of time when the objectives are completed, with an ideal duration of one-two weeks and no longer than one month. The scrum meetings commented before should be held daily to make sure that the objective is completed, but big changes on the goal should be avoided not to endanger the project. All the sprints and their estimated duration are described below, and then a scheme is presented in Figure 6:

    - Planning: In the first stage the planning was carried out. This implied figuring out how the thesis was going to be structured and what things were going to be done in the project. It took almost one week.

    - Literature Review: The second stage consists of reviewing all the relevant information available needed to complete the task. It is important to note that

other projects were taken into account to follow their structure and apply ideas to this project. Two weeks were spent on this, although it was finally completed several weeks before.

- Platform evaluation: Once the sufficient knowledge was gathered it was time to evaluate the available robotic platforms. There were many ways to choose a robot and this was carried out in the platform evaluation following different criteria, which took almost one week.

- Platform selection: After evaluating the platforms it was time to choose the most suitable one for this project. This platform should be the one that best fulfils the project specification. It was important to check if a platform was useful or easy to implement before selecting it, among other things. Therefore, one week was spent.

- Implement platform: This stage was crucial. There had to be a great understanding of how this platform worked. All the hardware and software used had to be mastered before developing anything. This part was extremely time-consuming. This was mainly because it was sometimes hard to find examples or quality tutorials that help understand how a given software works. This was one of the most important and tedious tasks, consequently, a month was approximately spent here, although the learning continued throughout the whole thesis.

- Develop task: Once the platform to develop the autonomous robot had been mastered it was time to develop the task. This stage and the building of the model were done together. This took about a month to fulfil.

- Debug: When all the code was implemented, a debug was carried out to solve all the existing errors. As there was a great number of errors, almost one week was spent here.

- Sufficient functions/lack of time: Once the robot was performing the proposed tasks correctly, an analysis was conducted in order to assess if there were more assignments that could be implemented within the time left. As the project went according to the time stipulated for each part, a fortnight was left to develop other tasks.

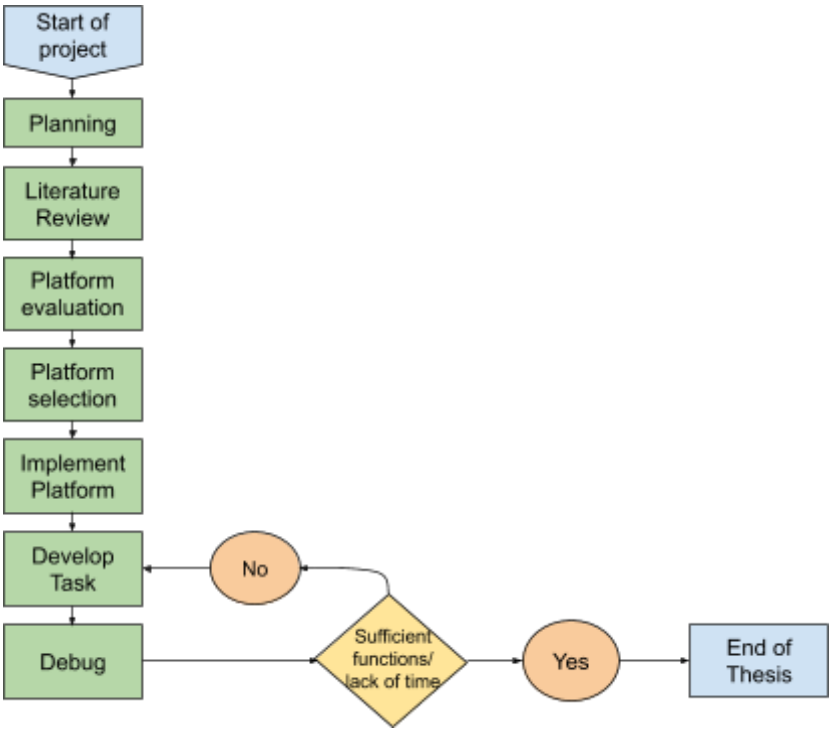


Figure 6: Sprints carried out in this thesis

- **Oversight by the Master:** An expert on the field that oversees the project giving advice and direction, but without interfering directly with the project.

- **Completed Sprints:** The sprint will be considered completed once the task is finished and ready to be shown to the customer, so the product needs to be tested before. In this case, the product would be the robot and the customer would be the supervisor.

- **Review:** This is a meeting to see what went well on the sprint and what did not go so well, so it can be taken into account in the next sprint.

- **Repeat:** Once all the steps described have been completed, it starts again by going to the next goal in importance order, but now trying to improve all the things that did not work, to make the process highly efficient.

4. Selection of Platform

4.1. Description of Hardware

This section includes the different platforms that are currently available to achieve the task of this project: build an autonomous mobile robot. To do so, the main platforms that can easily be found on the Internet have been taken into account. A brief explanation of every one of them can be found below.

Raspberry Pi:

Raspberry Pi is a credit-sized Single Board Computer (SBC) manufactured by the Raspberry Pi Foundation. It is very popular due to its low price (SEK 350) and features. The latest model while writing this thesis is the Raspberry Pi 3 Model B+, shown in Figure 7 [19], which comes with a 1.4Ghz 64-bit quad-core processor, 1GB SDRAM, Bluetooth 4.2 and a dual-band wireless LAN. Also, it has many inputs and outputs, such as Ethernet, HDMI, USB, SD Card port or CSI to connect a camera. This computer works on Debian Linux OS, although an optimized version of this OS called Raspbian is the most used [19, 20].

Figure 7: Raspberry Pi 3 Model B+ board [19]

Arduino:

Arduino is the name of a family of open-source microcontroller boards. As its hardware is open-source, there is a wide range of expansions (shields) that enhance some features of the microcontroller. The most purchased board is the Arduino UNO REV3, displayed in Figure 8 [21], which is sold for SEK 200.

It is built on the ATmega328P, a clock speed of 16MHz, 2KB of SRAM and 32KB of Flash Memory. It has 14 GPIO Pins and a USB port. Arduino boards are programmed in Arduino Software (IDE), written in Java and based on Processing, and it can be installed in Windows, Mac OS X or Linux [21].

Figure 8: Arduino UNO REV3 board [21]

Lego Mindstorm:

Lego Mindstorms is a hardware and software platform to make robots with the Lego Technic blocks. It is especially recommended for children due to its easy building, and the computer costs around SEK 2000. Its processor is the ARM9, and it has a USB port for Wifi, Micro USB card reader and 4 motor ports. As it is not open-source, it is compulsory to buy the sensors and actuators directly from the Lego website. The EV3 Brick (the controller) displayed in Figure 9 is programmed with special software called EV3 Software, which can be installed either in Windows or Mac [22].

Figure 9: Lego Mindstorms EV3 Brick [22]

Asus Tinker Board:

Asus Tinker Board, shown in Figure 10, is a Single Board Computer (SBC) made by Asus and specially designed to be compatible with Raspberry Pi components. It offers high specifications according to its price, SEK 600. Its CPU is a Rockchip 32-bits quad-core processor (which can play video up to 4K quality), accompanied by 2GB RAM, integrated graphics, Wifi and Bluetooth 4.0. Of course, it is full port-equipped: Gbit LAN, HDMI, USB, SD card 3.0 interface, 28 GPIO pins or CSI. It runs in TinkerOS, a Debian Linux version, also in Android [23].

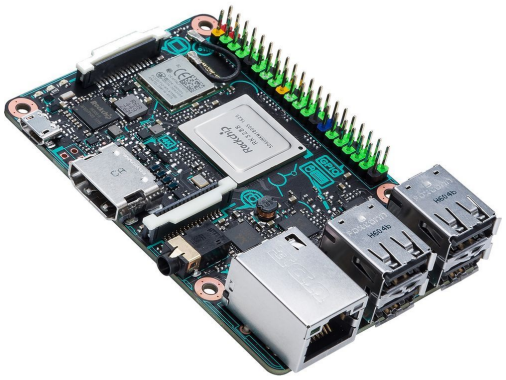


Figure 10: Asus Tinker Board [23]

4.2. Selection of Hardware

In this chapter, the final hardware platform is selected, following different criteria. These requirements have been sorted by relevance, taking into account that this is an educational project.

- Price: The platform should be low-priced, but fulfilling the required components, to overcome budget limitations in case there is a need to buy more.

- Constrictions: It would be better for the project to have some freedom in choosing and purchasing new parts, so the robot can be easily expanded with new sensors, actuators, drivers, etc.

- Processing: The robot must run multiple programs together and deal with both sensors and actuators information, so it needs a capable processor.

- The number of ports available: To provide more flexibility to possible future expansions.

- Operating System: As this is an educational project, the platform will be more suitable if it can run in more OS to give students more flexibility.

Once the criteria are described, the analysis of the platforms can begin. For the validation, the weighted-sum matrix designed by Pahl and Beitz is used. In Table 2, different weights and punctuations are assigned to the requirements and at the end, the platform with more points is the most suitable one.

Table 2: Weighed-sum matrix

Criteria	Weight	Raspberry Pi	Arduino	Mindstorms	Tinker Board
Price	2	5	5	1	3
Constrictions	2	5	5	1	5
Processing	1	5	2	2	5
Number of Ports	1	5	3	1	5
Operating System	1	3	5	5	3
Total		33	30	13	21

The Raspberry Pi is the platform that adapts best to the project, due to its low price, open-source system, fast-processing and versatility. Given this characteristics, it has been selected for the development of the robotic platform.

4.3. Required Components

In order to perform the task, the robot needs some essential components, explained below divided into two main categories. This list includes all the specific components that the robot kit purchased should have, but of course, the more expansions the better (as long as the price is reasonable).

Electronic:

- Raspberry Pi, model 2 or 3: The Raspberry Pi is the computer where the code is written and executed and is in charge of all the logic operations in order to move the robot.

- Raspberry Pi Camera Module (The official V2 or any other): The camera is really important in the project, as it is used to locate the other robot.
  It needs to have good image quality, colour fidelity and low-light performance. Furthermore, the camera could be used to do face-recognition, high definition video or still photographs for example. This device must have a CSI port to connect to the Raspberry Pi.

- Motor Driver Module (as TB6612 from Adafruit, Motor Control Board or Raspberry Pi Motor Driver Expansion Module): This expansion is connected directly to the Raspberry Pi and it gives the ability to control two DC motors and turn them in both directions.

- MicroSD Card 16Gb: This is where the Operative System and all the software is installed. 8GB is the minimum, so with 16GB is sufficient for most of the projects.

- Ultrasonic or Infrared Sensor (like HC-SR04 or ST188): As the robot have obstacle-avoidance it is imperative to have at least one Ultrasonic Sensor. This model can detect objects from 2 to 400 centimetres with a measuring angle of 30º. It works sending a high-frequency sound, and when it finds an object is reflected and received by the transmitter. Measuring the time between the sent and received sound allows the sensor to know the distance.

- Two or four DC Motors 5V: These motors are in charge of the mobility of the robot. To keep the project simple, the robot rotates using these motors directly (As a tank does) so there is no need of a Stepper Motor to turn the wheels.

- Battery Power Pack for the motors and MicroUSB break: As the robot is autonomous, it means that it does not have to be connected to a plug in order to work, so it needs an External Battery Pack to be powered.

- Photoboard (Breadboard): Used to establish all the connections easily between the components.

Design:

- Chassis (Could be bought or printed): This is the structure of the robot, where all the components are screwed. It can be directly bought or printed downloading a 3D model online.

- Wheels: The wheels must be of the right size according to the chassis, and it is preferable that they are made of plastic and rubber to avoid slips. They should be a Standard or Castor wheel as they will only need two degrees of freedom (Chapter 2.6. Locomotion). Also, if the robot only has two wheels, a solution to improve its stability is to add one or two omnidirectional or spherical wheels, as they will need three degrees of freedom, to make the third support and help to turn.

4.4. Description of Robot Kits

The next step is to decide which robot kit should be purchased. Firstly, some of the most popular Robot Kits that can be found online are described and then the choice is carried out. The purchase link of each kit is displayed in Appendix I: Purchase links. All the prices shown are only of the robot kit itself, without including delivery expenses or possible

taxes. Also, the prices correspond to the Euro (€) - United States Dollar ($) - Swedish Crown (SEK) change established in spring 2019, so it is subject to changes.

Turtlebot:

Turtlebot is a robot kit with open-software, created by Melonee Wise and Tully Foote. Its major strengths are the 3D sensor and the implementation with ROS -it was designed in collaboration with the original developers of ROS- although its price is quite high, SEK 5200 the most basic model, called TurtleBot 3 Burger. It comes with a Raspberry Pi 3 as computer, 360º laser distance sensor, two motors and a battery. It would not be necessary to buy anything else to run this robot.

UCTRONICS Smart Robot Car Kit for Raspberry Pi:

This Robot Car is developed by UCTRONICS was originally designed for education and entertainment. It includes all the components specified, with the exception of the Raspberry Pi itself and the two batteries. Also, this car has five motors, one for each wheel and another servo to turn. It costs 80€ on its official website, although as it is said before, the Raspberry Pi (40€) and two rechargeable batteries (6€) so the total would be of 126€, or SEK 1300 approximately.

WaveShare Alphabot2 Pi:

The AlphaBot2 robot kit is also designed especially for educational purposes, and because of this, it includes all the basic features to develop basic tasks. It already includes the Raspberry Pi 3B in its final price of 106€, but it does not include the batteries, so the final price would e of 112€ or SEK 1160.

Self-Built:

Also, a possible option would be to build the robot from scratch, buying each part separately. The inspiration source for this has been a Youtube video called “Build the Structure of a ROS + Raspberry Pi Robot | Ep.1” [24]. A list of all the pieces necessary is presented:

- Raspberry Pi 3 Model B
- 16GB Class 10 MicroSD Card
- Raspberry Pi Lipo Battery Power Pack Board
- Lipo Battery for the Powerboost
- Raspberry Pi Motor Driver Expansion DC-Stepper Motor Driver
- Raspberry Pi Camera Module
- Cable of the charger
- Adapter to USB
- H Bridge Motor Driver Hats
- Jumper Wires
- Jack Connectors
- AA Batteries
- Bluetooth keyboard

This would make a total of 286€ approximately, or SEK 2973.

4.5. Selection of the Robot Kit

Next, the selection of the robot kit is done following different criteria, explained below. These requirements have also been sorted by relevance, as it is done on the Selection of Hardware chapter.

- Price: The platform should be low-priced, but fulfilling the required components, to overcome budget limitations in case there is a need to buy more.

- Ease of extension: Once this project is finished, the robot kit should have the freedom to enlarge the components in the platform so it can be reused for any other project.

- Ease of construction: As this project is not focused on the construction of the robot, it should be as easy to build as possible not to delay the project.

- Implementation: In this section, the community and libraries behind each kit are analysed, as it is an educational project is preferable to have some feedback in case there is some problem.

Now that the criteria are described, the analysis of the kits can be carried out. To do so, the same method that as the Selection of Hardware is used, the weighted-sum matrix designed by Pahl and Beitz, in Table 3.

Table 3: Weighed-sum matrix

Criteria	Weight	Turtlebot	Smart Robot Car	Alphabot2	Self-Built
Price	2	1	4	5	2
Extension	1	1	1	1	5
Construction	1	5	4	4	1
Implementation	1	5	3	3	3
Total		13	16	18	13

Finally, the AlphaBot2 is the kit selected, mostly because of its low price, ease of construction. In addition, it only has two wheels, so the control is simpler as well. Furthermore, the implementation is effortless as there are libraries already developed on its manufacturer company, WaveShare, web page.

5. Understanding ROS

5.1. A Brief Presentation of ROS

ROS means Robotic Operating System and it is a software framework for programming robots. Its original idea and design came from Stanford AI, but then it was created and developed by Willow Garage in 2007. Nowadays, it is maintained by the Open Source Robotics Foundation. C++ and Python are the languages that ROS is built on.

It is usually said that ROS consists of four parts: Infrastructure, tools, capabilities and ecosystem. Infrastructure is referred to the multiple processing and computer architecture that is already created in the libraries. This middleware has a huge variety of tools and extensions, some of them already implemented, like Gazebo. Capabilities, as it comes with a wide range of common robotics algorithms making easier to implement new code. Lastly, its ecosystem is crucial because it is open-source and there is a huge community behind, so there are plenty of problems solved in books and online.

This is the middleware used in this project as it provides multiple advantages, explained below. For further details refer to [25].

- It is open-source, which means that there is a community to share the work done and ask for help. Also, already written software packages can be reused and implemented, so there is no need for being an expert in image recognition to implement one of these libraries, for example.
- It comes with lots of infrastructures, tools and capabilities already implemented, with Gazebo among them.
- Unlike most of the robotic applications, ROS has multitasking control and crashing procedures, so even if a node crashes, the system can continue the task.
- ROS is very light and does not need many resources to run, which perfectly fits with this project as the Raspberry Pi is not as powerful as a normal computer.
- Possibility to choose between different programming languages, like C++ or Python, between others. This chance is ideal for this project since the libraries that AlphaBot2 brings are written on Python and yet most of the image recognition code that can be found online is written on C++.

Among these advantages, the most important for an educational thesis like this one is the possibilities that a big community brings, from problem-solving to tutorials or already programmed code.

Nonetheless, ROS has also disadvantages, such as:

- It is hard to learn. This implies that, before starting to program, there is a need for learning basics ROS and Ubuntu terminal commands in order to be able to move within the interface, and most importantly, how ROS works. Furthermore, as this middleware does not work with a certain hardware, a longer debugging time has to be spent.

- It is not fully developed yet, so newer versions can vary and not work with older robots.
- Security and scalability are not crucial, as it does not require any authentication.
- It is only well supported for Ubuntu Linux.

5.2. ROS Structure

To understand ROS fully, first its structure has to be explained. As the ROS website presents [26], there are three levels of concepts: Filesystem Level, Computation Graph Level and the Community Level. This last level is composed of those resources that make possible to share software between the ROS community, and as they are not crucial in this thesis they are not explained. For further information about the ROS structure, this chapter is based on the previously mentioned ROS website [26] and also in the book "Mastering ROS for Robotics Programming" [27] and the "Using a general robot programming system to control an industrial robot" thesis [25].

5.2.1. ROS Filesystem Level

This first level covers the files and their particular organization that can be found on the disk, and it includes:

- **Packages:** They are the main unit as well as the most basic entity of ROS. All packages contain ROS execution programs (also called nodes), ROS-independent libraries, configuration files, datasets and third-party software organized together as a sole entity. This simple organization allows these packages to act as modular blocks, which of course, facilitates the exchange of files between the community [28].

- **Metapackages:** These special packages represents a whole group of packages that are related to each other. In older versions of ROS, they were called Stacks.

- **Package Manifests:** They are the "package.xml" that can be found inside each package, and they contain information it, such as its name, version, description, license dependencies or any other metadata.

- **Repositories:** Collection of packages (although they can contain only one), which share a common Version Control System or VCS.

- **Message (.msg) types:** Type of information that is shared from one ROS process to another.

- **Service (.srv) types:** Type of interaction between processes, in which one process generates a request to be replied by another.

5.2.2. ROS Computation Graph Level

This second level covers the computation done using the ROS processes called nodes within a peer-to-peer network. A basic scheme can be found in Figure 11 with the main concepts presented below. The computation graph level comprises:

- **Nodes:** They are the processes that perform the computation. Each one of them is written with the use of a ROS client library, such as roscpp (For C++ files) or rospy (For python files). Nodes build simple processes which make them easier to debug. They subscribe to topics to receive data and publish information to topics, in order to be used for other nodes [28].

- **Master:** it provides name registration and lookup to the rest of the Computation Graph. There always has to be a Master within a system, if not the nodes would not be able to find each other and exchange messages.

- **Parameter Server:** It allows data to be stored by key in a central location, which can be modified by the nodes. It actually is a part of the Master.

- **Messages:** They are the information that the nodes exchange when they communicate. There are many standard types, such as Twist, String, Point or Boolean…

- **Topics:** Nodes cannot communicate between them directly. They send a message that is *published* in the topic, and then another node, *subscribed* to that topic, reads that message. There can be many subscribers or publishers in one topic, as well as a node can publish or be subscribed to several topics. The communication in these entities is unidirectional, as a difference from the ROS services explained below.

- **Services:** They are the other way for nodes to communicate between them. Unlike ROS topics which are based on publish-subscribe, services consist of a request-response system in which a node request and another node send a response.

- **Bags:** Bags are a format for saving and playing back ROS data, such as sensor data. This tool is subscribed to one or more topics so it can store, process, analyze and visualize the messages as they are received, which allows the bag to forward the messages as if the same nodes sent them again [28].

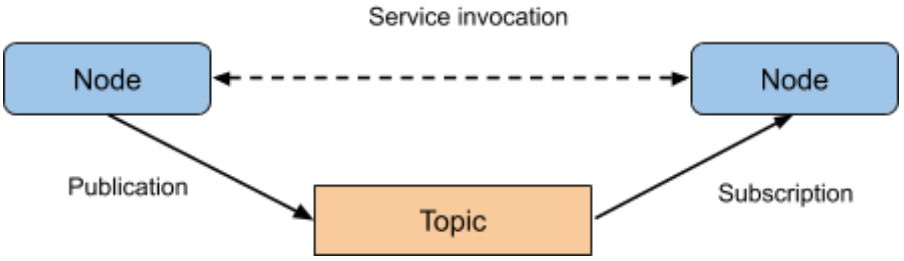


Figure 11: ROS basic concepts

5.3. Building ROS workspaces and packages

This chapter is dedicated to the creation of a workspace and a package in ROS. A catkin workspace is a folder where packages can be modified, built or installed, so it is one of the first concepts taught when learning ROS. “Catkin” makes reference to the current build system of ROS, as a successor to the original rosbuild.

After creating the mentioned workspace, a few concepts should be known. Inside each workspace there are three different spaces: Src (Source), Build and Devel (Development). Source contains the code of all catkin packages and is also the place where the code can be extracted, checked or cloned. It is the folder in which all the work is developed. Build is where CMake is called to build the catkin packages in Source, as well as the place where cache and intermediate files are stored. Lastly, Devel is where built targets are placed to be installed. The previously mentioned CMake is a .txt file that describes how to build the code and where to install it to [29].

To build the workspace (Referred to WS), the instructions below should be followed in the Ubuntu terminal, Home directory:

Create the Catkin WS and its src

```
mkdir -p ~/catkin_ws/src
```

Change the working directory to the just created Catkin WS

```
cd ~/catkin_ws/
```

Create the whole structure, with Build and Devel spaces

```
catkin_make
```

Tell the bash that you are working on that WS specifically

```
Source devel/setup.bash
```

The outcome of the instructions above is illustrated in Figure 12.

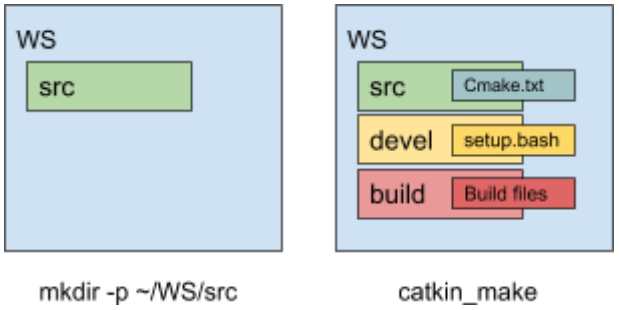


Figure 12: Building a workspace

Once the mentioned workspace is created, the next step is to create a package. As well as with the workspace, there are some concepts that need to be taken into account after creating it. Every new package created has, at least, these same directories: CMake.txt, include, package.xml and src. CMake.txt is the file explained above. The include file consists of libraries that the package needs. Package.xml is the "Package Manifest" as explained in Chapter 5.2.1. ROS FileSystem Level. Finally, the src folder is the same as the src in the workspace, the place where the code is placed.

All this structure can be seen in Figure 13.

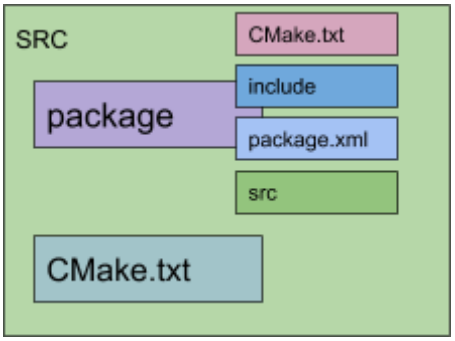


Figure 13: Package structure

To create the package, the instructions are detailed below:

Go to the src inside the WS previously created

```
cd WS
cd src
```

Create the package in src. Along with the command, the name of the package and the depending packages must be included. These packages are the dependencies in which on which the package depends, some examples could be rospy, roscpp or std_msgs.

```
catkin_create_pkg name depending_packages
```

After making all these changes in src, catkin_make is the last step to build the whole structure.

```
catkin_make
```

The outcome after creating both the WS and the package is illustrated in Figure 14.

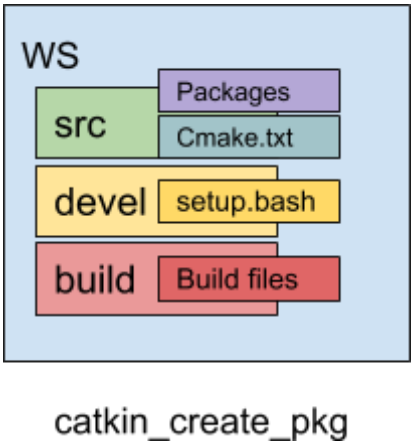


Figure 14: Workspace final structure

6. Robot Task Development

6.1. Task description

In this chapter the development of each node used for the performance of the robot is explained. It is important to note that this part is exclusively explaining the code, to deeply understand each part refer to the repository in GitHub [30].

To develop the tasks the authors chose to use the camera and the wheels. Once this basic task was developed, more complex tasks could be added. The first task consisted of localizing, centring and pursuing a ping pong ball coloured in blue. The main idea is that the robot is able to detect the colour and shape of the ball and follow it without any additional control. This implies that the robot is able to turn to pursue the ball and even adapt to a moving trajectory of the ball. It also means that it has to maintain a certain distance with the ball. In addition to this, an online model of the AlphaBot2 will be built and tested on Gazebo. It starts with basic functionalities, like moving forward and backwards, and it will be expanded depending on the time left.

After the first task was developed, the authors implemented a remote controller to perform different tasks. Furthermore, these additional tasks can be easily expanded depending on the time left, two weeks in this case. During those weeks, two extra tasks were implemented: a “manual” mode to control the robot with a remote controller and a “ball_following_servo” mode to face the ball using only the servo motors located on the camera, instead of the wheels. All three tasks are referred as modes of the robot. Modes are selected using the remote controller, which will be explored with detail further below.

The modes are programmed using different ROS nodes, previously explained in chapter 5.2.1. ROS Computation Graph Level.

The ROS programmes are classified into three sections coined by the authors as input chain, control and output chain. The input chain processes the data provided by the sensors and sends the information to the control. The control section computes the information through different algorithms and provides the output chain with the necessary inputs. The output chain simply transforms the inputs to be processed by the actuators. The process described is shown in Figure 15. It is important to note that the “find_ball_node” and “control_node” are performed in a different computer with more computational power. The rest of the nodes are launched directly from the Raspberry Pi.

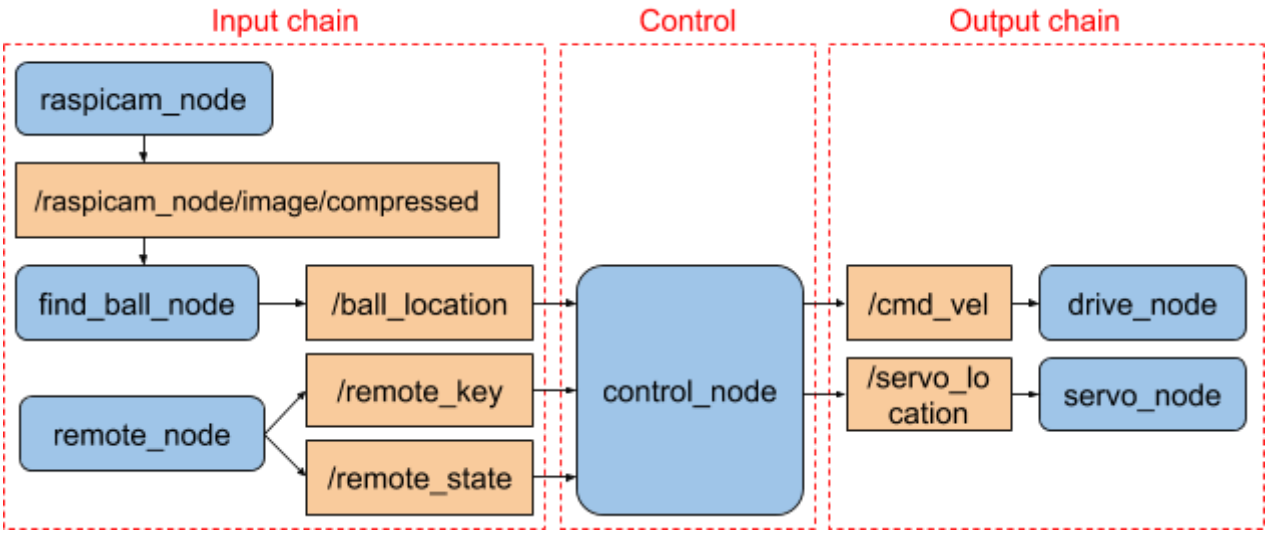


Figure 15: ROS task structure

6.2. Input chain

To be able to process the necessary information, the robot program needs an input chain. Inputs for the control node are obtained based on the different input data that the robot has, which is processed and published on a ROS topic. It is important to remember that ROS nodes can publish (write) and subscribe (read) from topics, as explained in chapter 5.2.2 ROS Computation Graph Level.

In this section, the different and basic input nodes used to perform the different tasks are analyzed.

6.2.1. Camera node

One of the most important parts is the camera of the robot. It allows it to see the world and react to it. This node provides the image input that will be used by the control node. There is a ROS package available at GitHub that publishes the streaming video of a Raspberry Pi camera into a topic. Figure 16 shows the structure of the camera input node:

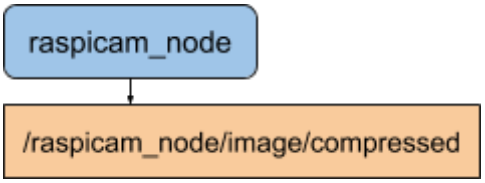


Figure 16: ROS camera node structure

6.2.2. Find ball node

For the purpose of utilizing the camera, a node that is able to locate a ball within an image was developed. Figure 17 shows the structure of the ROS node. It first subscribes to

the topic where the image is being published. Secondly, the image is processed and the coordinates of the ball are obtained. Lastly, these coordinates are published on a topic. All the steps that involve the processing of the image are explored in the following paragraphs.

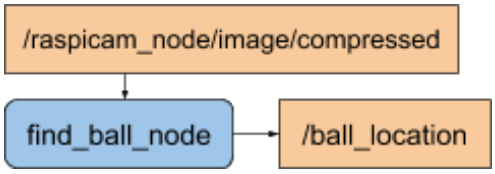


Figure 17: ROS find ball node structure

First of all, the node needs to have access to the camera stream video. This can be easily done by subscribing to the topic where the camera node publishes, as shown in Figure 16.

Secondly, to be able to process the image OpenCV will be used. OpenCV is a library of computer functions used for real-time computer vision. This library was selected for image recognition given that it is already implemented with ROS. There are many things to be taken into account when locating the ball [31]:

- **HSV Colour segmentation**: OpenCV is able to use the HSV colour space, which works really well with shadows and lighting variations. This acronym stands for Hue, Saturation and Value. Hue refers to the dominant wavelength and is especially useful because it only uses one channel to describe colour. The other two components, Saturation and Value depend upon the lighting and the surface of the object. In the task, the robot follows a blue ball [31].

- **HSV threshold values**: The target is to detect a blue ball, so a range of blues has to be defined so that the robot is able to differentiate the ball from the background. To determine the range of blue, different pictures of the ball were taken in different lighting conditions as shown in Figure 18. These photos were analysed by the authors and different blue Hue values were assigned between [218, 208], upper and lower values respectively. After this range has been set, the colour segmentation with HSV generates a black and white image. White pixels belonging to the HSV colour space of blues; and in black the rest.

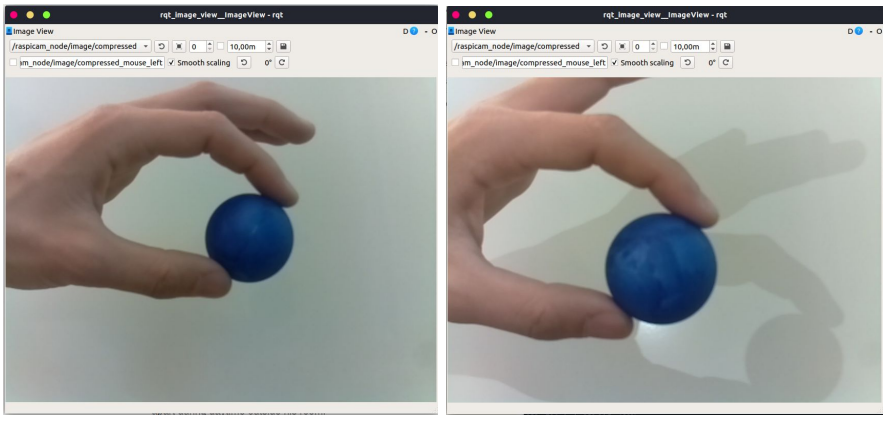


Figure 18: Different lighting conditions to obtain the HSV blue range

- **Erode and Dilate**. Now that the range of colour is set is time to filter. To perform the colour segmentation two functions are used: erode and dilate. These functions are used to expand or diminish the filtered area. It is especially useful to reduce noise and fill gaps. For the blue ball, three iterations of dilating are used [31]. The outcome of this process is shown on the right side of Figure 19.

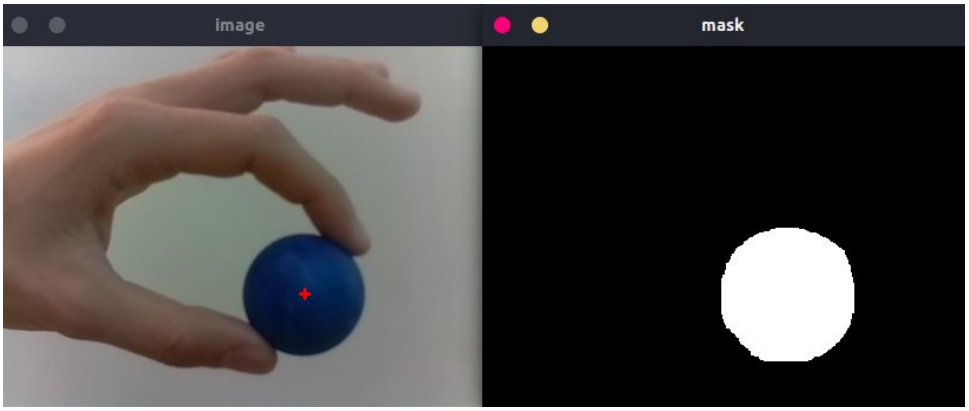


Figure 19: On the left, the original image with the mass centre; on the right, the filtered image.

- **Mass centre**: Once the ball has been filtered, the centre of the ball needs to be located. In order to provide the exact coordinates of the ball, the mass centre needs to be calculated. This is shown in Figure 19 with a red cross on the left-hand side of the image.

Lastly, once the coordinates of the centre of the ball have been obtained, the outcome is published on a topic. Any node can obtain the ball location if they subscribe to its topic.

6.2.3. Remote control node

This node is crucial for the overall performance of the robot. The different tasks to be performed are sent via the infrared remote control. In addition, the remote is also used to control the robot manually.

The node receives the signal from the remote. It then associates each value with an action and publishes the result into a topic depending on the type of button pressed. The structure is shown in Figure 20.

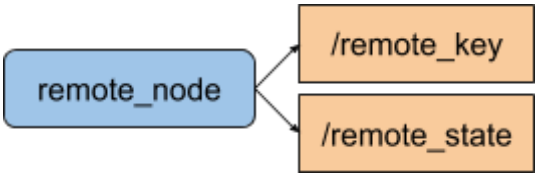


Figure 20: ROS remote node structure

Firstly, the node receives the input signal from the infrared remote. The signal is decoded into a hexadecimal number. Each number is associated with a key in the remote. These numbers are then linked to a string that the robot control node receives.

There are two types of buttons, red ones to select the task and numbers to specify direction. These two types of actions are published in different topics, one for the task and one for the direction. The relationship between buttons and actions are shown in Figure 21. For example, if the user presses button number 2, the remote node will publish in "remote_key" the string "forward". On the other hand, if the top left red button is pressed, the remote node then publishes in "remote_status" the string "manual".

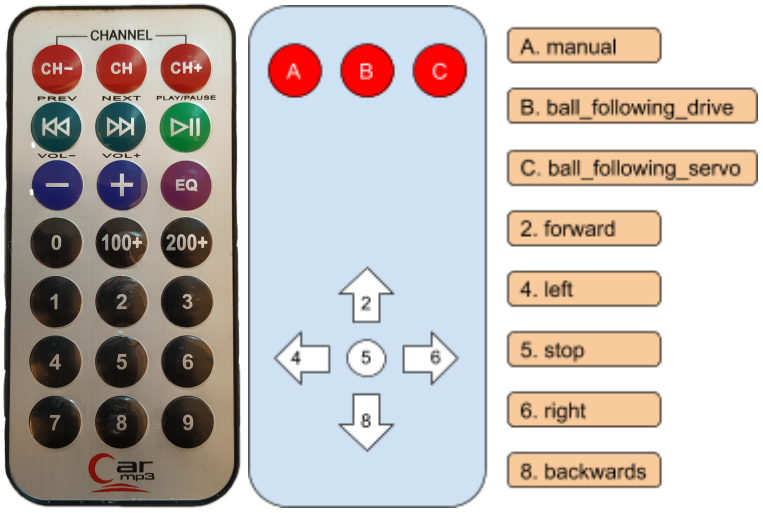


Figure 21: Infrared remote control

6.3. Control

6.3.1. Control node

The control node is the central coordination point. It is in charge of gathering all the necessary information from the input nodes and through different algorithms obtain outputs read by the output nodes. There are different processes that the node goes through.

The control node structure is shown in Figure 22. It receives values from the input chain and delivers the data to be read by the output chain.

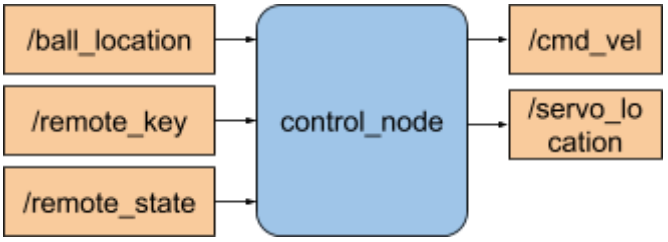


Figure 22: ROS control node structure

- **Subscribe to topics**. To be able to process this data, the first logical step is to subscribe to all the input topics where the input nodes publish. Each time a value is

published into a topic, a callback function is called. Inside the callback function the value obtained is then associated with an instance.
The location of the ball is used to help illustrate this concept. To store its value, the control node is subscribed to the “ball_location” topic. The callback function is used and the value is stored under “self.ball_location” as shown in Figure 22. This process is performed independently from the rest of the code. Furthermore, the instance value will always be up to date. Figure 23 also shows how the “remote_key” and “state” values are stored. There are many more instances saved from topics, each one follows the same storing process.

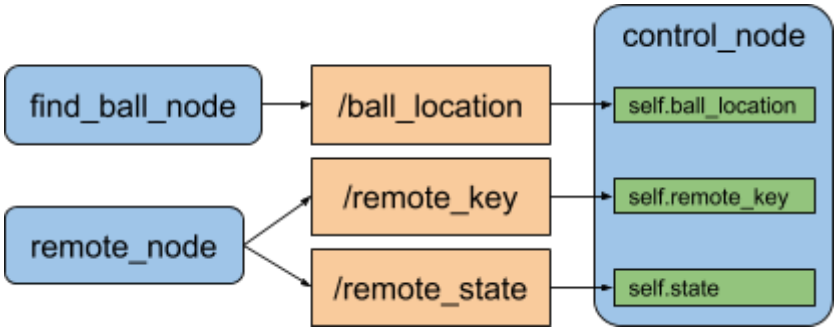


Figure 23: Storing a value in the control node

- **Select the task to be performed**. As described in the remote node, data published in “remote_state” is used to determine the different tasks. When a key is pressed, the remote node publishes that information into a topic. It is then read by the control node and a task is selected. The tasks associated with each key are: “manual”, “ball_following_drive” and “ball_following_servo” as shown in Figure 24.

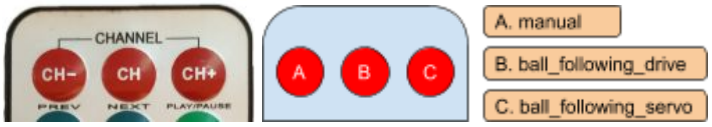


Figure 24: ROS remote keys for task selection

- **Perform the given task**. Once a task has been selected, the node goes through its algorithm to compute the velocity.

- **Publish on topics**. Lastly, the value obtained as outputs by the control node are now published on the different output topics. These topics will be read by the output nodes.

6.3.2. Control Node tasks

The control tasks, as shown in figure 24 are:

- **Manual.** A remote controller is used to control the robot.
- **Ball_following_drive.** Using the camera to locate the ball and follow it using the wheels.
- **Ball_following_servo.** Using the camera to locate the ball and face it using the servo motors.

6.3.2.1. Manual

The manual mode is controlled by the remote. The different numbers are used as arrows to select the direction of the robot. Basic instances used in the manual node are shown in Figure 25.

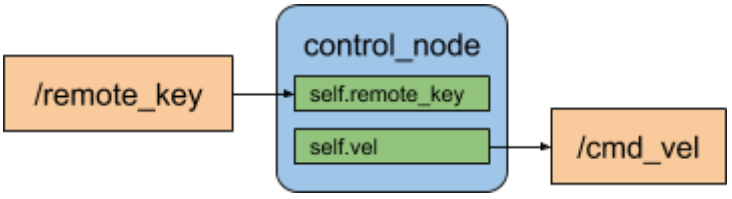


Figure 25: Instances used in manual mode

This mode is structured as follows:

- **Check direction instance:** The control node is subscribed the "/remote_key", where the direction received from the remote controller is published. The content in the topic is associated with the instance "remote_key" that contains the direction from the remote. This association is shown in Figure 26.

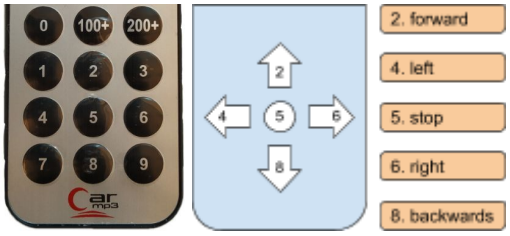


Figure 26: Number association with the direction

- **Publish outcome into the topic:** Associate each direction with a velocity. For example, if the "remote_key" instance is "forward", the class instance "vel" is assigned a positive linear velocity value.

6.3.2.2. Drive Ball Following

The ball following mode uses the location of the ball provided by the "find_ball_node" to face the ball using the wheels of the robot. Depending on the position of the ball certain velocity is assigned. Basic instances used in this mode are shown in Figure 27.

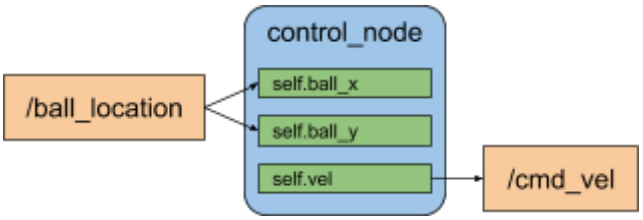


Figure 27: Instances used in ball_following_drive mode

This node is structured as follows:

- **Normalize ball location**. The location is given in pixels, which depend on the resolution of the video. To be able to use this mode with different resolutions the position is normalized. This is easily done by dividing the ‘x’ or ‘y’ value between the width or height of the image.

- **Assign different velocities to different positions**. Depending on the normalized coordinates of the ball (x, y), linear and angular velocities are set. Figure 28 shows the division constrictions. Limits (-0.5,+0.5) are normalized for any image resolution. The words inside the boxes refer to the direction that the robot has to move depending on the location of the ball. The purpose of the code is to maintain the ball at the “centre” of the image. More detailed explanations on GitHub [30]

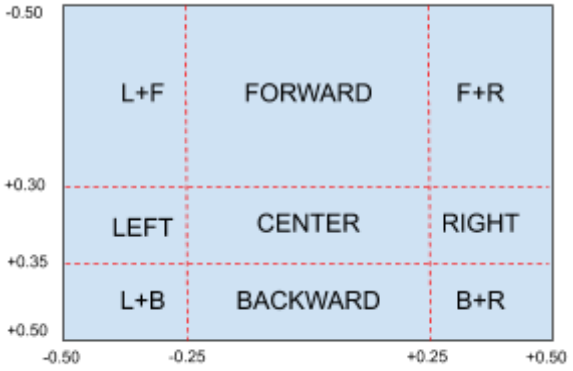


Figure 28: Direction depending on ball location

- **Publish outcome into topic:** Associate each direction with a velocity. This process is similar to the manual mode. For example, the ball could be in the top right corner. In this scenario, the components of the “vel” instance are “forward” and “right”. This translates into a positive linear velocity and a positive angular velocity value. The class instance “vel” is then published in the “cmd_vel” topic.

6.3.2.3. Servo Ball Following

The servo ball following mode works as the previous mode. It uses the servos that hold the camera to face the ball, having it always at the center of the image, so instead of using the location to move the wheels, the servos are moved. Basic instances are shown in FIgure 29.

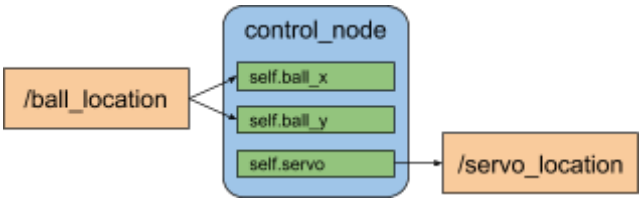


Figure 29: Instances used in ball_following_servo mode

This node is structured as follows:

- **Normalize ball location**. Exact same process as the ball_following_drive mode explained in Section 6.3.2.2.

- **Assign different servo movements to different ball positions.** Depending on the normalized coordinates of the ball (x, y), the location of the servos is set. Figure 30 shows the division constrictions.. Limits (-0.5,+0.5) are normalized for any image resolution. The words inside the boxes refer to the direction that the servo has to move depending on the location of the ball. The purpose of the code is to maintain the ball at the “center” of the image. More detailed explanations on GitHub [30].

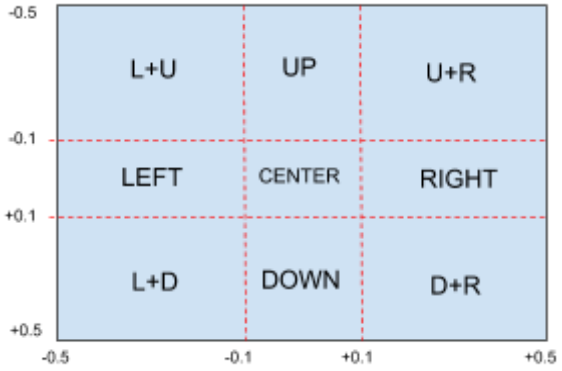


Figure 30: Direction depending on ball location

- **Publish outcome into topic:** Associate each direction with a position. This process is similar to the ball_following_drive mode. For example, the ball could be in the bottom left corner. In this scenario, the components of the “servo” instance are “down” and “left”. This translates into a negative position for the servo moving the x-axis and a negative position for the servo moving the y-axis of the camera. The class instance “servo” is then published in the “servo_location” topic.

6.4. Output chain

The output chain is necessary to convert the orders from the control node into data that the robot actuators understand.

6.4.1. Drive node

This node is in charge of transforming the velocity published in the “cmd_vel” topic into the PWM signal sent to the wheels. The structure of this output node is portrayed in Figure 31. It first subscribes to “cmd_vel” where the velocity is published and then it converts the magnitude into a PWM signal for each wheel.

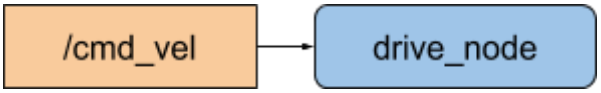


Figure 31: ROS drive node structure

The process will be explored in the following paragraphs.

- **Subscribe to topic**. The first thing to be done is to subscribe to the topic that contains the velocity information. As mentioned above, velocity comes in a Twist object. It contains linear and angular speed. This information is being published in the “cmd_vel” topic. The drive node subscribes to this topic.

- **Transform velocity into PWM signal**. Secondly, the node transforms this angular and linear velocity into a PWM signal for each wheel. The total velocity of each wheel depends on the linear velocity ±angular velocity, multiplied by a factor of 50 to magnify the signal. This process is shown in Figure 32. Linear velocity in green and angular velocity in red. The outcome of the two is the PWM signal for each wheel.

Figure 32: Relation between angular and linear velocity with wheel rotation

6.4.2. Servo node

This node is in charge of moving the servo motors that hold the camera. There are two servo motors: the bottom one moves horizontally and the upper moves vertically. The structure of the servo node is shown in Figure 33.

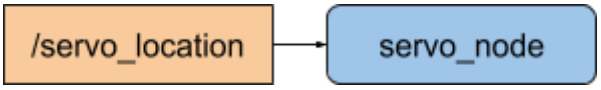


Figure 33: ROS servo node structure

The process is explored in the following paragraphs:

- **Subscribe to topic**. Firstly, the node is subscribed to the topic in which the control node publishes as shown in Figure 33. The data published in the topic comes from the control node. Once it is published on the topic, the servo node receives the values.

- **Transform velocity into PWM signal.** Secondly, the values are interpreted by the node. The data received consists of two values between 500 and 2500 that represent a PWM signal. The values are interpreted by the lower and upper servo motors as shown in the GitHub repository [30].

7. Gazebo Simulation

7.1. A Brief Explanation of Gazebo

Gazebo is a 3D dynamic simulator (integrated with ROS) with the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. As some game engines, Gazebo offers physics simulation at a high degree of fidelity, a suite of sensors and interfaces for both users and programs. Usually, Gazebo is used to test the algorithms created and the design of robots and test the result in a very realistic scenario, so it is widely used mostly because of its physics engines and libraries of robots models, environments and sensors [32].

The choice of Gazebo over Rviz, another extensively used 3D visualizer, is that Rviz only simulates the robot and the data, while Gazebo also offers the robot and its environment visualization. For this reason, it will be the simulator elected, since for example, being able to actually see what the camera is streaming on the simulation is necessary to this project.

Finally, the code where all the elements of the simulation, respecting the robot, are included is called URDF. The Universal Robotic Description Format (URDF) is an XML file format used in ROS to describe all elements of a robot.

7.2. How Gazebo works

In order to explain how to create a simulation of a robot in Gazebo from zero, a simple robot called “example” is developed now as an instance. Next to the code there will be some comments in the standard XML format, highlighted in blue for ease of recognition.

This chapter will be divided into two subchapters: The first one will consist of the requirements necessary after building the robot, such as the creation of the packages and folders according to the ROS standards, as well as the launch and the world files. The second chapter will cover the actual URDF code of the robot.

The actual simulation code will be displayed in the repository on GitHub [33]

7.2.1. Requirements

As mentioned before, this chapter will be presenting the structure that the Gazebo packages should have in ROS and the launch and world files.

The simulation packages will be built in the src of a workspace, new or not. For further details about building a workspace and a package, refer to chapter 5.3. Building ROS workspaces and packages. Inside /src, there are two packages: /MYROBOT_gazebo, covering world and launch files, and /MYROBOT_description, including the model and the robot description. “MYROBOT” will be replaced with the name of the robot, in lowercase letters. The structure followed is displayed in Figure 34.

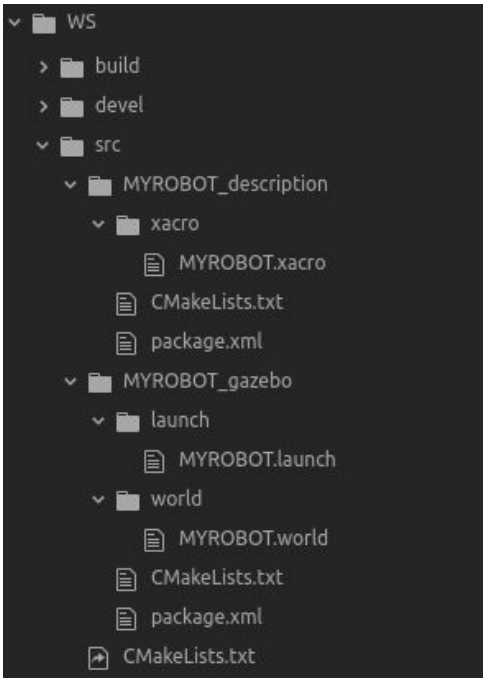


Figure 34: Structure of the simulation workspace

Inside the package /MYROBOT_gazebo, apart from the CMakeList.txt and the package.xml, the folders /launch and /world will be created, just to organize the files.

Inside launch, the launch file will be placed. It will be named according to the standard MYROBOT.launch. To convert the URDF file from xacro to XML format, first the package has to be installed, just writing the following command in the Ubuntu terminal:

```
sudo apt-get install ros-jade-pr2-common
```

Below in Figure 35 the launch file is explained.

```
<launch>

  <include file="$(find gazebo_ros)/launch/empty_world.launch"> <!-- First it is included a simple
     empty world that Gazebo provides -->
    <arg name="world_name" value="$(find example_gazebo)/worlds/example.world"/> <!-- After the plain
    world is included, the actual world  example.world is called, which can be found inside the
    example_gazebo package -->
    <!-- More default parameters can be changed here -->
  </include>

  <!-- With the following command the URDF file that will be created in xacro format can be modified to be
    in XML format -->
  <param name="robot_description"
    command="$(find xacro)/xacro.py $(find example_description)/xacro/example.xacro" />

  <!-- Lastly, the robot can be spawned in Gazebo, executing the "spawn_urdf" node and setting the
  arguments always the same -->
  <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
    args="-param robot_description -urdf -model example" />

</launch>
```

Figure 35: Launch file for example robot

Once created the launch file, the world will be explained. As this is just an example and the robot will not require any environment, the world will only consist of a ground plane and a sun as a light. As explained before, the file MYROBOT.launch will be placed in world, inside /MYROBOT_gazebo. The code is explained in Figure 36.

```
<?xml version="1.0" ?>
<sdf version="1.4"> <!-- First is alawys neccesary to include these two paragraphs in order to know
    which kind of document this is -->
  <world name="default">
    <include>
      <uri>model://ground_plane</uri> <!-- A ground plane is included here, models that gazebo already
        includes -->
    </include>
    <include>
      <uri>model://sun</uri> <!-- Here a light is also included, in this case a sun type -->
    </include>
  </world>
</sdf>
```

Figure 36: World file for the robot example

Finally, the package /MYROBOT_description has to be created. As said before, inside it there will be placed, apart from CMakelist.txt and package.xml, a folder called xacro. In xacro there will be the URDF MYROBOT.xacro. It will be developed in the next subchapter.

7.2.2. Robot example code

This robot, called “Example”, will consist of a very basic box with two mobile wheels. Next to the code itself there are some comments in light blue with the notation <!-- *comment* -->, to help understand what each parameter is.
Firstly, the body of the robot called “Chassis” is written in Figure 37.

```
<?xml version="1.0" ?>
<robot name="example" xmlns:xacro="http://www.ros.org/wiki/xacro">
<!-- Links: -->
  <!-- Chassis link: -->
  <link name="Chassis">
    <pose>0 0 0.1 0 0 0</pose> <!-- The coordinates are: x y z roll pitch yawn. The pose provides the
      position of the link, always related to the origin of the robot we are creating -->
    <inertial> <!-- This comprises the mass of the robot and how is distributed -->
      <mass value="2.5"/> <!-- In kilograms -->
      <origin rpy="0 0 0" xyz="0 0 0.8"/> <!-- This is the center of mass -->
      <inertia ixx="0.0395416666667" ixy="0" ixz="0" iyy="0.106208333333" iyz="0" izz="0.106208333333"/>
        <!-- How the mass is distributed around the shape of the link -->
    </inertial>
    <collision name="Chassis_Collision"> <!-- Part used by the physicss simulator to calculate collisions,
        including the dimensions of the link -->
      <geometry>
        <box size="0.3 0.15 0.07"/> <!-- In this case is a box, which is easier for the computer to
          compute -->
      </geometry>
    </collision>
    <visual name="Chassis_Visual"> <!-- Here it is described how the link is going to be represented -->
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <box size="0.5 0.3 0.07"/>
      </geometry>
      <material name="blue"/> <!-- This is only for Rviz visualization -->
    </visual>
  </link>
```

Figure 37: File describing the body of the robot, a box

All these parameters should be known after building the robot, like the size or weight. However, the inertia matrix is not easily calculated. Gazebo has a tutorial [34], which has been followed to find these values.

After that, the left and right wheels are described in Figure 38 and Figure 39, respectively. As can be seen they are exactly the same. The difference will come with where are they placed, but their model is essentially the same.

```
<!-- Left wheel link -->
<link name="Left_Wheel">
  <inertial>
    <mass value="0.2"/>
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
    <inertia ixx="0.000526666666667" ixy="0" ixz="0" iyy="0.000526666666667" iyz="0" izz="0.001"/>
  </inertial>
  <collision name="Left_Wheel_Collision">
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
    <geometry>
        <cylinder length="0.04" radius="0.065"/>
    </geometry>
  </collision>
  <visual name="Left_Wheel_Visual">
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
    <geometry>
      <cylinder length="0.04" radius="0.1"/>
    </geometry>
  </visual>
</link>
```

Figure 38: Left wheel link

```
<!-- Right wheel link -->
<link name="Right_Wheel">
  <inertial>
    <mass value="0.2"/>
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
    <inertia ixx="0.000526666666667" ixy="0" ixz="0" iyy="0.000526666666667" iyz="0" izz="0.001"/>
  </inertial>
  <collision name="Right_Wheel_Collision">
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
      <geometry>
        <cylinder length="0.04" radius="0.065"/>
      </geometry>
  </collision>
  <visual name="Right_Wheel_Visual">
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
      <geometry>
        <cylinder length="0.04" radius="0.1"/>
      </geometry>
  </visual>
</link>
```

Figure 39: Right wheel link

The last part of the robot is the joints, that is, the element that describes the dynamics, kinematics and the safety limits of the articulation. Every connection between two links is managed by a joint, in which one is the “parent” and the other the “child”, as it can be seen in Figure 40 from the ROS wiki [35].

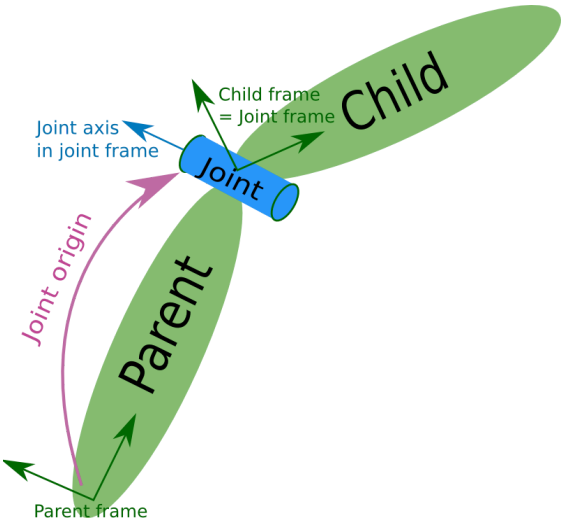


Figure 40: Scheme of a joint [35]

The movement possible between the two links depends on the type of joint: Revolute, continuous, prismatic, fixed, floating or planar. The most used ones are continuous (the one that will be used in this example) and fixed. For further details refer to the ROS wiki [35]. Below in Figure 41 the code for the joints is displayed.

```
  <!-- Joint between right wheel and chassis -->
  <joint name="Joint_Right_Wheel" type="continuous"> <!-- The type of the joint is continuous as there
      will be rotation in the Y axis  -->
    <origin rpy="0 0 0" xyz="-0.05 0.15 0"/> <!-- This is where, in the parent link, is the connection -->
      <child link="Right_Wheel"/> <!-- The name of the minor link, that will go attached to the main one -->
      <parent link="Chassis"/> <!-- Tha name of the main link, in this case, the chassis of the robot -->
      <axis rpy="0 0 0" xyz="0 1 0"/> <!-- This gives the infomation about the axis where the rotation is
        permited, in this case the Y axis -->
      <limit effort="10000" velocity="1000"/> <!-- An attribute for enforcing the maximum joint effort
        (|applied effort| < |effort|) and velocity -->
      <joint_properties damping="1.0" friction="1.0"/>
  </joint>

  <!-- Joint between right wheel and chassis -->
  <joint name="Joint_Left_Wheel" type="continuous">
    <origin rpy="0 0 0" xyz="-0.05 -0.15 0"/> <!-- PAY ATTENTION NEGATIVE 'y' -->
    <child link="Left_Wheel"/>
    <parent link="Chassis"/>
    <axis rpy="0 0 0" xyz="0 1 0"/>
    <limit effort="10000" velocity="1000"/>
    <joint_properties damping="1.0" friction="1.0"/>
  </joint>
</robot>
```

Figure 41: Right wheel-chassis and left wheel-chassis joints

Finally, in order to run the simulation the following commands have to be executed inside the WS:

```
source devel/setup.bash
roslaunch MYROBOT_gazebo MYROBOT.launch
```

7.3. Simulation

In this chapter the actual simulation is presented. Figure 42 shows the ROS simulation structure, a modified version of the ROS structure to perform the task. Instead, the simulation has an additional node called “gazebo”. This node provides the input image to the input chain and receives the output data from the output chain.

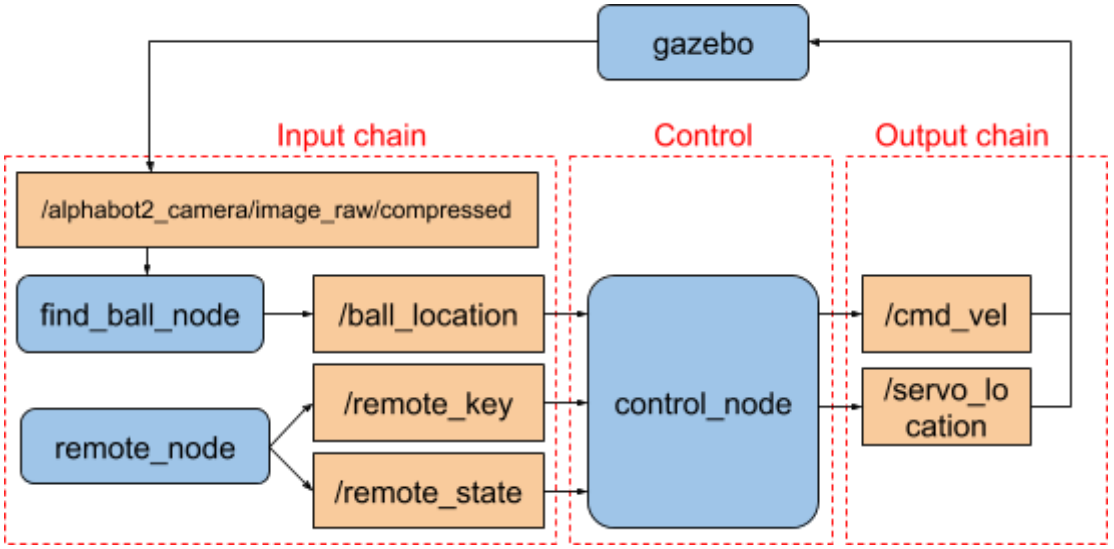


Figure 42: Gazebo Simulation Structure

7.3.1. Simulation Installation

Previous to installing the simulation the Linux computer must have Gazebo and ROS installed. The code for the simulation can be found in the GitHub repository [33]. The following paragraphs contain the commands needed and a brief explanation.

```
mkdir alphabot2pi_simulation_ws

cd alphabot2pi_simulation_ws

git clone https://github.com/nestoregon/alphabot2pi_simulation/
```

This is done in order to create a workspace in which the user can run the simulation. The code is then downloaded from the repository.

```
mv alphabot2pi_simulation src

source devel/setup.bash

cd src/alphabot2_simulation/sim_control/src
```

```
chmod  +x control_robot_node.py find_ball_blue_node.py servo_node.py
drive_node.py remote_node.py
```

The code is downloaded and set as the “src” file of the workspace. Refer to Section 5.3 for further information about ROS workspaces. The second command sources the workspace, allowing ROS to find its packages. The third command moves to the directory where the executable code is located. Lastly, the python code is made executable.

7.3.2. Running the code

Multiple terminal windows have to be open in order to run the simulation. Run the following commands in separate windows.

```
roslaunch alphabot2_world spawn_world.launch

roslaunch alphabot2_world spawn_robot.launch

roslaunch sim_control computer.launch
```

The first line launched the world, the second line spawned the robot and the last command launched all the nodes necessary to process the information (input chain, control and output chain). This last window has to be selected in order to move the robot using the keys “a, w, s, d” for the direction and “1, 2” for selecting the tasks. Number 1 is pressed for manual mode and number 2 for ball following drive.

Lastly, the blue ball needs to be added to the simulation. This ball has to be of 220 degrees (Hue value in the HSV scale). Such range represents the blue color that is needed in order for the “find_ball_node” to locate the ball. Figure 43 shows the complete simulation after following each step.

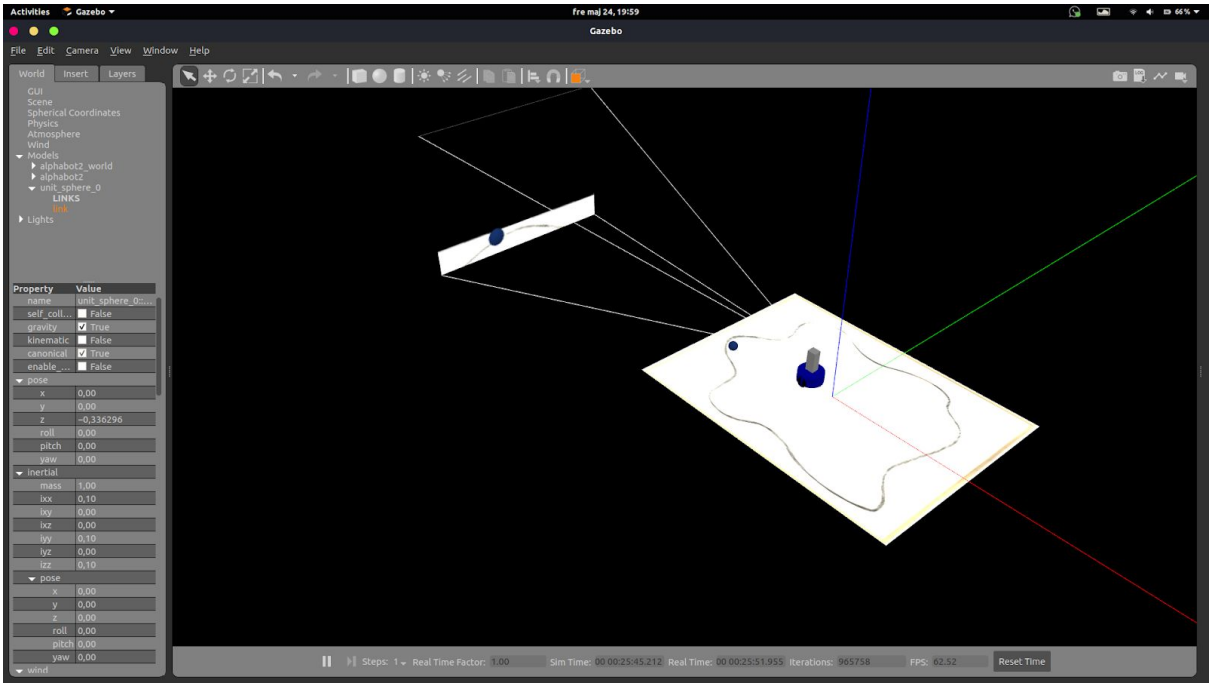


Figure 43: Alphabot2pi Gazebo Simulation

8. Discussion

Along the process, the authors have encountered different problems and possible improvements. One of the difficulties of this project is that the task was not clearly defined from the beginning, because it greatly depended on the time left, once the software was learnt. This implies that the authors had to make many decisions that would affect the project without sufficient knowledge. After months of work, the authors realized that some decisions from the commencement should have been different. In the following paragraphs, there is a brief explanation of decisions made and what other options could have been selected. In addition, to the Improvements and Problems chapters, a Future Work section is also displayed.

8.1. Improvements

8.1.1. Robot kit

After all the criteria was explained and the robot was finally chosen, the supervisor of this thesis bought it without second thoughts. It performed well all the tasks it was supposed to do, with a good accomplishment of the DC motors of the wheels, the servomotors of the camera and the camera itself. However, its low price made some limitations for the robot, such as:

- **Battery:** The goal established was to build an autonomous mobile robot, but the focus never was on its energy lifetime, as the time it can work without being connected to the electricity. AlphaBot2 can work both connected to electricity or with two AA batteries but if instead of those it would have had a bigger lithium-ion battery,

for example, its working time without being connected would be much higher. In this case, the duration of the batteries is about 30 minutes. For this reason, another criteria that needs to be taken into account before selecting the robot kit is its energy lifetime.

- **Range of the camera:** One of the most useful sensors for this thesis is the camera. Once it was able to stream the video via wifi to the computer, the problem of its field of view arose. As the task was to find the ball, with a higher range (especially the horizontal range) the robot would not have to turn so much to localize the ball. The horizontal range was measured with a result of 30º approximately. For example, the official Raspberry Pi V2 camera has an horizontal field of view of 62.2º [36].

- **Modularity:** The robot AlphaBot2 purchased is not modular at all, meaning that no add-ons can be placed. For example, if there is a need to expand the current project by adding a laser distance sensor, it will not be possible unless the original motherboard is modified, which can result really complicated. For this reason, modularity is another aspect to look into account before selecting the robot kit.

8.1.2. Software

The robotic middleware chosen was ROS. Also, as Gazebo was already implemented and it could simulate both robots and environments, it was the program chosen to build the simulation. This decision was based on different advantages, such as being open-source, supporting different programming languages and multitasking control (Section 5.1).

Although both ROS and Gazebo have proven to be useful there were some major drawbacks:

- **Long learning time**. The structure of ROS is complex and difficult to learn. This means that the authors spent a huge amount of time learning it instead of developing the tasks. Learning ROS has been a continuous process. At the beginning of the project, it took four weeks to learn the basics of ROS.
  It is crucial to have a solid understanding of its working process given that mistakes arise. A problem that could have been easily solved in minutes takes hours, due to difficulties finding the root cause of the problem. The authors have experienced multiple problems that have slowed down the process. An easier middleware like Matlab could have been used for the control of the robot, being simpler and easier to learn.

- **Code performance:** Most of the code used was adapted into ROS. The ball following mode (Chapter 6.3.2.2) could be improved for better performance, taking into account delays in the system. Also, the simulation could have a better mass distribution, so the model does not tilt while it is going forward or backwards.

Given these arguments, a possible replacement for ROS could have been Matlab. With help from its Robotics System Toolbox and image recognition libraries, Matlab is an excellent replacement. Future analysis could be performed in order to identify the best software for autonomous robotic platform development. It is also possible for both system to coexist, Matlab being an independent node connected to ROS.

8.2. Problems

8.2.1. Robot Performance

- **Light conditions:** The task finally chosen to be performed by the robot was to follow a blue ball using the camera. To do so, as explained before, a range of blues had to be settled. The problem was that this range of blues changed from daylight to night, as the light is not the same varying from cold to warm. In order for the ball to be recognized under any conditions, the range of blues was enlarged but it brought another problem: the robot confused the ball with other objects in the room.

- **Internet connection:** All the image processing was carried out by the computer, so the Raspberry Pi had to stream the video from the camera to the computer to be processed, and then receive the location of the ball. Because of that, there was a great depence on the Internet. If it was slow, once the robot received the location, the ball was in a different place so the robot performance was not adequate.

- **Assembly:** The construction of the AlphaBot2 was carried out following the instruction manual that its manufacturer company, WaveShare, provides. However, once the servo motors that move the camera were going to be implemented, they did not work properly. After a week trying to look for the error in the code, the authors realised that it was an assembly error that is not specifically specified in the instructions manual, so it can be confusing. The error was that the wire was connected backwards, with the brown wire connected with S0 and S1 instead of with the GND port, as it can be seen in Figure 44.

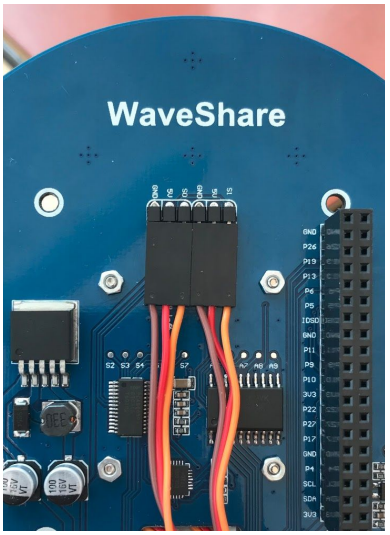


Figure 44: Wires of the servo motors

8.2.2. ROS Performance

- **Multitasking control delays**. Multitasking has been a great tool to develop and try programmes separately. Furthermore, being modular implies that nodes can be launched and connected to one another. For instance, the camera node (Section 6.2.1), can be launched independently.
  However, multitasking has also major drawbacks. Communicating nodes over topics takes time. This is mainly because the first node has to publish on a topic and the second node has to receive the information from the topic [37]. Additionally, the more nodes a system has, the longer the delay. This is especially important for image detection, given that the robot has to be able to detect a ball instantly to correct its position. Delays in the system equal to poor performance.

- **Debug:** ROS does not provide directly a debugging tool for the code. It does have several for testing the communication within its structure, for example roswtf. Apart from it, the possible errors in the Python or C++ code can not be detected automatically, which makes the debugging process really tedious.

8.3. Future Work

This project has enormous potential for growth. Most of the time spent on the project was spent on developing a strong understanding of ROS and implement a simple task. Once this goal has been achieved, there are endless possibilities. These possibilities include:

- **Improve camera nodes:** The basic camera node is used to detect a blue ball. Many more camera nodes can be implemented simultaneously to detect, for example, different colours. More tasks could be developed if further objects are detected. For instance, the robot could detect a goal and try to push the ball inside. Therefore, these camera nodes will uplift the robot's overall performance.

- **Creation of new functionalities:** In this project the authors have examined three tasks using wheels, camera and servos. However, the AlphaBot2 Pi can offer much more. As inputs, it has line following and obstacle avoidance sensors and as outputs it has leds and a buzzer. With these components, the possible expansions of the current project are almost limitless.

- **Video tutorials:** The hardest part of this project is understanding how the robot works. Developing quality teaching material that ensures an easy learning process is a priority. Doing so will ensure that students effectively understand the content.

- **Matlab implementation:** Investigation on the use of Matlab as a middleware for autonomous robots development (instead of ROS) or image recognition (instead of OpenCV). Try the different alternatives offered by Matlab. This middleware will also

have benefits and drawbacks. It is up tor future studies to decide which software is best fitted for autonomous robots.

9. Conclusion

9.1. Goals achieved

In this chapter the authors ponder on the results obtained and how were those achieved. The thesis had three main goals which will be explored in the following paragraphs.

The first goal established was to select the most suitable platform and robot kit for this project. The performance of the Raspberry Pi combined with the AlphaBot2 was adequate, taking into account their low price and limitations. Those limitations are explained further in section 8.2.2. Robot kit. Still, the price is the most appealing characteristic of both Raspberry Pi and Alphabot2, in a thesis with educational purposes the one presented.

Secondly, the robot had to be built in order to perform the task. This task was not defined, it was up to the authors to find a suitable function. The first idea was to develop a task related to ball detection. It included the camera and the wheels. After this task was completed, the authors implemented the use of a remote controller to select modes, giving more flexibility to the project and making it possible to involve more parts of the robot. Furthermore, it gives the authors the possibility of developing more code rather than focusing on one specific task. It allows future students to develop their own tasks and modes. These additional modes were developed using the code provided by the manufacturer company. The code was adapted to be able to support ROS. Sometimes this part involved errors because the authors were unable to implement each function. It often lead to dead ends, time was wasted in attempts to adapt, for example, the LEDs implementation in ROS.

Nodes that required more computational power such as “find_ball_node” (Section 6.2.2) and “control node” (Section 6.3.1) were performed in a different computer. The rest of the nodes that required less computational power were performed in the Raspberry Pi. ROS future to connect computer was tremendously useful to be able to work with up to three computers at the same time.

A limitation of this project has always been the time. The tasks were being developed until there was no more time to continue with its development. Had the authors more time, it would have been spent perfecting the robot’s task. Once the deadline approached, the authors had developed three tasks: “manual”, “ball_following_drive” and “ball_following_servo”. As stated in Future Work (Section 8.2), the code could be optimised and further functions could be implemented.

Drawbacks, as explained in Section 8.1, were related to difficulties learning the middleware. Mistakes were made and it was extremely difficult to solve the problem due to the authors’ lack of understanding of ROS. Another problem related to the “find_ball” node was the delays of the program. A more sophisticated solution could be developed if these delays are taken into account.

Lastly, the final task was to implement a simulation in Gazebo to test the code. The AlphaBot2 model was built and then the code implemented. In the simulation, the robot can move forward, backwards and also turn following keyboard commands. In addition, the camera streams what the robot faces in the simulation, and it follows a blue ball and keep a certain distance, as well as the real one.

10. Bibliography

1. Capek K. Rossum's Universal Robots. 1 ed. Prague: Aventinum; 1921 [cited 2019 Mar 1]

2. Asimov I. I, Robot. 1 ed. New York: Gnome Press; 1951. [cited 2019 Mar 1]

3. Oxford Dictionaries. Robot | Definition of Robot in English by Oxford Dictionaries [Internet]. [cited 2019 Mar 1]. Available from: https://en.oxforddictionaries.com/definition/robot

4. Devol G, inventor. Programmed Article Transfer. American patent US 2988237A. 1954, Dec 10.

5. Kurfess, T. (Ed.), Nagarkatti, S., Zefran, M., Goodwine, J., Raanes, C., Falcon, J., et al. Robotics and Automation Handbook. 1 ed. Boca Raton: CRC Press; 1970.

6. IFR forecast: 1.7 million new robots to transform the world´s factories by 2020 - International Federation of Robotics [Internet]. [cited 2019 Apr 23]. Available from: https://ifr.org/ifr-press-releases/news/ifr-forecast-1.7-million-new-robots-to-transform-the-worlds-factories-by-20

7. Statista. Size of the global market for industrial and non-industrial robots between 2017 and 2025 (in billion U.S. dollars). https://www.statista.com/statistics/760190/worldwide-robotics-market-revenue/ : Statista; 2018.

8. Nations U. Resolution adopted by the General Assembly on 16 September 2005 60/1. 2005 World Summit Outcome [Internet]. 2005 [cited 2019 Apr 23]. Available from: https://www.un.org/en/development/desa/population/migration/generalassembly/docs/globalcompact/A_RES_60_1.pdf

9. Benevides, Chris. Automated guided vehicle [photography]. 2016 [cited 2019 Mar 5]. Available from: https://www.conveyco.com/advantages-disadvantages-automated-guided-vehicles-agvs/

10. Seewald AK. Mobile robotics - A practical introduction, 2nd edition, by Ulrich Nehmzow. Appl Artif Intell. 2004.

11. NASA, An artist drew this picture of a rover exploring the surface of Mars [photography]. 2015 [cited 2019 Mar 5] Available from: https://www.nasa.gov/audience/forstudents/k-4/dictionary/Rover.html

12. NASA, Definition of Rover [Internet]. 2015 [cited 2019 Mar 5] Available from: https://www.nasa.gov/audience/forstudents/k-4/dictionary/Rover.html

13. Oxford Dictionaries. Intelligence | Definition of Intelligence in English by Oxford Dictionaries [Internet]. [cited 2019 Mar 1]. Available from: https://en.oxforddictionaries.com/definition/intelligence

14. Douglas J. Chess 4.7 versus David Levy. BYTE. 1978; 3(12): page 84.

15. Everett HR. Sensors for Mobile Robots: Theory and Application. CRC Press. 1996;

16. Siegwart R, Nourbakhsh IR, Scaramuzza D. Introduction to Autonomous Mobile Robots. 2 ed. MIT Press; 2011.

17. Patle BK, Babu L G, Pandey A, Parhi DRK, Jagadeesh A. A review: On path planning strategies for navigation of mobile robot. Def Technol; 2019.

18. CollabNet VersionOne. VersionOne 12th Annual State of Agile Report [Internet]. 2018. [cited 2019 Apr 24]. Available from: https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report

19. Raspberry Pi. Raspberry Pi 3 Model B+ [Internet]. [cited 2019 May 8]. Available from: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

20. Bermúdez A, Gaztelumendi J. Building and programming an autonomous robot using Raspberry Pi as a PLC. [Bachelor Degree Project on the Internet]. Skövde: University of Skövde; 2016 [cited 2019 Mar 1]. Available from: http://www.diva-portal.org/smash/get/diva2:952952/FULLTEXT01.pdf

21. Arduino. Arduino Uno Rev3 [Internet]. [cited 2019 Mar 5]. Available from: https://store.arduino.cc/arduino-uno-rev3

22. LEGO. LEGO Mindstorms EV3 [Internet]. [cited 2019 Mar 5]. Available from: https://shop.lego.com/en-ES/product/LEGO-MINDSTORMS-EV3-31313

23. ASUS. Tinker Board [Internet]. [cited 2019 Mar 5]. Available from: https://www.asus.com/us/Single-Board-Computer/Tinker-Board/

24. The Construct. [Morpheus Chair] Build the Structure of a ROS + Raspberry Pi Robot | Ep.1 [Video file]. 2019, Jan 29 [cited 2019 Apr 25]. Available from: https://www.youtube.com/watch?v=TABVZf5vKVA

25. Igelmo V. USING A GENERAL ROBOT PROGRAMMING SYSTEM TO CONTROL AN INDUSTRIAL ROBOT [Bachelor Degree Project on the Internet]. Skövde: University of Skövde; 2018 [cited 2019 Apr 26]. Available from: http://www.diva-portal.org/smash/get/diva2:1221562/FULLTEXT01.pdf

26. ROS Wiki. ROS/Concepts [Internet]. [cited 2019 Apr 26]. Available from: http://wiki.ros.org/ROS/Concepts

27. Joseph L. Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using Robot Operating System and master its out-of-the-box functionalities. 1 ed. Birmingham, United Kingdom: Packt Publishing Ltd.; 2015.

28. Lum J. Utilizing Robot Operating System (ROS) in robot vision and control [Master´s thesis on the Internet]. Monterrey: Naval Postgraduate School; 2015 [cited 2019 Apr 28]. Available from: https://calhoun.nps.edu/handle/10945/47300

29. ROS Wiki. catkin/workspaces [Internet]. [cited 2019 Apr 30]. Available from: http://wiki.ros.org/catkin/workspaces

30. GitHub. Alphabot2pi [Internet]. [cited 2019 Jun 1]. Available from: https://github.com/nestoregon/alphabot2pi_real

31. Yadav H., Srivastava S., Mukherjee P., Lall B. A real-time ball trajectory follower using Robot Operating System. Third International Conference on Image Information Processing (ICIIP). 2015 pp. 511-515. DOI: 10.1109/ICIIP.2015.7414826

32. Gazebo. Tutorial : Beginner: Overview [Internet]. [cited 2019 Mar 1]. Available from: http://gazebosim.org/tutorials?tut=guided_b1&cat=

33. GitHub. Alphabot2pi_simulation [Internet]. [cited 2019 Jun 1]. Available from: https://github.com/nestoregon/alphabot2pi_simulation

34. Gazebo. Tutorial : Inertial parameters of triangle meshes [Internet]. [cited 2019 Apr 25]. Available from: http://gazebosim.org/tutorials?tut=inertia&cat=build_robot

35. ROS Wiki. urdf/XML/joint [Internet]. [cited 2019 May 2]. Available from: http://wiki.ros.org/urdf/XML/joint

36. Raspberry Pi. Camera module [Internet]. [cited 2019 May 16]. Available from: https://www.raspberrypi.org/documentation/hardware/camera/

37. Forouher D, Hartmann J and Maehle E. Data Flow Analysis in ROS. ISR/Robotik 2014. 2014; pp. 1-6.

11. Appendix I: Purchase links

Turtlebot 3 Burger:
http://www.robotis.us/turtlebot-3-burger-us/

UCTRONICS Smart Robot Car Kit for Raspberry Pi:
https://bit.ly/2Uqt60A

WaveShare Alphabot2 Pi:
https://www.waveshare.com/product/robotics/alphabot2/kits/alphabot2-pi3-b-plus.htm

Self-Built:

- Magician Chassis: https://amzn.to/2VDf7RE
- Raspberry Pi 3 Model B: https://amzn.to/2IeMUgN
- 16GB Class 10 MicroSD Card: https://amzn.to/2Z3jgAu
- Raspberry Pi Lipo Battery Power Pack Board: https://amzn.to/2IcqlcM
- Lipo Battery for the Powerboost: https://amzn.to/2WWhcbB
- Raspberry Pi Motor Driver Expansion DC-Stepper Motor Driver: https://amzn.to/2KrU3fR
- Raspberry Pi Camera Module: https://amzn.to/2InEO4M
- Cable Micro USB: https://amzn.to/2VzNXLh
- Adapter to USB: https://amzn.to/2I8I052
- H Bridge Motor Driver Hats: https://amzn.to/2KrU3fR
- Jumper Wires: https://amzn.to/2UcHy7k
- Jack Connectors: https://amzn.to/2UTk1wy
- AA Batteries: https://amzn.to/2UtkeqZ
- Bluetooth keyboard: https://amzn.to/2VCODQf