POLITECNICO DI MILANO

School of Industrial and Information Engineering
Master of Science in Automation and Control Engineering

BUILDING AND CONTROLLING A TRACKING ROBOT FOR ROBOTGAMES

AI & R Lab
Laboratory of Artificial Intelligence
and Robotics of Politecnico di Milano

Relatore: Prof. Andrea Bonarini

Tesi di Laurea Magistrale di:
Juan Luis Guillén Ramírez 877243

Academic Year 2017 - 2018

To my loved ones

ABSTRACT

Humans and robots had been sharing some areas of the society for decades and this coexistence is expanding to diverse daily applications. The number of interactions between people and autonomous machines are increasing and the need to implements recognition systems and, more importantly, social skills into robots is a reality. With the intention of dealing with these situations, the Human-Robot Interaction (HRI) field of research is targeting these collaborations concerns to guarantee safety.

Robotics is a constantly expanding field that evolves to define the future society. Robots are no longer exclusively at the industries, doing the work that no human can or should do for safety or comfort reasons. The human daily activities are including robots with many different applications and there are diverse topics of research concerning this changing situation.

One of the most promising field of study is robotgames, an area of robotics based on interactive games between humans and autonomous robots. By defining new robots and games, the possible applications are very diverse and stimulating, like using the robotgames as a therapy support for children with impairments or disabilities.

The aim of this thesis is to build from the design a small ground robot using inexpensive components. Once the hardware architecture is produced, the scope of this project it to create the necessary code on the Arduino IDE platform to be implemented and set the system as a tracking robot for robotgames. The building, programming and laboratory tests are carried out at the AirLab of Politecnico di Milano.

ACKNOWLEDGEMENTS

First of all, I would like to express my gratitude to my supervisor Professor Andrea Bonarini for his suggestions and advices during the process of this thesis. Without his guidance this thesis would not have been the same.

To all the friends who have been beside me along this academic period and made this experience more exciting. A special mention to Pablo, Sebastián and Jose Carlos for all the years of engineering and friendship through two countries.

I very special thanks to Marta for being there through thin and thick. Part of what I have accomplished during this time in Milan and personal development would not have been possible without her.

Last but not least, I would like to thank my family, specially my parents and sister, for their encouragement and constant belief in me. Their support and caring have always been there for the past quarter of a century. Without them I would not be the same.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF EQUATIONS

CHAPTER 1: INTRODUCTION

The robotic field has been expanding exponentially toward new applications of the state-of-the-art technologies and its interaction with the human life is becoming more and more common. A progressive integration in the industry and, lately, in quotidian places like schools, hospitals and private houses have been taking place during the past years, making this sector grow and explore present and future potential implementations.

At its origin, the word 'robot' was first applied almost one hundred years ago by a Czech playwright which used the term '*robota*' with the meaning of '*forced labour*'. It has to be admitted that this has been and still is the main function of most of the robots, to replace the human beings in dangerous, strength demanding or inaccessible situations, as well as making their tasks more comfortable. However, this has been changing and the robot-human coexistence leads to an investment in the human-robot interactions (HRI).

The study of these relations between androids and human beings has always been a key point for the development of new applications of robotics. New generation of autonomous robots are equipped with sensory systems with the intention of achieving a safer and more efficient tasks performance. Out of the vast improvement that has been made on this field, it has a strong potential the research area defined as Robotgames.

This area covers the development of interactive games involving people and autonomous robots. It is important the engagement of the player with the proposed game as well as the robot interaction with the environment and the player while performing its tasks. Apart from the pure enjoyment and pleasure of the game activity, an important objective of this topic is to offer play opportunities to children in general and as a help to develop intellectual functioning in children with disabilities.

The scope of this project was to design and create a robot which performs an interactive game. It was decided to produce a dedicated robot based on a previous model which was available at the AirLab of Politecnico di Milano. This new robot was intended to be a small ground mechanism, transportable, with autonomous moving and behaviour, and with the main purpose of acting as a player in a human tracking game.

This report shows the developed work to design, create and produce a tracking robot for robotgames. The first part of the thesis was to build the mechanical and electronic system that will perform the game. The intention was to limit the construction to cheap technology and materials to make it replicable. However, the main difference with respect to the model that was used as inspiration was the change of simple electric motors for a

pair of DC motors with encoders, which offer more possibilities in terms of trajectory and speed control.

Consequently, the second part of the project was to provide the robot with autonomy by programming and implementing a control code. This task was carried out by using the open source programming environment called Arduino. A platform like Arduino offers multiple options for future improvements of the robot. The code is explained and attached to this report for better understanding of the system performance.

The structure of this dissertation document tries to explain in a clear way the work that has been carried out during the time of this project.

In Chapter 2, it is briefly outlined the state of the art of the Human-Robot Interaction (HRI), the robotgames field of study and the Arduino environment. These sources are analysed and commented through the literature review that have been done.

In Chapter 3, it is explained the goals and motivation that set the beginning of this thesis. It presents and defines the problem to be solved and the reasons that made this work possible and, consequently, the methodology that was stablished and followed is described under this section.

In Chapter 4, the building process of the hardware architecture of the robot is explained. This englobes the design and decision development through the creation of the mechanisms, components description and its utilities for this project, and a step by step construction of the desired systems. Every approach that has been taken on this part of the project is reported and justified through this chapter.

In Chapter 5, following the previous section, the building of the robot is completed by explaining the software architecture. It is described the approaches to organise the robot behaviour and its tasks performance. Under the scope of this chapter it is involved the programming decisions, the controller creation and application, as well as a report commenting the main parts of the developed code.

Finally, in Chapter 6, the conclusions of the project are exposed. It also presents a critical analysis of the work that has been done in order to define the suggestions for future challenges that seems more appropriate to continue or improve the project.

The last part of the document is the Appendix, which includes the programming code that has been created and implemented in the robot.

CHAPTER 2:
STATE OF THE ART

This section is intended to show the exploration that has been done through the available literature related to the topic of this dissertation. It is a brief outline of all the material that has been researched concerning the different study areas which are involved in this project. It has to be mentioned that this should be taken as an initial previous knowledge that must be collected to achieve an appropriate understanding of the theme of concern.

2.1 – Human-Robot Interaction

The robotic field is a growing sector that is continuously evolving and moving forward in parallel with the technologies advances. The scenarios in which robotic is applied are changing from being exclusively applied in industrial areas to daily applications. This have activated the concern about introducing robots in human environments where the interaction between machines and humans is inevitable. Safety and common understanding became the key points for a successful integration of robots.

Human-Robot Interaction (HRI) is an area of research that focus its activity to understanding, designing and evaluating robotic systems to be used by or in collaboration with humans [1]. When referring to interaction, it is denoted the communication that should exists between a human and a robot. This communication can be physical or remote, and this factor is closely related to the safety issue that has been mentioned as a crucial concern.

At the literature of the early years of the past century, science fiction authors like Isaac Asimov were thinking about an optimistic view in which robots are benign and helpful beings which adhere to the Law of Robotics or code of non-violence against humans. Parallelly, after the robots were already introduced in the industry field, while the research of artificial intelligence became more and more complex, robots were created with a cognitive system to interact with and operate in the non-social environment. However, in [2] is suggested that social recognition and the so called ‘social skills’ should be the first thing to develop when developing an intelligent robot. In this article it can be found a large summary of the evolution of social intelligent robots, HRI studies and it develops social rules for robot behaviour.

As part of the survey that has been taken in [1], it is defined the HRI problem as “to understand and shape the interactions between one or more humans and one or more robots”. It has been mentioned before what is meant as interaction and it is perfectly clear how essential they are in every human-robot application. Consequently, the article

continues by referring how designers should understand the interaction that will occur at the concrete application and focus, when creating a robot or an application, in the five attributes that can affect the interactions: autonomy; information exchange; structure of the team participating in the interaction; adaptation, learning and training of humans and robot; and the shape of the designed task.

For the scope of this project, special care should be focused on physical interactions due to the intended tracking task. From the physical point of view, the main risk is on collisions between the robot and its user, so with the intention of increasing robot safety, it has to be considered at the design moment all the aspects as mechanics, electronics and software [3]. Every aspect of the robot creation should be considered as a potential issue for the HRI.

Lastly, it is important to include in this literature review the survey that has been done in [4]. The discussion about the context of these robots, current state of the art and the design methods and components used to build them, lead to a final description of their impact in the society.

2.2 – Robotgames

There are many studies on the literature as [5] which shows the benefits of playing for children development, including physical and psychological skills and the problem of people with disabilities to access the toys available on the market. It is reported how people with impairments and intellectual disabilities have a different capacity to respond to certain game activities and, based on the improvements that playing can make at childhood, it is defended the introduction of more toys targeted for this group of the children population.

The research topic of robotgames is part of the robotics field of study and it is based on interactive games with autonomous robots [6]. Many different applications are involved in this area, but a common shared attribute is that the robot has to interact with the human player while performing some action in the real environment in which is located.

There are currently in the market some toys that can fit in the mentioned topic of robotgames. More and more toys in the past years are progressively incorporating technology to make them more attractive to small children with a seed of science interest. However, most of them are not adapted to children with physical impairments (PI), i.e. problems in the body functions and structure. As part of the GIODI project (GIOco per la Disabilita – Play for disability), the document [7] evaluates the capacity of adaptation for children with PI of different robotic toys which are available on the market, as for example the one of Figure 1. The types of play are divided, according to the LUDI classification, in four categories: practice play, symbolic play, constructive play and rule play.

Figure 1. Zoomer, robotic toy available on the market [5]

Based on that classification, five toys are analysed in [7] and evaluated through the Test of Playfulness (ToP) [8]. The method and the results are shown and discussed in [9] after some play sessions with children. The overall conclusion was that the studied toys were successful but with the essential role of an adult. However, the issues were remarked and there were suggested some possible improvements that can be easily implemented to make these toys more accessible to children. The highest goal was to influence and cause an impact on toys producers.

Regarding the robotgames projects that have been developed in the AirLab of Politecnico di Milano, it deserves a special mention the robot Teo [10] [11], a mobile robot designed for children with Neuro-Developmental Disorder (NDD). It was built to act as a caregiver support and to interact and play with the children affected with this disease. It can be used for therapy-driven game-based activities as well as free play [11]. These characteristics offer a large amount of play combinations that make this robot a perfect example of the robotgames field.

Figure 2. Initial and final version of the robot Teo [11]

The promising results which were obtained from the play tests that were carried out with children demonstrate the importance, success and future of this topic of research.

2.3 - Arduino

One of the most common programming languages due to its open-source format is Arduino [12]. Arduino boards are embedded circuits with microcontrollers that can transform read inputs into desired outputs. The importance of being a free public platform is based on the easy access to knowledge and advices from the Arduino Community and Forums [13].

The programming software, Arduino IDE (Integrated Development Environment), uses the Arduino programming language which is based on the C and C++ languages. Referring to the official website [12], the advantages of this platform are:

- **Inexpensive**: making a comparison on the market, its boards are cheaper than the average microcontroller platforms.

- **Clear and understandable programming environment**: simple but flexible enough for advanced applications.

- **Open source software and hardware**: the huge amount of software and libraries, as well as the experts who are at the forums, make this language an interesting option. Moreover, its circuits boards can be freely modified and improved by users.

- **Cross-platform:** the programming environment can run on every operating system, which gives the demanded flexibility for coding.

CHAPTER 3:
PROJECT PRESENTATION

3.1 – Aim and Objectives

The core aim of this thesis project is to perform the building up and control of an autonomous robot for the dedicated task of tracking a moving object. The creation of the hardware body involved the mechanical and electronic areas to complete the connections between components. Furthermore, it was a main objective to write the code to be implemented into the embedded circuit and to provide the mechanism with certain autonomy and interaction.

The intention was to replicate a previous model with a similar hardware composition but in a poor state of conservation. Moreover, it was not built carefully enough to be a valid option for the control purpose of the thesis. With the new architecture, the objective was to get a functional robot that could be immediately programmed with a tracking task. For the creation of the robot, it was decided to keep a low-priced profile for the included technology, making the challenge of designing a suitable code with these elements.

Regarding the programming, the aim was to code the robot in the open-source platform Arduino, which is an interesting alternative for controlling embedded systems. Being a huge community of programmers, this platform offers a lot of online content and literature to start with any robotic project. It is also a clear and understandable language which would make the code easier to replicate in future improvements of the prototype.

In order to accomplish these purposes, the project goals were divided into differentiated phases that have organised the activity that needed to be done:

I. Revision and understanding of necessary literature to start this project. This involved the concepts of HRI, robotgames field and the knowledge of Arduino IDE.

II. Create a feasible hardware architecture based on the necessary mechanical and electronic components that are required for the task purpose. Selection of elements and integration in a single robot body.

III. Code the robot behaviour applying the logic of desired actions. Implement this code into the Arduino board and provide the robot with autonomy.

IV. Debug the code and the robot performance until obtaining a suitable first prototype of the intended robot for tracking robotgames.

3.2 – Motivation

The robotic field is constantly moving forward and innovating in diverse areas of application. This expansion has increased the interest of society to incorporate robot in many activities to improve the quality of the events. One of the most promising topics is the robotgame sector, in which mechanisms can help with the children integration and personal development, particularly if they have any physical impairment.

One purpose to follow with this project was to take a step back in technology related to robotics in order to take a step forward in replicability of the hardware and software, achieving a quick prototype which offers a suitable response. Nevertheless, being involved in all the stages of a robot creation set a big motivation to carry out the work reported in this document.

3.3 – Methodology

The thesis objective was divided in different stages that have been followed and completed during this project in order to achieve a successful outcome. The available set time for the project was distributed proportionally between the diverse planned tasks for the work. The organisation made the progressive improvements on the work.

Firstly, it was necessary to understand every single component that was ready to be incorporated in the robot. It was important to know their characteristics, working conditions, purposes and working process before mounting, stacking or wiring them in the architecture.

Then it was time to complete the building of the physical part of the robot, the electromechanical body. With the materials and tools of the AirLab plus the provided components, this stage was based on transforming the design ideas into a working mechanism with its locomotion, sensors and actuators systems. Additionally, as the last part of this phase, it was done the wiring and electronic connections between the Arduino board and the different elements.

The third big stage of this project was to program the embedded circuit to get a good combination of interpretations and responses from and to the environment. Based on the Arduino IDE platform, it was defined this part of the work as applying the gained open-source knowledge to create a new code to give a tracking performance and obstacle avoidance.

Finally, it was intended to leave a working first prototype of the robot that could be improved in the near future.

CHAPTER 4:
HARDWARE ARCHITECTURE

In this chapter it is described the process that has been followed to build up the robot under the scope of this thesis. Before starting with the software development, it was necessary to create the hardware system in which the code will be lately implemented. There cannot be a brain without the correspondent body. This mechatronic architecture was inspired by previous projects that were carried out at the AirLab of Politecnico di Milano.

The next sections contain the decision process for the design of the robot, focusing on the selected structures, the sensors and actuators that has been chosen and the reasons and purposes that leaded to these choices.

4.1 – Board

The core of the system is the microcontroller board which will be programmed with the code, constituting the main part of the electrical circuit and becoming the 'brain' of the robot. Once the programming language had been selected, being Arduino the chosen approach, the board options that have been considered were the Arduino Uno [14] and the Arduino Mega 2560. Regarding the project necessities and possible future improvements, the final decision was to use the enhanced Genuino product Arduino Mega 2560 [15].

Its name is taken from its integrated microcontroller, the ATmega2560, which is the supervisor of the board tasks and actions programmed in the embedded system. Due to the higher number of I/O pins (54 digital, 16 analogue) and larger memory than its competitor, the suitability of this board for the project under scope is proven. A general overview of the board can be seen in the following image [16]:

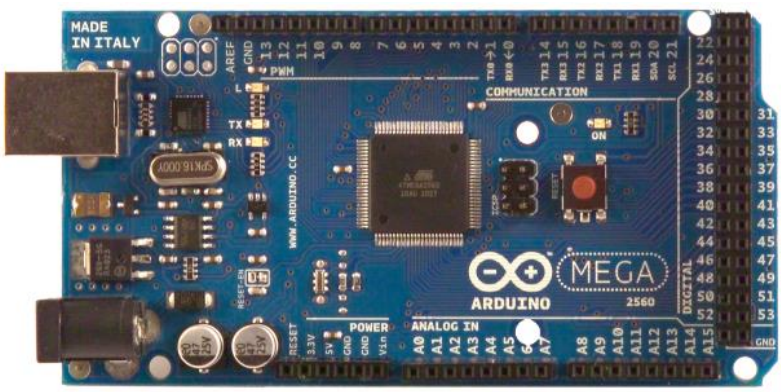

Figure 3. Arduino Mega2560 board overview [16]

It has been connected to the computer using the USB port and programmed with the Arduino Software. However, the board works offline on the robotgame and it only needs to be connected to the computer to upload new code or for debugging purpose because the board powering has been designed to be due to an external battery.

All the other system components are connected to the board through its pins (sensors, shield) and power jack plug (battery). These connections will be detailed in the following sections.

4.2 – Shield

The term shield in Arduino is related to additional boards that can be mounted on top of the main boards (in this project, Arduino Mega2560) and they contribute with extra capabilities. There is a large list of complementary shields available on the market but for the creation of this robot and based on the control needs, it has been chosen the 'Adafruit Motor Shield V1' [17] that is shown in the next figure:

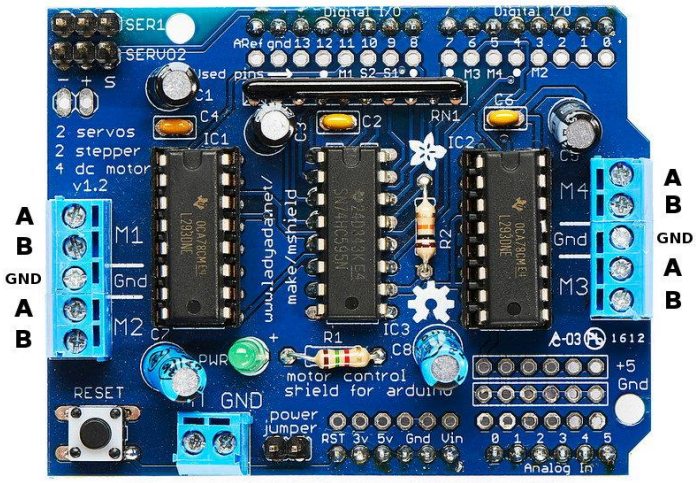

Figure 4. Adafruit Motor Shield Overview [17]

The aim of combining this shield into our platform is to obtain a tidy compact mechanism to control the DC motors that are driving the robot wheels. Its two L293D dual H-Bridges motor drivers allow to work with up to 4 bi-directional (forward/backward) DC motors which can be connected to the M1, M2, M3 or M4 screw terminal blocks. Additionally, it can manage two supplementary servomotors and two stepper motors, which were not considered under the scope of this project. Finally, it must be mentioned that this added board requires power supply which is provided by and external battery through the power header of the motor shield. Its on-board LED lights up to indicate the proper energy input.

4.3 – Locomotion System

Once the electronic brain of the robot has been described, it was necessary a moveable body to be attached at. Regarding the robot future task and desired performance, a round wooden base was cut and selected to fix the different elements that compose the system hardware. The chassis design and building were based on the intention of keeping the simple robot architecture of the previous model.

The robot propulsion was designed to be done by a 3-wheels working system with two differential drive motorized wheels, which set the motor movement and direction, and a castor wheel which can steer around its vertical axis. The driving wheels are situated at the front of the base for a more suitable steering and aligned along the same horizontal axis for stability intention. On the other hand, the castor wheel is positioned at the back on the perpendicular axis from the mid-point of the wheel base distance, providing the essential overall robot stability and parallel plane of the body to the ground.

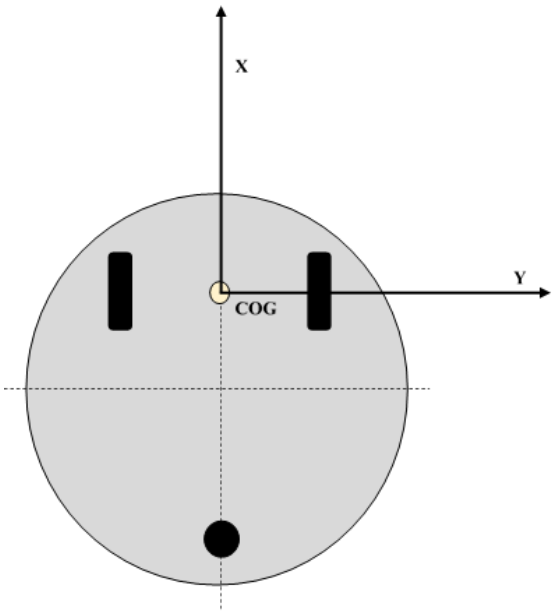

Figure 5. Wheelbase configuration

4.4 – Sensors System

Continuing with our robot-human comparison and metaphor, once the body and the brain were designed and created, it was time to incorporate the architecture with the ‘senses’ or sensory system to get the environment signals and send these measurements to the embedded microcontroller of the board. The selection of these sensors was made based on the desired task to achieve, the movement tracking and obstacle avoidance on the trajectories.

In order to transform the mechanism into a perceptive robot and looking at the tracking goal, it has been chosen to integrate in the hardware four ultrasonic sensors HC-SR04 [18], which are composed by transmitter, receiver and a control circuit. Its function is to measure the distance to an object by using an ultrasonic wave: a timer starts when the signal is triggered; it will be reflected backward in case it finds an entity on its way, stopping the timer when the signal reaches the receiver. Distance can be calculated based on the wave travel duration and its speed through the air.

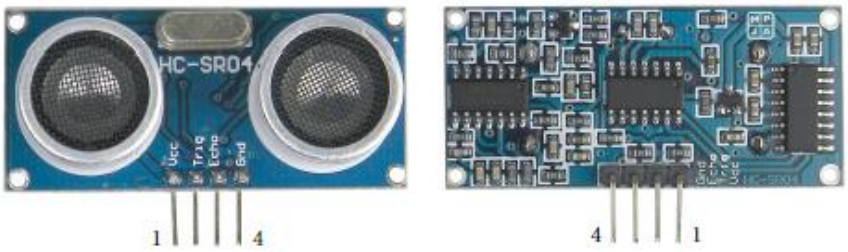

Figure 6. Ultrasonic Sensor HC-SR04 Overview [18]

Each sensor has four pins (ground, trigger, echo and voltage) that needs to be connected to the main circuit to get power, transmit and receive information. In this project, the four components are linked to the digital pins of the Arduino Mega2560 board and placed in pairs at the front and back edges of the base, choosing its orientation based on the coverage cone for best performance shown in the following figure:

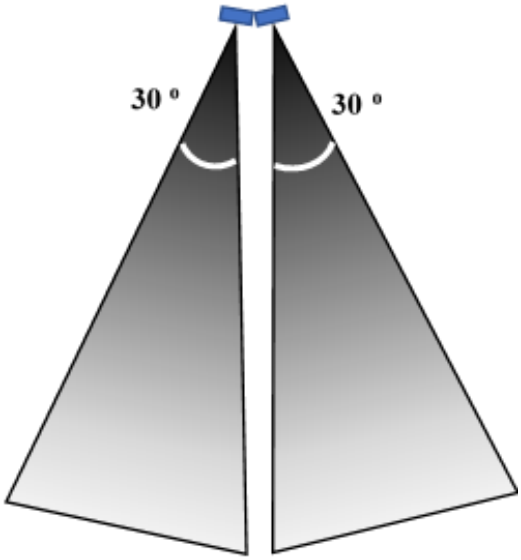

Figure 7. Ultrasonic Sensor Coverage Cone

As it can be seen, the measuring angle is limited to 15º (30 degrees in total for each sensor) and its range goes from 2 cm up to 4 m. However, for the aim and environment conditions of this project, this sensor was chosen as a suitable option.

Regarding the secondary but important task of obstacle avoidance while moving, the infrared sensors FC-51 [19] were selected and aggregated to the architecture. These elements are constituted by a transmitter, an emitting LED which sends an infrared frequency signal; and a receiver, a photodiode which detects the reflected wave. The embedded LM393 dual voltage comparator of the circuit check the obtained signal with the set reference (distance to any obstacle that needs to be avoid).

Figure 8. Infrared Sensor FC-51 Overview [19]

The sensors have a coverage cone similar to the one shown in Figure 7 of 35 degrees angle for each one and a short detection distance of 30 cm maximum. For this reason, these sensors are used to avoiding obstacles and not to track the moving objects. Additionally, its low cost makes the election interesting even considering the existing drawbacks as for example detecting black surfaces (which absorb the infrared waves). A well-designed environment for the future game becomes essential.

Its position on the wheelbase were chosen to be at the lateral edges, slightly outside to avoid wrong measurements with the robot body itself, and the length to perceive obstacles was adjusted manually with the analogue potentiometer to provide the robot with enough manoeuvrability.

Finally, it has to be mentioned that there is one more type of sensor which was included in the architecture, the motor encoders. This will be explained together with the motors in the following subsection.

4.7 – Actuators System

After describing, in the previous subsections, the locomotion and perceptive systems of the robot, it is explained how both systems gather together in the actuators system. The elements that transmit the movement are two DC gear motors GM25-370 of 6V with integrated encoders [20].

Figure 9. DC Motor Overview [21]

These motors are installed accordingly with the distribution explained in the subsection 4.3 to drive the front wheels. The design includes these motors with encoders with the goal of controlling the robot speed and make its performance more optimal. Two independent actuators were selected, one for each wheel, and they are powered and controlled through the explained Arduino Adafruit Motor Shield, which makes this task more practical and easier to achieve.

The sensor attached to these motors is an encoder which can track the pulses that the motor shafts have turned and in what direction (with the proper code). These give the opportunity to have a better control on the wheels of the robot than in the previous model in which this project has been based.

Figure 10. Motor Encoder Overview [18]

Finally, the wheels needed to be chosen to fit with the motor shaft form. For this reason, two 3D printed wheels were ordered, with their thin tyre, to be incorporated with the motors. The selected wheels have a size accordingly with the surface in which the robot will operate, smooth internal grounds.

4.8 – Power

After describing all the hardware elements that have been used to build up the robot under the scope of this project, it is necessary to describe how to power all the components. A recap of every hardware system is made regarding their power needs and a final decision has been accordingly taken.

Starting from the main electronics, the Arduino Mega2560 can be powered via the USB connection to the computer or using an external power supply. Its range of working conditions is from 6 to 20 V, with a recommended voltage of 7-12V not to damage the circuit. Moreover, the described motor shield which is stacked on top requires a variable voltage depending on the motor that are attached to the circuit. However, the controllers on this shield are intended to work correctly on a range from 4.5V to 25V and to provide up to 600 mA per motor with a 1.2 peak current.

Looking at the different sensors which are installed, the ultrasonic sensors have a working voltage of 5V and working current of 15mA; the infrared sensors a range from 3.3 to 5V; and the motor encoders require 3.3V of power supply.

Regarding the actuators or DC motors, they need 6V and provide their maximum power at a current of 1A.

Considering all these elements characteristics and the need for the robot to work independently of a computer connection (which will not provide enough power for the motor requirements), the design included the selection of an external battery pack. In this case, the 'Turnigy nano-tech' Li-Po with 3 cells and specifications of 11.1V and 5000mAh [22]. These power matches the total voltage and current conditions of the project hardware.

Figure 11. Battery Pack Overview [22]

It has to be mentioned that Lithium Polymer batteries are one of the best types for small robots due to its characteristics: high capacity, high power and high discharge rate. Moreover, its weight and shape make it a suitable pack to be stacked into the designed wheelbase.

4.7 – Complete system building

Once every component of the hardware architecture has been independently described in the above subsections, the process of creating the robot is explained step by step. This construction has been based on the decision process previously commented.

First of all, the round base or wheelbase has been cut from a wood panel taking the same dimensions as the preceding model of the AirLab, which was the inspiration for this project. On this circular piece, it has been marked the locations of the different elements that need to be stacked to the base. These locations were measured to fit with the design requirements.

The first system to be attached was the motion mechanism. Both DC motors were connected through their shafts to the wheel connection piece. On the 3D printed wheel, four holes were made to screw it to the connection pieces, so the motors and the wheels were put together. Following the wheelbase configuration of Figure 5, the driving wheels needed to be attached to the front part of the base by using a fixing L-shaped metal element. This has four holes on one side to be screwed to the wooden board and four holes on the other side to be joined to the motor body.

As it can be seen in the following picture, the composition of the wheels position matches the design. The castor wheel has been fixed in the perpendicular to the motor axis and the wheelbase had its 3-wheel configuration created. Special care was taken to keep the horizontality of the robot.

Figure 12. Building the robot: wheel location

Two small holes were made to pass the motors and encoders wires to the upper part of the base where all the other components have been placed and the wiring has been done.

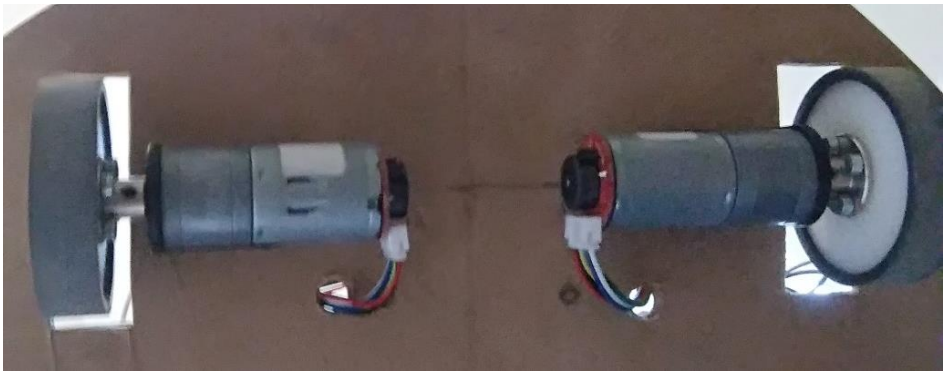

Figure 13. Building the robot: Motors system zoom in

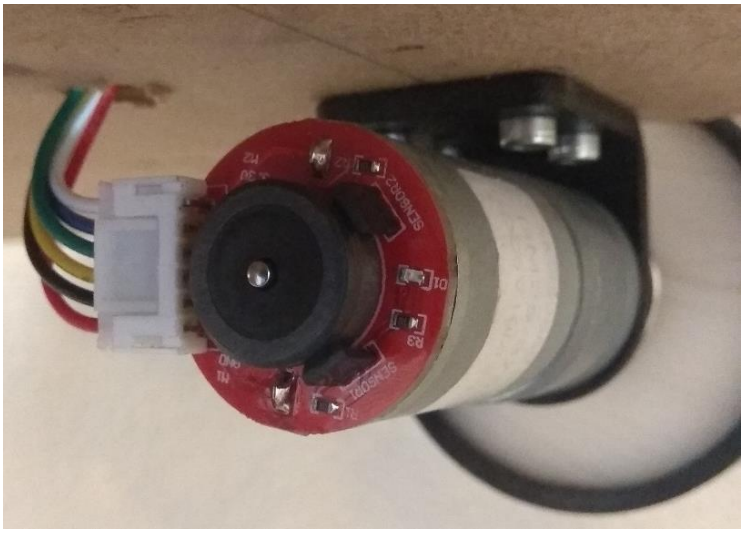

Figure 14. Building the robot: Encoder and motor fixing

In the next figure it can be appreciated how the switching ON/OFF button for the robot power supply has been located in the outer part of the wheelbase, near the castor wheel. The reason to situate it there was to have an easy and comfortable access when the upper part is covered in the future by a protection case.

Figure 15. Building the robot: Castor wheel and switching button

Then, it has been created the fixing components for the ultrasonic sensors. This were made from metal, holed and screwed together to be attached to the base and to fit the sensors transmitter and receiver in the holes. Having an immobile position is a key point for a successful performance and measurements accuracy of the HC-SR04. At the last stage of the creation process, they were set steady in the holes by inserting plastic rings around the transmitter and receiver. Moreover, the four infrared sensors were also stacked in their designed positions.

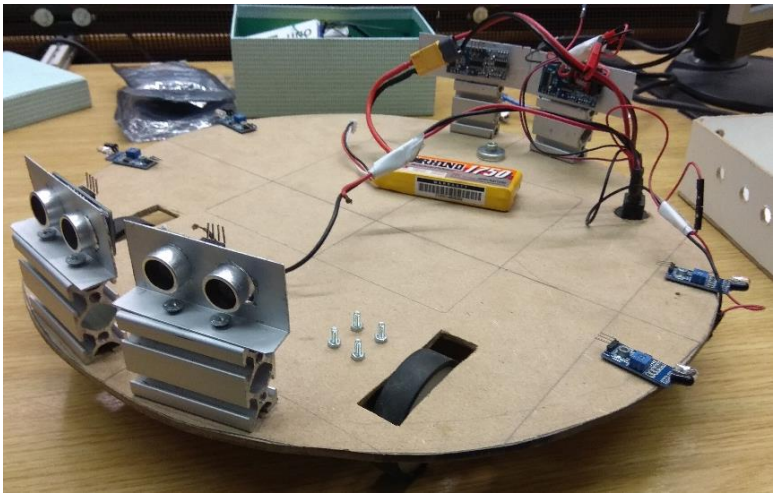

Figure 16. Building the robot: Intermediate stage, sensors placement

It can be seen in the previous picture the presence of an old and invalid battery pack that were substituted later for the one described in the above subsections.

The next step was to situate the Arduino Mega2560 board in the centre, mount the motor shield on top, locate the battery pack at the back and carefully make the whole system wiring. From the button, eight wires (four Vcc, four ground) were connected in pairs: two from the battery pack so the button is in charge of switching on/off the power of the mechanism; two were linked to the motor shield screw terminal to give the necessary power supply; two were joined with a Jack plug and connected to the correspondent

Arduino board terminal; and the last two are passing through a voltage transformation chip that converts the battery voltage into 5V and were pinned into the additional 5V/ground power pins, which were necessary as an extra connection to power the sensors (no enough powering pins on the Mega board).

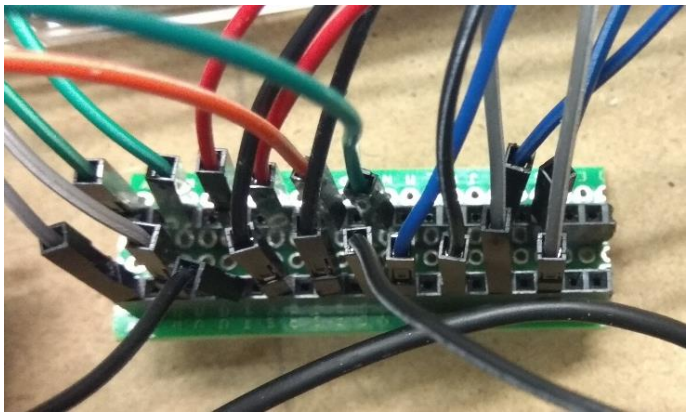

Figure 17. Building the robot: Additional power (Vcc / ground) pins

However, the encoder sensors need 3.3V instead of 5V. For this reason, a supplementary voltage transformation chip was used.

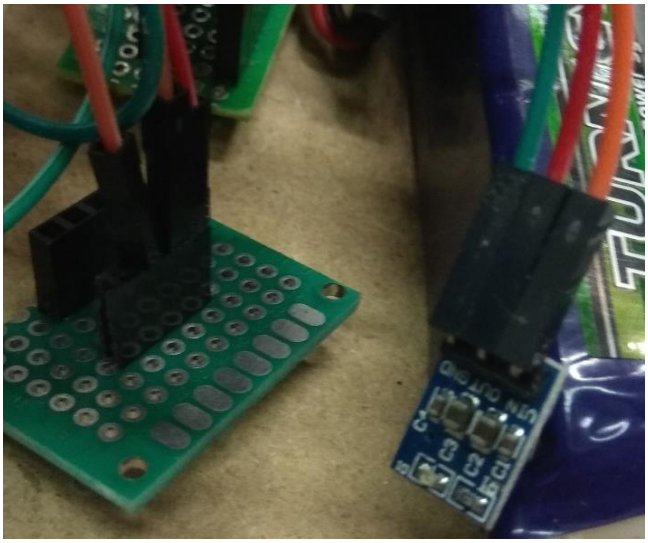

Figure 18. Building the robot: 5V to 3.3V transformation chip

The ultrasonic sensors were wired to the power/ground additional pins and the trigger and echo pins were connected to the Arduino board digital pins. Regarding the infrared sensors, two wires were for power supply and the control wire of each FC51 was connected to a digital pin of the Mega2560. Lastly, the motor power wires were linked to the Adafruit motor shield screw terminals and the other four wires, corresponding to the encoder, were connected to the additional power pins and to the Arduino board digital

pins with interruptions (this is important regarding the encoder functionality and the developed code).

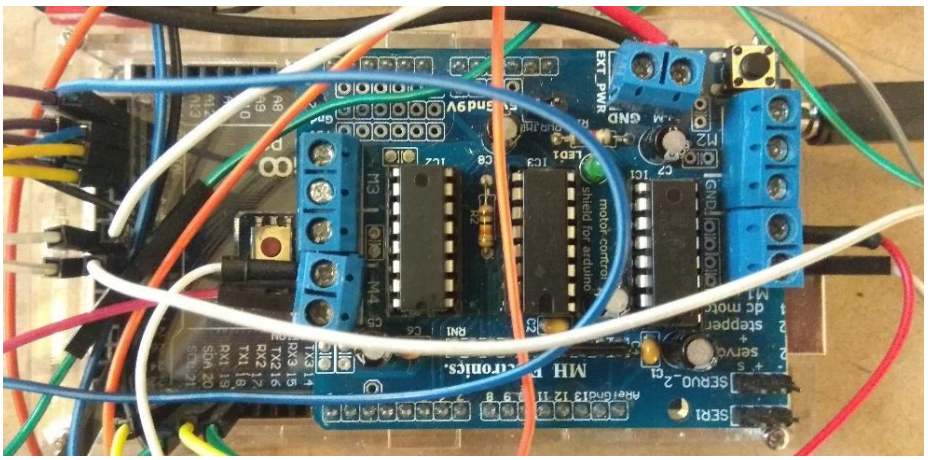

Figure 19. Building the robot: Arduino Mega2560 and motor shield wiring

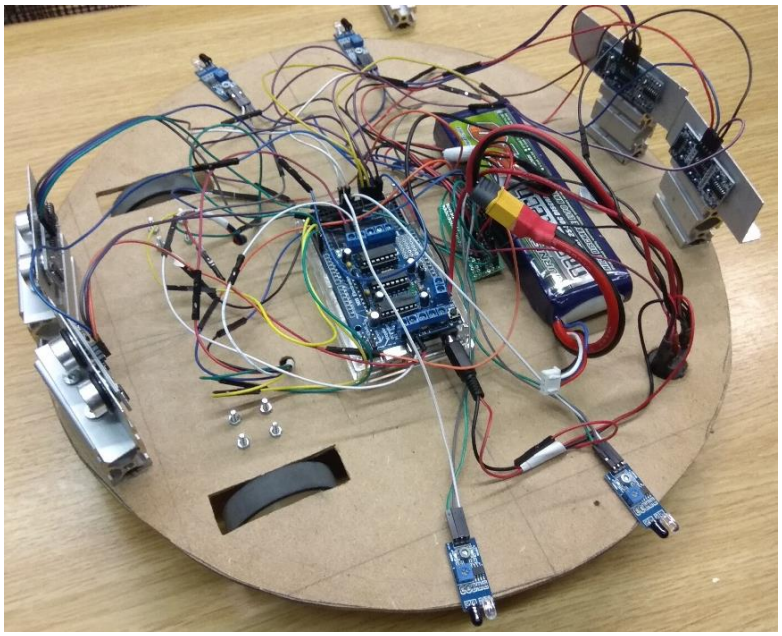

Figure 20. Building the robot: Intermediate step, wiring

At this point of the robot building, it was noticed the need to compensate the weight of the battery pack. An unbalanced weight distribution can affect negatively to the motion performance of the system, provoking the wheels rolling without enough adherence or a bad turning movement. To avoid this mass location at the rear of the robot, which elevates the front part, it was necessary to create an innovative weight point. This was solved by fixing a metal rectangular box full of small solid metal pieces.

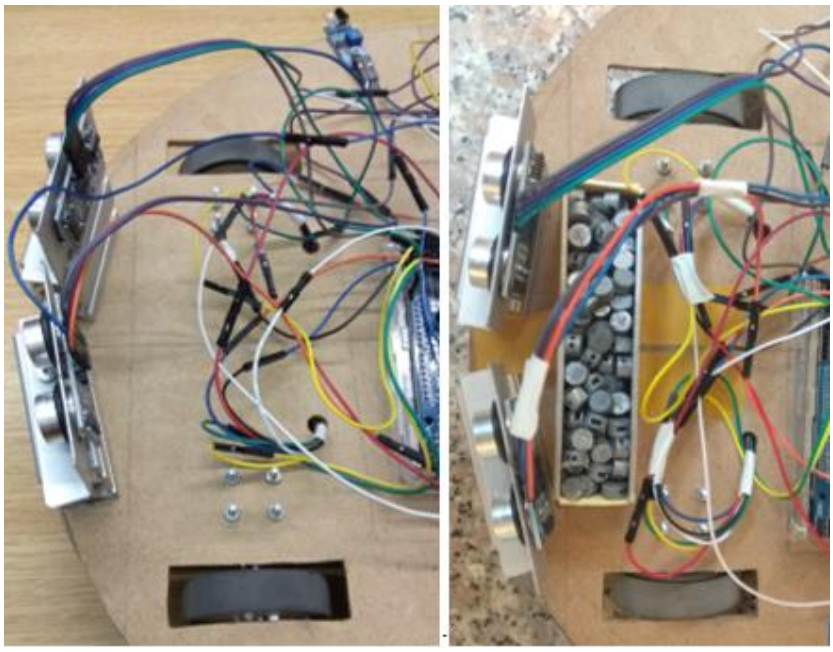

Figure 21. Building the robot: Weight distribution

Finally, some pictures are included to show the current state of the robot once the building process have been finished.

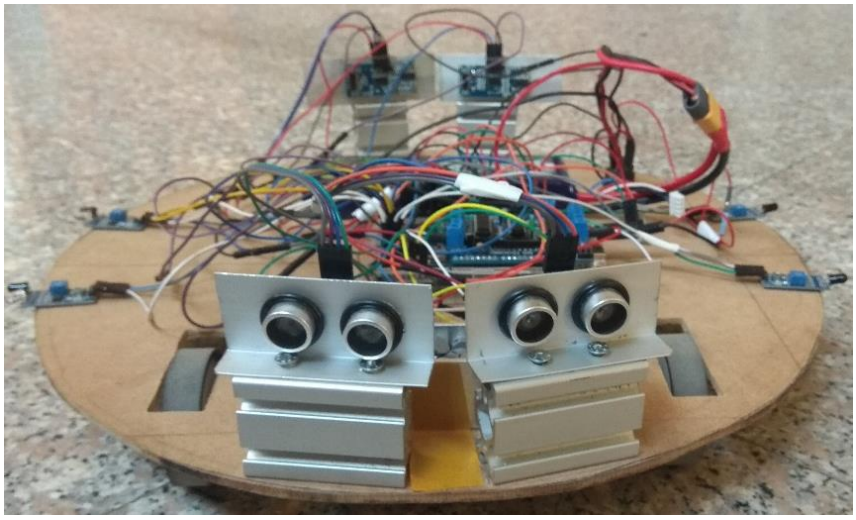

Figure 22. Building the robot: Front view

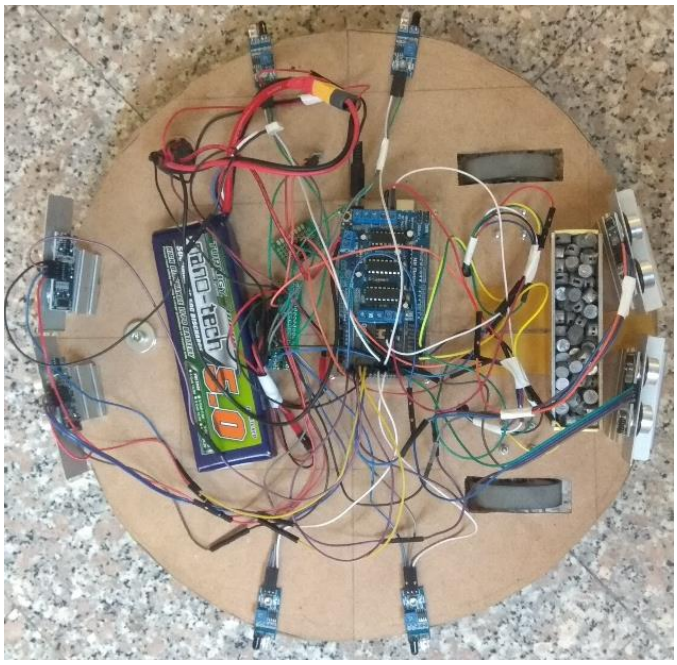

Figure 23. Building the robot: Top view

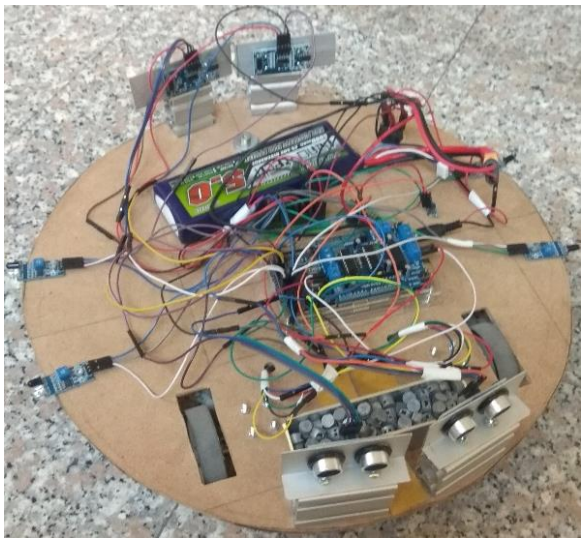

Figure 24. Building the robot: Perspective view

CHAPTER 5:
SOFTWARE ARCHITECTURE

Once the physical architecture of the system was finished, the mechanical and electronic work of the thesis leaded the way to the software creation. It was the moment to think about the code that needed to be implemented into the robot brain, i.e. the Arduino Mega2560 board. The developed code has been generated using the Arduino IDE environment and the logic behind it is the core of this chapter.

The explanation of the main parts of the code and the decision process that has been followed to create it are the content of the following sections. Firstly, it is briefly described the intentions when programming the robot behaviour and the reasons behind it. Then, part by part, the software is commented and clarified.

5.1 – Programming logic

The robot autonomy was programmed in a single Arduino sketch named '*Robot_Tracking_Game.ino*'. The code file can be found at the appendices of this document and can be run directly on the system. In case it needs to be upload, it is important to specify the board for which is was designed (Mega2560), the processor (should be automatically selected) and the communication port. This configuration of the sketch is essential when creating and implementing the programme.

As it can be deduced from the hardware architecture, the created robot offers many different variants of the classic tracking game, i.e. the robot can play as the tracker and has to 'catch' the child; the roles can be inversed and the robot can escape from the player who tries to catch it; or a combination of both game modes could be switch and taking place. In order to apply the second and third option, the four ultrasonic sensors are needed. However, for this first prototype and test, it was selected to implement the traditional mode in which the robot has to track the child.

The code logic is based on the robot environment recognition through the sensory system, the signal interpretation and response at the embedded circuit of the Arduino board and the consequent action which is delivered from the actuators. For that reason, the mechanism has to periodically send and receive ultrasonic and infrared waves to detect the object to track and to avoid obstacles on the way. There has to be a decision planner based on those sensor measurements to select the next movement of the robot.

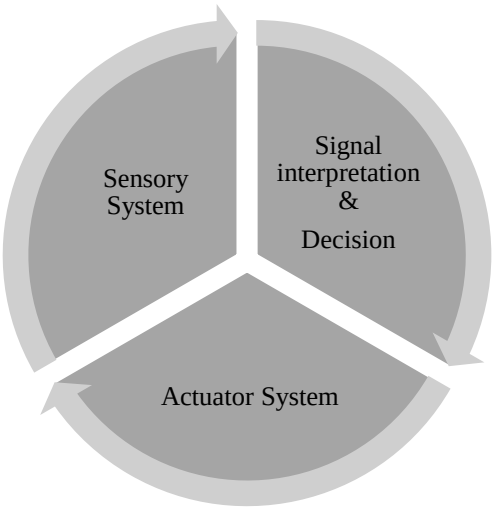

Figure 25. Programming Logic Cycle

In terms of control, the objective was to set the DC motors speeds to make the task performance adequate for the game conditions. There are many possible control techniques that can be applied but it was decided to use a classic PID controller for this purpose. This option was tested to be suitable and to give a good performance.

Finally, it has to be mentioned that the code structure was designed to be as understandable as possible, keeping the main loop with few lines and arranging the different actions in clear functions that are called from the main loop or from other functions. In the next subsection, it is explained more these decisions through the code-

5.2 – Code structure

5.2.1 – Variables definition

The first thing was to include the necessary libraries that help with the creation of the code. The Adafruit Motor Shield library [23] makes the control of the motor easier by containing motor classes and functions to set the direction of the shaft rotation and its speed. On the other hand, the PID library [24] makes clearer the creation and computation of the controller.

Making use of the mentioned motor shield library, two DC motor classes were created, one for the terminal M1 (right motor) and the other for the terminal M4 (left motor). The next step was to define the variables to save the pin numbers in which the motor encoders were connected, having A and B pins for each one. These pins signals will be read for the encoder measuring, which is explained later.

Regarding the ultrasonic sensors, four variables were set to save the pins numbers for the trigger and echo pins of each front sensor. Additionally, some variables were designed to store the measures obtained from the sensors, i.e. duration of signal reflection and distance, which needs to be calculated. On the sensors subsection it is explained how is made.

Analogically, for the infrared sensors it was defined four variables to declare the wired control pins of the four elements, two at the right side and two at the left one. It has to be mentioned that these pins definitions are not essential, but they certainly make the code clearer and easier to use, because the variables names are used instead of the pin numbers. Then, another four variables will save the received signal from the sensors' waves.

There were some parameters that were designed to be set by the user based on the desired game environment and conditions: the maximum speed at which the motors can run, set at 120 rpm; the velocity the motors should have when turning, defined as 100 rpm; the maximum distance at which it was decided for the ultrasonic sensors to measure, set as 150 cm; and the minimum distance at which the robot considers that it has caught the child, defined as 20 cm. All these parameters were decided based on the game conditions and the testing environment specifications, and they can be modified accordingly to a new set of conditions.

Regarding the encoders, there are many variables that needed to be declared. It was defined as a constant parameter the number of pulses per revolution that an encoder can measure, obtained from the datasheet. Two volatile variables were created to store the pulses which are measured by each encoder and some time variables were necessary to save the code running time and the differences between measurements.

It is important to explain the defined coefficients that quite proportionally relate the rpm values which are measured at the encoders and the PWM signal values that are required by the motor classes to set the speeds. These coefficients were obtained from the tests that were carried out at the DC motors, checking the speed at which they run when the PWM values were incremented from 0 (stop) to 255 (higher voltage to the motor, maximum speed). Comparing the results, it could be found the relationship at the desired range of velocities, limited by the maximum set speed.

Finally, the PID classes were created using the mentioned library terms., with the proportional, integral and derivative gain values defined based on the desired behaviours of the motors. There is the need of using one PID for each motor as they can run independently and at different speeds on a differential driving wheels system.

5.2.2 – Setup configuration

The '*void setup ( )*' function runs the configuration code only once when the system is powered up or after a board reset. It has been used to initializes pins mode for

the ultrasonic (trigger pins are outputs, echo pins are inputs), infrared (inputs) and encoder sensors (inputs).

It has to be commented that the serial port can be started in order to track the different signals through the Arduino Serial Monitor and test the programme and do some debugging.

An important part of this setup is to declare the interruptions for the motor encoders. Using the command '*attachInterrupt ( )*', two interruptions were designed at the encoder A signals in pins of the board with interruptions function (in this case, pins 19 and 21). As its name says, an interruption will stop the main code to execute a function when the defined action happens on the declared pin. In this project, when the encoder A signals are rising (from LOW to HIGH value), the encoder values are read. This is explained in the following subsection.

Some other PID parameters are initializes in the setup too using the previously defined classes and the library functions: the mode (automatic instead of manual), sample time (how often the controller algorithm is evaluated) and the output limits for the resulting signal from the PID, which has to be limited by the speed range (0 to the maximum speed).

To conclude the setup configuration, the initial state of the motors is declared as stop state and saved into the motor state variable, which is useful to know the current state of the motor movement and it will be used in the decision-making function.

5.2.3 – Main loop

Once the configuration of the system has been done, the following section is the '*loop ( )*' function, which is the code main cycle that is repeated consecutively while the system is powered up. This allows the programme to perform periodic actions to change and respond to the robot environment.

In the case of this project, it has been included three different functions that are actively called to perform the robot task. The execution order can be simply divided in two operation blocks: first it reads the sensors signals, through the functions '*readIR ( )*' and '*readUS ( )*', to identify the environment conditions; and secondly it calls a function, '*defineMove ( )*' to select the appropriate robot movement, which is performed by the actuators. These functions are explained in the paragraphs of the sections below.

The intention when programming this main loop block was to keep it as clear and concise as possible, leaving the rest of the actions to independent functions that can be called one form each other. In this way, the code is more straightforward and understandable for other users.

5.2.4 – Sensors functions

As mentioned before, the first step for the robot task is to perceive the environment. From the main loop, two functions are called and described in this section regarding their inputs, actions and outputs.

The function which englobe the ultrasonic sensors actions has been named '*readUS*' and it takes as inputs the trigger and echo pins for a pair of sensors. In this case, both front sensors pins are passed to the function for calculations. The process starts by generating the ultrasonic wave by setting the trigger pin from LOW to HIGH value. Then, the reflected signal needs to be read at the echo pin by using the '*pulseIn*' command. This gives the duration that the wave has travelled. By having this time value and using the Equation 1, the distance from the closest object can be obtained.

$$D = \left(\frac{t}{2}\right) \cdot v$$

Equation 1. Distance measured by HC-SR04

In this formula, *D* is the distance from the sensor to the object to be tracked, *t* is the total time the wave has travelled, and *v* is the speed of sound in the air. However, the time that has been calculated is in microseconds and it counts both ways of the wave, from the transmitter to the object and back to the receiver, so that is the reason why is divided by two. Once the factor conversions are applied, the resulting Equation 2 gives the distance in the desired units, cm.

$$D\ [cm] = \left(\frac{t\ [\mu s]}{2}\right) \Big/ 29.1\ [cm/\mu s]$$

Equation 2. Distance measured by HC-SR04 in cm

The measured distances from both front sensors are saved in the correspondent global variables '*distanceFrontRight'* and '*distanceFrontLeft'*, so the function does not return an output.

Regarding the infrared signals, the function '*readIR*' takes as inputs the four sensors pins names and read the value of the signal received at the sensor, storing the value as HIGH if the infrared wave is not reflected in the set reference length or LOW if there is an obstacle to avoid at that length. These values are saved in global variables, so the function does not return an output too.

The third type of sensor to be considered is the encoder which is attached to each DC motor. As explained at the setup section, the interruptions call the function which update

the encoder values which are the number of pulses that has been read in a period of time. For better understanding on how the encoders calculations have been programmed, it is shown in the following figure the signals emitted at both pin A and B of a rotary encoder:

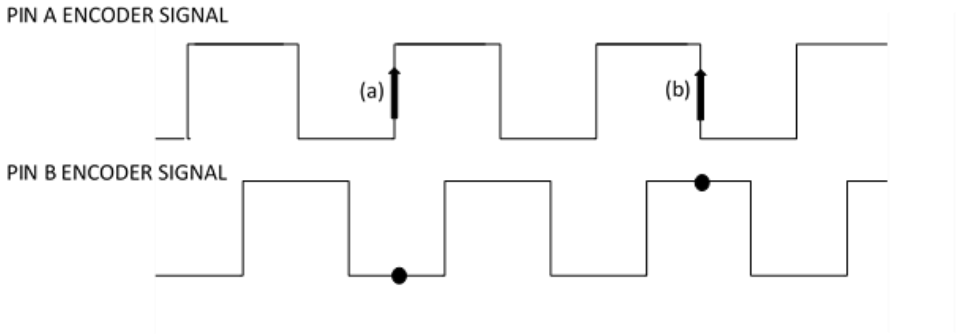

Figure 26. Rotary encoder signals

The signals of the encoder have a 90 degrees phase difference, the pin A signal is faster, so there can be two situations as shown in the previous figure: (a) when pin A signal is rising, pin B signal is LOW so it means that the motor is moving forward; (b) when pin A signal is rising, pin B reading is HIGH so the waves are moving in the opposite direction. The encoder values can be updated accordingly to the chosen direction of movement.

The first interruption calls the '*updateRightEncoder*' function when the pin A signal of the right encoder is rising. At this function, it is read the value at that moment of the pin B signal and, depending if this wave is LOW or HIGH at the moment, the motor is moving forward or backward so one pulse has to be added or subtracted to the encoder measurement.

Similarly, the second interruption calls the function '*updateLeftEncoder*' function when the pin A signal of the left encoder is rising. However, the different with the previous function is that the adding/subtracting situations are inverted, because it was considered how the motors were mounted at the axis in opposite directions of rotation.

5.2.5 – Decision-making

After the Arduino board has received the information from the sensors, it can apply the logic to compute the necessary actions to take to complete the intended task of the robot. In this case, it has been decided to program the decision process in a function named '*defineMove ( )*'. This code block takes as inputs the measurements obtained from the previous part of the programme, i.e. two length values from ultrasonic sensors and four responses from the lateral infrared sensors.

The robot decision making process was planned for an environment and space conditions previously set according to the game. Due to that, certain possibilities were not included in this initial prototype because there will not be the surrounding circumstances for them to happen, e.g. the robot will not have to pass through a narrow corridor to track the child.

The first action which was programmed was to check the presence of obstacles on the way. It was set as a priority the action of turning in the opposite direction of where the obstacle is found. If any infrared sensor of the right side of the robot detects an object, it sends the command to turn to the left for a small amount of time enough to avoid the obstacle. Analogically, if any infrared of the left side got activated, the robot will turn to the right to avoid the collision. For both of these turning actions, the motor state will be declared differently as if the robot were turning in a tracking situation.

Once it has been checked that there is no object in the robot trajectory, it has to be considered the received signals from the sonars. Three possible situations can occur:

- In case both of the front measurements are larger than the set maximum distance in which the robot is able to track the movement, it was decided to consider the previous motor state to know the trajectory that the robot was following before it lost the reference. With that intention, it is checked which is the last state saved at memory and the robot movement is set to continue that trajectory.

  The stopping condition is a counter that has been included to set a time limit for the robot to stop continuing with the trajectory and go to steady state when it has lost the tacking reference. This counter is increased until a user defined maximum and it is reset whenever the motor changes its state.

- If any of the measurements is smaller than the set minimum reference distance, then the robot has to stop because it has caught the moving objective.

- When the measurements from the ultrasonic sensors do not fit in the previous conditions, then the robot has to track the moving object. The implemented logic is based on a comparison between the right and left measures through the absolute result of the values subtraction.

  If the result of the comparison is among the defined range of 5cm (this range was set to consider the possible small differences of measures between sensors due to their orientation), then the robot is set to move forward, update its state and reset the counter mentioned before.

  If the result of the comparison is out of this range, then it is checked which of the two measurements is bigger: if the right distance is smaller than the left one, then the robot was programmed to turn right to track the moving object which is closer to that

sensor; the same logic is applied when the left length to the tracking reference is smaller than the right one and the robot turns left.

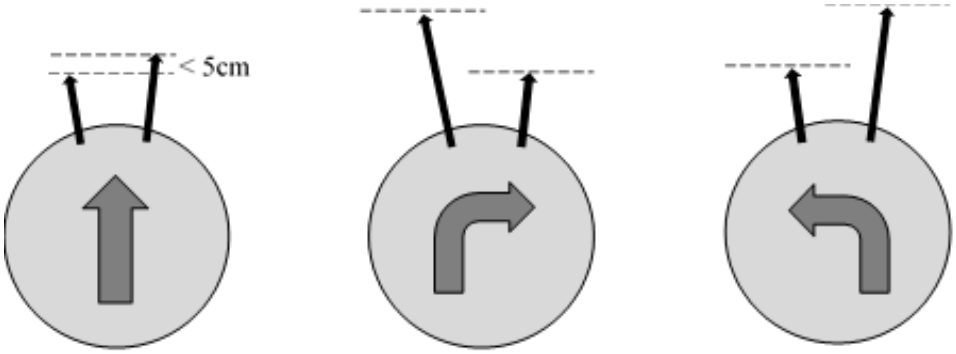

Figure 27. Movement decision logic

This is how it has been chosen to program the decision making from the sensor signals. In the next subsection it is explained how these decisions are transformed into concrete actions at the wheels.

5.2.6 – Movement functions and PID controller

Out of the main loop, it has been created four different functions to specify the actuators signals, one for each possible movement condition: forward, stop, right and left. These functions are called from the decision-making function which has been described above; moreover, they do not require any inputs nor outputs to be executed. They work with the global variables and classes which have been declared at the top of the code.

The programming structure that has been selected at these functions block follows the next steps:

- Check the current state of the motors (it is updated after the movement function is executed) and in case the state is the same as the intended movement, keep the motors set as they were.

- Otherwise, first define the desired setpoints for the speed of the motors. This reference values act as the controller aim.

- Call the function which englobes the PID controller computation. This is explained in detail further on in this section.

- Set the motors movement direction once the speed has been obtained from the previous PID function.

- Finally, update the '*previousMillis*' time variable by storing the last measured code running time. Then, reset to zero the variables which store the encoders values or pulses that has been counted.

At this point, it is necessary to explain how the controller has been designed and implemented to control the DC motors speed. Having the option to control the motor speeds give a lot of flexibility to the possible applications of the mechanism. For the scope of this project and this first prototype of the tracking robot, it has been decided to create a PID controller which has the conditions to cope with the intended task of keeping a smooth transition between different movements of the robot, i.e. changing speeds.

The PID controller is a well-known technique that calculates the existing error between a measured input (usually, coming from a sensor) and a user defined setpoint that has to be achieved. The controller purpose is to adjust an output signal that can make that error disappear or minimize it. This adjustment is done through the tuning parameters *kd* (proportional term to the error), *ki* (integrative term, proportional to the cumulative error) and *kd* (derivative term, proportional to derivative of the error or error slope).

$$output = K_P \cdot e + K_I \cdot \int e\,dt + K_D \cdot \frac{d}{dt} \cdot e$$

$$e = error = setpoint - input$$

Equation 3. PID general algorithm

It is shown above the formula of the PID controller and a lot of information can be found in many sources from the literature. However, the interesting part for this project is how to apply it at the Arduino environment and for the purpose of speed control. It has been explained before at the variables definition subsection that the PID controller has been implemented by using a documented Arduino library. This makes the code clearer, easier to understand and the implementation is straightforward.

As mentioned, the first step was to create the PID classes and define its attributes: *input*, the variable that needs to be controlled; *output*, the variable that is adjusted by the PID action; *setpoint*, the reference value; and the tuning parameters *Kp*, *Ki* and *Kd*. These tuning parameters are defined to give a good response and tested through experimentation. Two PID controllers or classes were created, one for each motor, and at the setup loop, some parameters as the sample time and the output range were defined.

Following these attributes, in this project the *setpoint* attribute correspond as commented before with the speed that the motor needs to achieve, in rpm, for the intended movement; the *input* attribute correspond to the measured current motor speed, in rpm, that can be obtained through the encoder sensor; and the *output* attribute correspond to the PWM

(pulse-width modulation) signal that it is the necessary voltage to pass to the actuators to generate the speed. In the next figures it can be seen a general PID function block with the library attributes and the specific one for the case of this project.

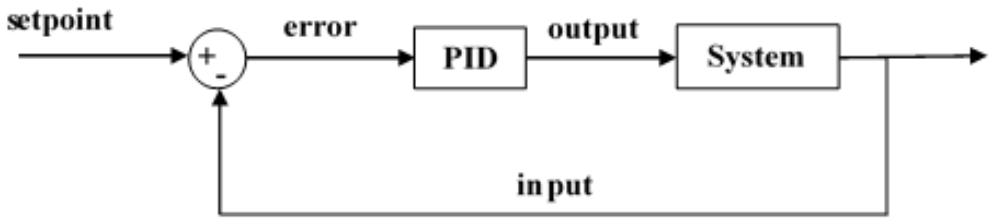

Figure 28. General PID function block

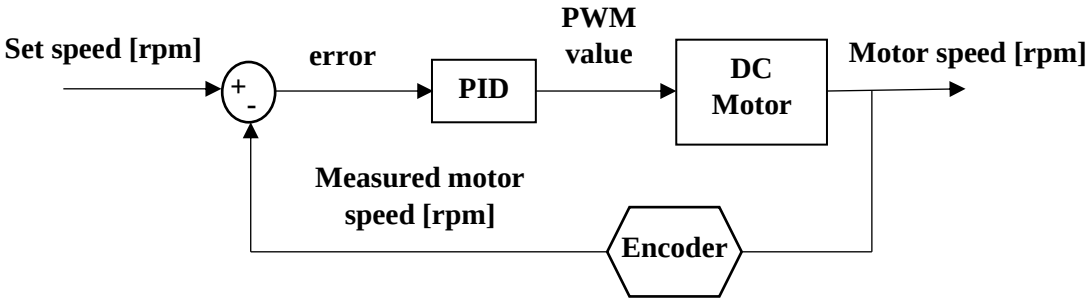

Figure 29. Implemented PID function block

Then, after explaining this implementation process, it has to be described the function which makes the controller calculation. This function was named '*PID_Computation*' and takes as inputs the setpoints variables that are requested at the movement functions. At the beginning of the function, it is saved the current running time of the code and compare to the previous time stored to get the interval of time between measurements. Once this interval has been obtained, it can be calculated the speed in rpm of each motor by applying the following equation:

$$v\,[rpm] = \frac{(encoder\ value\ [pulses] \cdot 1000\ [ms/s] \cdot 60\ [s/min])}{(interval\ [ms] \cdot encoder\ PPR[pulses/rev])}$$

Equation 4. Formula to calculate the motor speed from encoder

The '*encoder value*' is the number of pulses that has been measured by the encoder in the correspondent '*interval*' of time in milliseconds and the '*encoder PPR*' is the pulses per revolution that the encoder is able to read. This equation has to be applied for both DC motors and the resulting speed is the '*input*' attribute for the PID controller. Now all the attributes for the PID class are defined and the library function '*Compute ( )*' can be

execute. This contains the PID algorithm that has been explained before and it calculates a new value for the attribute '*output*'.

At this point, it is necessary to identify the correspondent PWM value for the obtained motor velocity. For this purpose, it has been run some motor tests to obtain the speed in rpm for different values of the PWM signal, that were gradually incremented. Then, focusing on the desired range from zero speed (robot stop) to the defined maximum motor speed, it has been extracted the transformation coefficient for each motor. These coefficients were slightly different because the motors performances were not the same.

Now that the motors speeds were obtained, the movement functions can set the motor direction of rotation depending on the specific function. When going forward, both motors are set at forward direction; when turning, first both motors start running forward for a short period so the motors speeds can reach the desired values, and then the wheel of the side at which the robot is turning is release to perform a skid steering turning; finally, when stopping, the motors are set forward for a short period of time so the motor speeds can reach the zero values from the values they have before and then they are release to avoid destroying the motors.

The whole code loop has finished, and it is repeated and constantly executed during the robot task to achieve a good performance.

CHAPTER 6:
CONCLUSIONS AND FURTHER WORK

6.1 – Conclusions

This dissertation was aimed to explore the robotic field of robotgames and study the applications that different robot can have in this area. This project was set to create an easily replicable first prototype of a small ground robot with the intention of participating in a tracking game with children. Programming and controlling this mechanism through the Arduino IDE environment were the set challenge after the mechatronic building.

Firstly, a research through the existing literature was carried out to collect information about general concepts and theories that have been released regarding the interaction of robot with children and the possible or real benefits of incorporating robots into the children games, especially in cases of children with disabilities or social disorders. The importance of playfulness through the children development needed to be understand before setting the task of the projected robot.

An important part of the project that has been done during this thesis is the design and creation of the robot itself. The production of a new ground robot with the desired specifications, instead of reusing an old prototype, was a necessary stage to obtain an appropriate model to be controlled. Producing the hardware architecture of the mechanism, understanding the relationships between the different components and setting all the electromechanical structure has been a fulfilling activity that has been completed with success. Getting to know the creation of the hardware helps later with the software development.

It can be said that the outcome was satisfactory as it copes with the needs of the system for being able to keep tracking of a moving object. All the elements were set and leaded the path for the following phase of programming. That was the second important part of this work, understanding the world of Arduino platform and applying its functionalities to create a suitable code for the prototype. The software development was carefully intended to be a first approach to an optimal performance and, after running some real tests at the laboratory and checking the robot responses, it can be considered as a proper code to achieve the intended task.

At the end of this thesis project, considering the achieved stage of the prototype, it can be concluded that the work that has been carried out for the past months is satisfactory as it matches with success the proposed goals. The key achievement of this dissertation was to define the bases for a robot that can be used as part of the therapies related with the

robotgames area. Designing the specifications, building the mechanism and programming the code and control system were fulfilled challenges of this project.

6.2 – Further Work

The defined goals for this study have been achieved during the progress made with this project, which has been described in detail with this report. However, regarding the amplitude of the robotic and robotgames topics, it can also be seen as a start point for some other future works related with the content explained and the robot created.

Firstly, the prototype can be fully completed by covering the electronics of the robot and creating an attractive shape for involving the children into the game. It can be used some foam material to fill a soft cover that should keep the round form of the base. Moreover, a tactile sensor is planned to be added at the top of the robot, so it can be pushed by the child whenever the game is over.

As a close future expansion of the game, the implemented code can be slightly modified to include the integrated ultrasonic sensors of the back and set a second version of the tracking game in which the robot, instead of tracking the child, will be tracked by the player and its task would be to escape without colliding on its way.

It is projected, as a next stage of the robot, the incorporation of two small antennae at the front of the system to simulate the shape of an animal or bug. These antennae will be linked to two servo motors which will allow the movement of these devices, making them suitable for the purpose of expressing some emotions while moving. There are some researches going on in this area and it will be a good opportunity to combine this additional feature to the prototype of this project.

Moreover, for validating this project it was enough to make laboratory tests in order to check the correct performance of the programmed code. But when it comes to testing the game within a therapy or as an interactive mode of playing with technology, it would be necessary to make real tests with children under safe predefined conditions. From those tests, it could be extracted the acceptability and interest from the users as well as verifying and improving the robot performance by debugging with real tests.

Finally, it can be mentioned that the technology which was installed on the robot is not the state of the art in the robotic area. This was the intention for this project for the reason explained in the previous chapters, but it can be considered the option of adding different technologies to the next prototypes. This can go from simple light or sound actuators that will increase the interaction to cameras or stereo vision system to perform a better recognition of the robot environment.

REFERENCES

[1] Kanda, “Human-Robot Interaction. A Research Portal for the HRI Community,” 8 Feb 2012. [Online]. Available: http://humanrobotinteraction.org/1-introduction/. [Accessed 2018].

[2] K. Dautenhahn, “PMC. US National Institutes of Health's National Library of Medicine,” 13 Feb 2007. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2346526/. [Accessed 2018].

[3] S. D. L. B. De Santis, “An atlas of physical human–robot interaction,” *ScienceDirect,* vol. Mechanism and Machine Theory, no. 43, pp. 253-270, 2008.

[4] T. N. I. D. K. Fong, “A Survey of Socially Interactive Robots: Concepts, Design and Applications,” The Robotics Institute Carnegie Mellon University, Pittsburgh, 2002.

[5] C. M. e. al, “Toys, games and disabilities: The importance of a Universal Design,” AIJU, Ibi (Alicante), 2007.

[6] «AirWiki,» Department of Electronics, Information and Bioengineering of Politecnico di Milano. [En línea]. [Último acceso: 2018].

[7] F. e. a. Veronese, “Off-the-shelf, robotic toys and physically impaired children: an analysis and suggested improvements,” in *DSAI 2016*, Vila Real (Portugal), 2016.

[8] G. a. B. A. Skard, “Test of playfulness,” in *Play in occupational theraphy for children*, Amsterdam, Elsevier, 2008, pp. 71-94.

[9] B. S. e. al, «Mainstream robotic toys and children with physical impairment: what about playfulness?,» de *DSAI 2016*, Vila Real , 2016.

[10] A. e. a. Bonarini, “A huggable, mobile robot for development disorder interventions in a multi-modal interaction space,” in *IEEE International Symposium on Robot and Human Interactive Communication*, New York, 2016.

[11] A. Bonarini, “Playful interaction with Teo, a Mobile Robot for Children with Neurodevelopmental Disorders,” in *DSAI 2016*, Vila Real (Portugal), 2016.

[12] “What is Arduino?,” Arduino.cc, [Online]. Available: https://www.arduino.cc/en/Guide/Introduction. [Accessed 2018].

[13] “Arduino Forum,” Arduino.cc, [Online]. Available: https://forum.arduino.cc/. [Accessed 2018].

[14] “Arduino & Genuino Products > Arduino MEGA 2560 & Genuino MEGA 2560,” [Online]. Available: https://www.arduino.cc/en/Guide/ArduinoUno. [Accessed 2018].

[15] “Getting Started with Arduino and Genuino MEGA2560,” [Online]. Available: https://www.arduino.cc/en/Guide/ArduinoMega2560. [Accessed 2018].

[16] “Mantech Electronics,” [Online]. Available: http://www.mantech.co.za/datasheets/products/a000047.pdf. [Accessed 2018].

[17] “Arduino Motor Shield,” [Online]. Available: https://playground.arduino.cc/Main/AdafruitMotorShield. [Accessed 2018].

[18] “HC-SR04 User Guide,” [Online]. Available: https://www.mpja.com/download/hc-sr04_ultrasonic_module_user_guidejohn.pdf. [Accessed 2018].

[19] G. M. Giorgio De Nunzio, “FC-51: IR Infrared Obstacle Detection Sensor Module 2 - 30cm,” [Online]. Available: http://www.dmf.unisalento.it/~denunzio/allow_listing/ARDUINO/FC51.pdf. [Accessed 2018].

[20] “Bau da Eletronica,” [Online]. Available: https://www.baudaeletronica.com.br/Documentos/6V-100RPM.pdf. [Accessed 2018].

[21] “Banggood,” [Online]. Available: https://www.banggood.com/it/CHIHAI-MOTOR-6V-100RPM-Encoder-Motor-DC-Gear-Motor-with-Fixed-Support-Mounting-Bracket-p-1230683.html?version=3&cur_warehouse=CN. [Accessed 2018].

[22] “HobbyKing,” [Online]. Available: https://hobbyking.com/en_us/turnigy-battery-nano-tech-5000mah-3s-25-50c-lipo-pack-xt-90.html. [Accessed 2018].

[23] L. Ada, “Adafruit Motor Shield,” [Online]. Available: https://learn.adafruit.com/adafruit-motor-shield/library-install. [Accessed 2018].

[24] B. Beauregard, “Arduino PID Library,” [Online]. Available: https://playground.arduino.cc/Code/PIDLibrary. [Accessed 2018].

APPENDICES

A.1 - Arduino Code: '*Robot_Tracking_Game.ino*'

```
//
// Code created by Juan Luis Guillen Ramirez as part of his Thesis
work
// for the MSc Automation and Control Engineering of Politecnico di
Milano.
// The following code was developed at the installations of the AirLab
and
// its implementation is intended to be for the board Arduino
Mega2560.

// Include libraries
#include <AFMotor.h>
#include <PID_v1.h>

// Create motor variables
AF_DCMotor motorRight(1, MOTOR12_1KHZ); // Create motor Right #1 using
M1 output to 1kHz PWM freq.
AF_DCMotor motorLeft(4, MOTOR34_1KHZ); // Create motor Left #4 using
M4 output to 1kHz PWM freq.

// Define variables for the encoder pins
#define rightEncPinA 21 // Quadrature right encoder pin A
#define rightEncPinB 20 // Quadrature right encoder pin B
#define leftEncPinA 19 // Quadrature left encoder pin A
#define leftEncPinB 18 // Quadrature left encoder pin B

// Define trigger and echo pin numbers for the Ultrasonic Sensors HC-
SR04
const int trigPinFrontRight = 52;
const int echoPinFrontRight = 53;
const int trigPinFrontLeft = 50;
const int echoPinFrontLeft = 51;

// Define variables for the measures of the Ultrasonic Sensors HC-SR04
long duration, distance;
long distanceFrontRight = 0;
long distanceFrontLeft = 0;

// Define variables names and pins for the Infrared Sensors FC-51
const int infraredRightFront = 37;
const int infraredRightBack = 39;
const int infraredLeftFront = 36;
const int infraredLeftBack = 38;

// Define and set variables for the measurements of the Infrared
Sensors FC-51
// HIGH means no obstacle
int obstacleRF = HIGH;
int obstacleRB = HIGH;
int obstacleLF = HIGH;
int obstacleLB = HIGH;

// Define speed and distance constant parameters defined by user
#define MAX_SPEED 120 // Sets the DC motors maximum speed to a maximum
of 120 RPM
```

```
#define TURN_SPEED 100 // Sets the DC motors turning speed to a
maximum of 100 RPM

#define MAX_DISTANCE 150 // Sets maximum measuring of the Ultrasonic
sensors to 150cm

#define MIN_DISTANCE 20 // Sets the minimum distance for the
Ultrasonic sensors to 20cm

#define ENCODER 823 // PPR (Pulses Per Revolution) of the motor
encoders from datasheet

// Variables to save the set PWM values for the motors to run at
int speedSetRight;
int speedSetLeft;

// Coefficients to convert RPM to PWM signal values for the motors
signals
// On the range of application, based on representation of data from
motor tests
float RIGHTCOEFF = 1.1;
float LEFTCOEFF = 1.4;

// Variable to save the current state of the robot movement
String movState = "";

// Define counter
int count = 0;

// Declare PIDs (one for each motor) and define its variables

double kpR = 7, kiR = 1.2, kdR = 0.01, inputR = 0, outputR = 0,
setPointR = 0;
double kpL = 8, kiL = 1, kdL = 0.01, inputL = 0, outputL = 0,
setPointL = 0;

PID myPIDright (&inputR, &outputR, &setPointR, kpR, kiR, kdR, DIRECT);
PID myPIDleft (&inputL, &outputL, &setPointL, kpL, kiL, kdL, DIRECT);

// Variables to store the number of encoder pulses for each motor
volatile long encoderRightValue = 0;
volatile long encoderLeftValue = 0;

// Variables to store code running time and difference
long previousMillis = 0;
long currentMillis = 0;
long temp;

// Variables to save the current RPM of each motor, that are measured
with the encoders
float rpmRightMotor = 0;
float rpmLeftMotor = 0;

// System setup code ------------------------------------------------
-----------------------------------------------------------------
----------------------------------------------

void setup() {

  // Serial Port begin - To track the signals through Serial Monitor
and debugging
```

```
  Serial.begin (9600);

  // Define Inputs and Outputs of the system
  pinMode(trigPinFrontRight, OUTPUT);
  pinMode(echoPinFrontRight, INPUT);
  pinMode(trigPinFrontLeft, OUTPUT);
  pinMode(echoPinFrontLeft, INPUT);

  pinMode(infraredRightFront, INPUT);
  pinMode(infraredRightBack, INPUT);
  pinMode(infraredLeftFront, INPUT);
  pinMode(infraredLeftBack, INPUT);

  pinMode(rightEncPinA, INPUT_PULLUP); // Quadrature right encoder
input A
  pinMode(rightEncPinB, INPUT_PULLUP); // Quadrature right encoder
input B
  pinMode(leftEncPinA, INPUT_PULLUP); // Quadrature left encoder input
A
  pinMode(leftEncPinB, INPUT_PULLUP); // Quadrature left encoder input
B

  // Define interruptions for the encoder sensors
  attachInterrupt(digitalPinToInterrupt(21), updateRightEncoder,
RISING);
  attachInterrupt(digitalPinToInterrupt(19), updateLeftEncoder,
RISING);

  // Define PIDs types and parameters
  myPIDright.SetMode(AUTOMATIC);
  myPIDright.SetSampleTime(10);
  myPIDright.SetOutputLimits(0, MAX_SPEED);

  myPIDleft.SetMode(AUTOMATIC);
  myPIDleft.SetSampleTime(10);
  myPIDleft.SetOutputLimits(0, MAX_SPEED);

  // Initial states of the motors - Stopped
  motorRight.run(RELEASE);
  motorLeft.run(RELEASE);
  movState = "STOP";

}

// Main loop ----------------------------------------------------------
------------------------------------------------------------------------
-----------------------------------------------

void loop() {

  // Read the infrared sensors signals
  readIR(infraredRightFront, infraredRightBack, infraredLeftFront,
infraredLeftBack);

  // Read the Ultrasonic Sensors signals
  readUS(trigPinFrontRight, echoPinFrontRight, trigPinFrontLeft,
echoPinFrontLeft);

  // Select the next robot movement
  defineMove(distanceFrontRight, distanceFrontLeft, obstacleRF,
obstacleRB, obstacleLF, obstacleLB);
```

```
  delay(5);
}

// Function to set the FORWARD movement ------------------------------
-------------------------------------------------------------------
-----------------------------------------------

void goForward() {

  // Define desired setpoints for the PID to achieve
  setPointR = MAX_SPEED;
  setPointL = MAX_SPEED;

  PID_Computation(setPointR, setPointL);

  motorRight.run(FORWARD);
  motorLeft.run(FORWARD);

  // Update time variables
  previousMillis = currentMillis;
  encoderRightValue = 0;
  encoderLeftValue = 0;

}

// Function to STOP the motors ---------------------------------------
-------------------------------------------------------------------
-----------------------------------------------

void goStop() {

  if (movState == "STOP") {
    motorRight.run(RELEASE);
    motorLeft.run(RELEASE);
  }
  else {

    // Define desired setpoints for the PID to achieve
    setPointR = 0;
    setPointL = 0;

    PID_Computation(setPointR, setPointL);

    motorRight.run(FORWARD);
    motorLeft.run(FORWARD);

    delay(5);

    motorRight.run(RELEASE);
    motorLeft.run(RELEASE);
  }

  // Update time variables
  previousMillis = currentMillis;
  encoderRightValue = 0;
  encoderLeftValue = 0;

}
```

```
// Function to turn RIGHT ------------------------------------------
-----------------------------------------------------------------
-----------------------------------------------

void goRight() {

  if (movState == "RIGHT") {
    motorRight.run(RELEASE);
    motorLeft.run(FORWARD);
  }
  else {

    // Define desired setpoints for the PID to achieve
    // SKID STEERING - ONE WHEEL RELEASE, ONE WHEEL FORWARD
    setPointR = 0;
    setPointL = TURN_SPEED;

    PID_Computation(setPointR, setPointL);

    motorRight.run(FORWARD);
    motorLeft.run(FORWARD);

    delay(5);

    motorRight.run(RELEASE);
    motorLeft.run(FORWARD);
  }

  // Update time variables
  previousMillis = currentMillis;
  encoderRightValue = 0;
  encoderLeftValue = 0;

}

// Function to turn LEFT -------------------------------------------
-----------------------------------------------------------------
-----------------------------------------------

void goLeft() {

  if (movState == "LEFT") {
    motorRight.run(FORWARD);
    motorLeft.run(RELEASE);
  }
  else {

    // Define desired setpoints for the PID to achieve
    // SKID STEERING - ONE WHEEL RELEASE, ONE WHEEL FORWARD
    setPointR = TURN_SPEED;
    setPointL = 0;

    PID_Computation(setPointR, setPointL);

    motorRight.run(FORWARD);
    motorLeft.run(FORWARD);

    delay(5);

    motorRight.run(FORWARD);
    motorLeft.run(RELEASE);
```

```
  }

  // Update time variables
  previousMillis = currentMillis;
  encoderRightValue = 0;
  encoderLeftValue = 0;

}

// Function to define the type of movement the robot executes --------
-------------------------------------------------------------------
-----------------------------------------

void defineMove(long distanceFrontRight, long distanceFrontLeft, int
obstacleRF, int obstacleRB, int obstacleLF, int obstacleLB) {

  // Obstacle avoidance (priority)
  if ((obstacleRF == LOW) || (obstacleRB == LOW)) {
    goLeft();
    movState = "LEFT_OBST";
  }
  else if ((obstacleLF == LOW) || (obstacleLB == LOW)) {
    goRight();
    movState = "RIGHT_OBST";
  }
  else {
    if (distanceFrontRight > MAX_DISTANCE && distanceFrontLeft >
MAX_DISTANCE) {
      if ((movState == "FORWARD") && (count <= 5)) {
        goForward();
        count += 1;
        movState = "FORWARD";
      }
      else if ((movState == "RIGHT") && (count <= 5)) {
        goRight();
        count += 1;
        movState = "RIGHT";
      }
      else if ((movState == "LEFT") && (count <= 5)) {
        goLeft();
        count += 1;
        movState = "LEFT";
      }
      else {
        goStop();
        count = 0; // Reset counter
        movState = "STOP";
      }
    }
    else if (distanceFrontRight <= MIN_DISTANCE || distanceFrontLeft
<= MIN_DISTANCE) {
      goStop(); // Do not move
      count = 0; // Reset counter
      movState = "STOP";
    }
    else {
      if ((abs(distanceFrontRight - distanceFrontLeft)) <= 5) {
        goForward(); // Move forward
        count = 0; // Reset counter
        movState = "FORWARD";
      }
```

```
      else { // if ((abs(distanceFrontRight - distanceFrontLeft)) > 5)
        if (distanceFrontRight < distanceFrontLeft) {
          goRight(); // Turn right
          count = 0; // Reset counter
          movState = "RIGHT";
        }
        else { // if (distanceFrontRight > distanceFrontLeft)
          goLeft(); // Turn left
          count = 0; // Reset counter
          movState = "LEFT";
        }
      }
    }
  }
}

// Function to generate and read the signals from the ultrasonic
sensors ------------------------------------------------------------
-------------------------------------------------

void readUS(int trigPin1, int echoPin1, int trigPin2, int echoPin2) {

  // Generate ultrasonic wave for sensor 1
  digitalWrite(trigPin1, LOW); // Short LOW pulse to ensure clean HIGH
pulse.
  delayMicroseconds(5);
  digitalWrite(trigPin1, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin1, LOW);

  // Read the signal
  duration = pulseIn(echoPin1, HIGH);
  distance = (duration / 2) / 29.1; // Convert time into distance[cm]

  distanceFrontRight = distance;

  // Generate ultrasonic wave for sensor 2
  digitalWrite(trigPin2, LOW); // Short LOW pulse to ensure clean HIGH
pulse.
  delayMicroseconds(5);
  digitalWrite(trigPin2, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin2, LOW);

  // Read the signal
  duration = pulseIn(echoPin2, HIGH);
  distance = (duration / 2) / 29.1; // Convert time into distance[cm]

  distanceFrontLeft = distance;

  delay(5);
}

// Function to read the signals from the infrared sensors ------------
-------------------------------------------------------------------
-----------------------------------------------

void readIR(int IF_RF, int IF_RB, int IF_LF, int IF_LB) {
  obstacleRF = digitalRead(IF_RF);
  obstacleRB = digitalRead(IF_RB);
  obstacleLF = digitalRead(IF_LF);
```

```
  obstacleLB = digitalRead(IF_LB);
}

// Function to update the right motor encoder position; it is called
at interruption event ----------------------------------------------
-----------------------------------------------

void updateRightEncoder()
{
  // Add encoderValue by 1, each time it detects a rising signal
  // from hall sensor A
  int val_1 = digitalRead(rightEncPinB);
  if (val_1 == LOW) {
    encoderRightValue++;
  }
  else {
    encoderRightValue--;
  }
}

// Function to update the left motor encoder position; it is called at
interruption event --------------------------------------------------
-----------------------------------------------

void updateLeftEncoder()
{
  // Add encoderValue by 1, each time it detects a rising signal
  // from hall sensor A
  int val_2 = digitalRead(leftEncPinB);
  if (val_2 == HIGH) {
    encoderLeftValue++;
  }
  else {
    encoderLeftValue--;
  }
}

// Function to make the calculations for setting the PID output values
-------------------------------------------------------------------------
-----------------------------------------------

void PID_Computation(double setPointR, double setPointL) {

  currentMillis = millis(); // Current running time

  temp = (currentMillis - previousMillis); // Interval of time that
the encoder has been measuring since last robot movement

  // Calculate RPM of each motor from the pulses which are measured by
the encoders
  rpmRightMotor = (float)(((encoderRightValue) * 1000 * 60) / (temp *
ENCODER));
  rpmLeftMotor = (float)(((encoderLeftValue) * 1000 * 60) / (temp *
ENCODER));

  // Take as PID inputs the measured RPM at which each motor is
running
  inputR = rpmRightMotor;
  inputL = rpmLeftMotor;

  myPIDright.Compute();
```

```
  myPIDleft.Compute();

  // Define as the motors speed
  speedSetRight = int round (outputR * RIGHTCOEFF); // outputR in RPM
and converted to PWM
  speedSetLeft = int round (outputL * LEFTCOEFF); // outputL in RPM
and converted to PWM

  motorRight.setSpeed(speedSetRight);
  motorLeft.setSpeed(speedSetLeft);

  delay(5);
}
```