# Car Race

**Hussein Amin Givá**

Thesis to obtain the Master of Science Degree in

**Computer Science and Engineering**

Supervisor: Prof. Alberto Manuel Ramos da Cunha

**Examination Committee**

Chairperson: Prof. Daniel Jorge Viegas Gonçalves
Supervisor: Prof. Alberto Manuel Ramos da Cunha
Member of the Committee: Prof. Teresa Maria Sá Ferreira Vazão Vasques

**June 2021**

# Acknowledgments

I would like to thank Professor Alberto Cunha for his guidance and support during the elaboration of this thesis. His valuable advice and concise input were fundamental for the proper development of the work being presented.

I would also like to extend my gratitude to my parents Amin and Safira, my brother Karim, my uncle Salim and my girlfriend Inaara for always supporting me and not letting me give up when times were hard.

Finally, one word of appreciation for my friends, family, colleagues and professors that motivated me and made me realize I made the correct choice regarding my degree.

## Resumo

A introdução de robôs móveis na educação traduz-se em novas possibilidades de aprendizagem, para alunos que estejam a ser apresentados a temas como robótica e internet das coisas pela primeira vez e para alunos a desenvolverem projetos com âmbitos profissionais e de pesquisa. Utilizadores mais avançados requerem robôs mais precisos e com características especificas para a tarefa em questão mas, neste trabalho, o foco são os estudantes a iniciar os seus estudos na área, que beneficiam de um robô mais generalista e mais didático que potencie a sua aprendizagem.

Este trabalho tem como primeiro objetivo fornecer uma análise completa sobre a operação do robô móvel Waveshare Alphabot2-Ar e criar uma base sólida para o correto funcionamento do robô, nas suas possibilidades de operação. O objetivo final deste trabalho passa pela criação de guias de laboratório que permitam aos alunos desenvolver programas para controlo do robô de forma idêntica ao obtido durante o desenvolvimento deste trabalho.

Primeiramente é feita uma análise detalhada e completa do robô e de seguida são expostos os conteúdos que constituem os guias de laboratórios desenvolvidos juntamente com este trabalho. Após esta análise, são apresentados os resultados da aplicação do código desenvolvido, que constitui a resolução de cada laboratório, em 2 pistas desenvolvidas para avaliação e análise das soluções propostas e adequação dos guias de laboratório. Os resultados comprovaram a eficácia do código desenvolvido em seguir a linha da pista, detetar e evitar choques com obstáculos e uma interação robusta e completa com os periféricos do robô.

**Palavras-chave:** Robô móvel, Alphabot2-Ar, Arduino, Seguimento de linha, Deteção de obstáculos

## Abstract

The introduction of mobile robots in education translates into new learning possibilities for students who are being initiated on topics such as robotics and the internet of things for the first time and for students developing projects with professional and research scopes. For this work, the focus lies on students starting their studies in the area and that benefit from a generalist and didactic robot that enhances their learning.

This work's objective is to provide a complete analysis on the operation of the Waveshare Alphabot2-Ar robot and to create a solid base for the correct functioning of the robot, in all its operating possibilities. The final objective of this work is the creation of laboratory guides that allow students to create programs that control the robot in a similar way to the obtained code produced during the development of this work.

First, a detailed analysis of the robot is made and the contents that make up the laboratory guides, developed alongside this work, are exposed. After this analysis, the results of the application of the developed code, which is constituted by a resolution of each of the laboratory guides, are shown. The developed code, alongside the respective laboratory guides, were tested in 2 racetracks, developed for the evaluation and analysis of the proposed solutions and the adequacy of the laboratory guides. The results prove the efficacy of the developed code in line following, detection and avoidance of obstacles and a robust and complete interaction with the robot's peripherals.

**Keywords:** Mobile robot, Alphabot2-Ar, Arduino, Line following, Obstacle detection

# Contents

# List of Tables

# List of Figures

# Nomenclature

| | |
|---|---|
| $ACIC$ | Applications and Computation for the Internet of Things. |
| $ADC$ | Analog-Digital Converter. |
| $AGV$ | Autonomous Guided Vehicles. |
| $AI$ | Artificial Intelligence. |
| $AMR$ | Autonomous Mobile Robot. |
| $CMU$ | Carnegie Mellon University. |
| $EEPROM$ | Electrically-Erasable Programmable Read-Only Memory. |
| $EOC$ | End Of Conversion. |
| $EOF$ | End Of File. |
| $I2C$ | Inter-Integrated Circuit. |
| $IDE$ | Integrated Development Environment. |
| $I/O$ | Input/Output. |
| $IoT$ | Internet of Things. |
| $IR$ | Infrared. |
| $ISR$ | Institute for Systems and Robotics. |
| $IST$ | Instituto Superior Técnico. |
| $LE$ | Low Energy. |
| $LED$ | Light Emitting Diode. |
| $MRAS$ | Mobile Robots and Autonomous Systems. |
| $MSB$ | Most Significant Bit. |
| $OLED$ | Organic Light Emitting Diode. |
| $PCB$ | Printed Circuit Board. |

|  |  |
|---|---|
| $PID$ | Proportional, Integral and Derivative. |
| $PWM$ | Pulse Width Modulation. |

**Roman symbols**

|  |  |
|---|---|
| $RAM$ | Random-Access Memory. |
| $RGB$ | Red, Green and Blue. |
| $R/W$ | Read/Write. |

**Subscripts**

|  |  |
|---|---|
| $SCL$ | Serial Clock Pin. |
| $SDA$ | Serial Data Pin. |
| $SPI$ | Serial Peripheral Interface. |
| $SRAM$ | Static Random-Access Memory. |
| $SSD$ | Solid State Drive. |
| $UK$ | United Kingdom. |
| $USB$ | Universal Serial Bus. |

# Chapter 1

# Introduction

Over the last few years, mobile robotics has become an increasingly more important area in the world of robotics. Newly developed mobile robots have played an important role in areas such as scientific research, industrial use, planetary exploration, military use, personal service and medical use [1]. One common aspect between mobile robots used in the most diverse of fields is the fact that all of them rely on actuators to move around and a set of sensors to sense their surroundings and send data to make calculations regarding the robots' next movements [1]. Despite this common ground between mobile robots, the actuators used vary greatly from robot to robot and also between the different areas in which they are used. Depending on the mean where the robots are circulating, they may rely on wheels, caterpillars or even wings to move around. And the same occurs with their sensor set, where physical limitations, price and use case objectives dictate which sensors should be used. However, one constant is the fact that mobile robots operate in a more or less autonomous way, meaning without the need for human interference [1].

For this work, the focus relies on the educational experience that mobile robots can provide. In particular, an analysis of one specific robot is made, including seeing how well it integrates into a pedagogic environment. As mobile robots become increasingly more advanced and capable machines, the more difficult it becomes for beginners to learn the basis behind the operation of said robots. With that in mind, the need for a good and simple mobile robot, with easy to use characteristics and an educational feature set, as well as an affordable price, arises. Alongside such a robot, an introductory course that takes advantage of the characteristics of the robot and that deploys a correct theoretical approach to the field also has to exist. As Demetriou [2] points out, "...using robots to teach courses that are directly related to robotics. These courses are usually introductory robotics courses and teach the basic concepts of mobile robotics. These courses are usually divided into lectures and laboratory sessions. At the lecture sessions students learn concepts such as kinematics, perception, localization, map building and navigation. At the laboratory sessions, students experiment on real or simulated robots.".

With that in mind, it becomes clear that the goal of this work is to not only analyse the mobile robot in discussion but to also prepare content that allows students, who are approaching the subject for the first time, to have a complete yet practical introduction to the topic of mobile robots. This introduction must be divided into theoretical and practical components, with the theoretical part targeting the teaching of the necessary concepts to operate the components of the robot correctly and the practical part to show the students the implementation and results of their work. This is achievable through the use of a race track where they will see the behaviour of the Waveshare Alphabot2-Ar wheeled mobile robot [3] in action and where they can understand the effects of having the robot programmed in a correct way but also in a not so correct way. For them to achieve a correct setup of the robot, they must master the theoretical content that was introduced to them.

To measure the accuracy of the movement of the mobile robot and the adequacy of the work being requested from the students, solutions to the laboratory guides were developed and measured for their reliability, accuracy, speed and overall behavior. To achieve these measurements, the robot performed laps over two racetracks, whose design was developed to test different aspects of the robot and the solutions, where times were taken, deviations from the race line were analyzed and response to commands and obstacles were captured.

The basis for the laboratory guides was to have the students, who are taking an introductory class to $IoT$ devices and components, understand the working of the components that constitute the mobile robot in use and to apply the learnings in the real robot, through the programming of the Arduino board that constitutes the processing unit of the robot. The basic tasks that the students need to program consist of having the robot follow a black line over a white surface, to have it react to obstacles using the robot's sensors and to respond to commands given by the user in real-time, while also taking advantage of the robot's peripherals to increase the quality of the experience of working with this robot. As a final task, after completing the laboratory guides, it is asked for the students to compete between them in a race, to see which team has programmed the best solution. This fun and didactic approach aims to increase the interest and quality of interaction between the students, the robot and the learning materials.

## 1.1 Motivation

As the field of robotics keeps evolving and growing, new and better technologies start to get integrated into robots. The area of mobile robots is just one of the sub-fields where this reality happens. And this growth in capabilities and potential for robotics attracts a greater interest from scholars, professionals and enthusiasts. This translates into an ever growing need for a solid and complete introduction to the area from the behalf of the schools. As Crnokić et al. [4] pointed out, "Educational robotics is primarily focused on creating a robot that will help users to develop more practical, didactic, cognitive and motor skills. This approach is intended also to stimulate interest for research and science through set of different activities designed to support strengthening of specific areas of knowledge and skills that the students develop through design, creation, assembly and operating with a robot.".

This means that a more diverse and practical approach to introductory students in engineering schools, through the use of educational robots, with the purpose of developing the necessary skills for these students to handle more complex and complete robotic systems, must be done. However, in order to engage in such an approach, schools must be ready to support the students in their learning, both by having adequate materials and also by teaching the necessary subjects. This is the main motivation behind this work.

After having analyzed multiple schools, their curriculum regarding robotics learning, their robots and hardware material as well as their current projects and investigations, a development plan was drawn. Analysis of universities such as Aalto University [5], in Finland, and ParisTech Institute [6], in France, shows that that European schools, $IST$ partners, have a significant presence in leading investigations surrounding mobile robots but, at the same time, lack in terms of introductory courses and educational robots meant to be used by inexperienced students.

This motivated the development of this work. By having students with an early contact with mobile robotics and by having appropriate hardware for a complete educational experience, it is expected that students will further develop their interest in the field as well as be better prepared for more advanced and complex research projects, like the ones currently being developed by the $ISR$ [7]. Such projects include, but are not limited to, “Accurate Agent Localization and Mapping Methods for Structured Indoor Retail Store Environments” [8], “Computer Vision for the Operation of Unmanned Aerial Vehicles in Maritime and WildfIre Scenarios” [9], “Autonomous mobile robot programming using ROS on a remote web-based real robot lab” [10] and “Developmental Learning of Internal Models for Robotic Manipulation based on Motor Primitives and Multisensory Integration” [11].

## 1.2 Topic Overview

Mobile robotics is one of the fastest growing fields in the world of robotics. Over the last few years, mobile robots have been used for the most diverse of use cases, such as planetary exploration, military use, medical research, industrial use, among others [1]. For all of these areas, mobile robots have had special designs to meet the necessary criteria in order to have an appropriate usage. This made them increasingly more complicated and advanced. Such examples include the exploratory mobile robots developed by Boston Dynamics [12] and also the vacuum mobile robots developed by Roomba [13].

However, these robots do not fill the necessities presented by schools and universities. Unlike the robots being used in professional and research activities, which present expensive and highly delicate materials and components, educational and pedagogical robots must be durable and present an easier to use interface. However, it is important for them to retain the main characteristics that make up a mobile robot.

Starting with the sensors, mobile robots targeted for education should always include a sensor set that encompasses the main technologies used by mobile robots to navigate around their environment. These include, but are not limited to, line detecting infrared sensors, obstacle detecting infrared sensors, obstacle detecting ultrasonic sensors and a camera for image processing.

The infrared line detecting sensors are usually placed on the underside of the robot and are used to track a line over a contrasting background. They are usually used in quantities from 3 to 5 sensors.

Obstacle detecting infrared sensors are used to detect obstacles by having an infrared light beam emitted and, if the reflected ray is detected beneath a given threshold, an obstacle is detected. Setup and configuration of these sensors vary from mobile robot to mobile robot but the inclusion of at least one sensor is necessary to assure students get hands-on experience with this technology.

Obstacle detecting ultrasonic sensors present a similar behaviour to infrared obstacle detecting sensors but with one key difference, they use sound waves instead of light ones. This allows the sensor to return not only the presence of an obstacle but also an easier calculation of the distance to which that obstacle is. With this distance value, a more complete and detailed operation can be programmed for the robot.

Finally, for the sensors that are the most crucial for the autonomous/semi-autonomous behaviour of the robot, there is the camera. A camera adds a new dimension to the data retrieved by the robot by allowing the robot to identify multiple objects and obstacles in its vicinity and distinguish individual entities. This is useful for when the robot needs to perform human/object recognition, track limit detection or even to detect the line to follow, in order to steer its movement. However, due to the fact that using a camera requires image processing and pattern detection algorithms, a high quality camera to make the robot behavior precise and a powerful enough processor to process the data sent by the camera, most educational mobile robots can skip this sensor in their sensor set. Since almost all navigation tasks the camera can perform can also be done by the sensors mentioned above, a camera ultimately becomes an additional extra to further extend and enhance the learning experience.

Moving on to the actuators, educational and pedagogical robots must have a basic set of actuators that are adequate for the mean on which the robot is navigating. This means a propeller for water moving robots, wings for air moving robots and wheels (or possibly caterpillars or legs) for land moving robots. Simple and durable controllers for these components should be installed, as to withstand errors from the behalf of the students such as negative speeds and wrong motion commands.

Finally, when it comes to the processing unit of the mobile robot, educational robot should have a relatively common processing unit, as to ease research and available material. These processing units should be powerful enough to control the entire mobile robot, durable enough to support errors from the user as well as inexpensive enough so the schools can afford the board and the robot. The most commonly used ones are Arduino boards and RasperryPi boards [14]. The $I/O$ characteristics of these boards also make them ideal to use in educational robot robots, as these boards are already used in educational environments and present a panoply of accessories and components like the sensors and actuators mentioned above, with ideal characteristics for this type of robots.

Due to these differences in types of sensors, actuators and processing boars, a division had to be made between the different kinds of mobile robots. According to Siegwart and Nourbakhsh [15], the categories are differentiated according to 4 sections: locomotion; perception, cognition and navigation.

Locomotion refers to what enables mobile robots to move. This division is first made between the surface on which the robot moves (water, air or land) and then for land-based robots, on whether they move using wheels, legs or other means of motion [1].

Then, for the perception abilities, the robots can be divided based on the types of sensors they have [1]. They can be tactile sensors, infrared sensors, ultrasonic sensors, sonar, among others [1].

For the cognition division, the robots are split according to the way they create their cognitive model, that allows them to make autonomous decisions and the way they act to execute the operations calculated in the cognitive model [1].

Finally, robots can be divided according to their navigation abilities. These include the robots' capabilities to locate themselves, the capability to plan a trajectory, to map their surroundings and to avoid obstacles, among others [1].

Both cognition and navigation divisions can be summarized into a single more generalized division: $AMR$s and $AGV$s [16].

Autonomous Mobile Robots are robots which are able to navigate their surroundings without the need for instructions or external guidance devices [16]. As such, they are able to explore their surroundings freely [16].

Autonomous Guided Vehicles are robots which require external guidance to be able to move around their environment [16]. This external guidance can be provided by magnets, wires, sensors or even tape to map where the robots are allowed to move. By using these physical parts to limit the robots' movements, the mobility and autonomy of the robots is decreased, while also making the development more expensive and difficult to achieve [16]. However, this type of robots is easier to control, contain and also to debug. As such, this document will focus on developments made to an $AGV$.

The robot model being used in this work is actually capable of being both an $AMR$ and an $AGV$, depending on the software controlling its behavior. It possesses the sensors and computational power to have software that enables it to roam freely and in an autonomous way. However, it also has the sensors needed for it to act as an $AGV$, if the software running on it so dictates.

## 1.3 Objectives and Deliverables

After this analytical and development work, several objectives were drawn, and expected to be achieved.

The first objective is to provide an in-depth report on the functioning of the Waveshare Alphabot2-Ar and of its components. This report is integrated into this thesis document and explicits the different components that make up the mobile robot, how the system is assembled, how the peripherals interact with each other, how the Arduino board controls the peripherals, among several other aspects.

After this analysis, the adequacy of this robot to be used in an educational environment with students of introductory robotics and $IoT$ classes is testified with the development of laboratory guides. These guides constitute another deliverable objective of this work - develop a platform for the use of mobile robots in an educational laboratory environment. Through the development of these laboratory guides, it is understandable that the robot does fit the educational panorama and the requirements for the application of the contents taught in these introductory classes. These laboratory guides will serve as base for the development of educational activities that show how this specific mobile robot can be operated and also how mobile robots are used to perform given specific tasks. These laboratory guides consist in a theoretical introduction followed by feature development tasks so that the students are able to successfully code the Arduino boards to have the robots perform tasks such as following a black line over a white background and avoiding obstacles using their sensors and also to use all their peripherals to increase the functionalities of the robot and the quality of the experience.

The pedagogical objective of this work and of these laboratory guides is to introduce students to concepts such as mobile robotics, real-time system control, communication protocols, hardware and software components used in computational systems, control algorithms as well as the difference in programming and optimization needs when working with low power hardware. For this to happen, each of the developed laboratory guides serves a different purpose.

Laboratory guide 0 [17] is meant to help students identify each of the components present in the Waveshare Alphabot2-Ar and show them how to assemble the components correctly in order to have the robot fully functioning. With this laboratory guide, it is expected that students learn about the correct way of assembling such delicate components as well as having them learn about how each component is connected in a more advanced system like a complete mobile robot.

Laboratory guide 1 [18] is meant to have students make the mobile robot follow a single black line placed over a white background, by utilizing the $IR$ line detecting sensors placed beneath the robot. With this laboratory guide, it is expected that students learn about interaction between components in an Arduino environment, how to access an $ADC$ and how it works and also learn about system control algorithms, including how to fine tune the $PID$ multiplier parameters (KP, KI and KD).

Laboratory guide 2 [19] is meant to have students make the mobile robot detect and avoid obstacles placed in front of it, first by reducing the robot speed and then by stopping the robot entirely. With this laboratory guide, it is expected that students learn about communication protocols such as $I2C$, how to use external libraries in the Arduino platform, the functioning of the $IR$ and ultrasonic sensors attached to the Arduino (and how to retrieve data from those sensors) as well as how to use that data to change the behaviour of the robot in real time.

Finally laboratory guide 3 [20] is meant to have students make the mobile robot interact with its peripherals, both internal and external. Through the interaction with these peripherals, the robot should change its behavior depending on the input received and communicate, in real time, with several components such as an $OLED$ display and $LED$s. With this laboratory guide, it is expected that students learn about some more complex external libraries and their functioning, some Arduino specific functions and their purpose, how to calculate battery voltages and the voltage flowing through the Arduino board as well as to how to process commands sent in real time to the Arduino, both via the $IR$ remote control as well as the joystick.

One other objective expected to be achieved with this work is the creation of a solid base for the operation of the robot. This base must operate all the components of the robot in a way that serves perfectly to demonstrate the capabilities of the robot when it is correctly programmed and assembled. Alongside these purposes, one other purpose of this base is to serve as platform for future work developments being done on this robot, specifically over machine learning and artificial intelligence solutions. This base is actually achieved with the third laboratory guide [20], where the solution specifies a complete program that controls all the peripherals of the robot, all sensors and all actuators. Given the fact that students may not achieved the optimal solution for each of the laboratory guides, alongside each laboratory guide, a complete solution, in the form of an Arduino program for the guide, is provided. These solutions implement all the requested features of the laboratory guide and can be used as comparison base and as a benchmark to grade the programs created by the students.

The third and final objective of this work is an analysis of the performance of the developed code. For that, measurements were made and comparisons between different sets of parameters to show the efficacy of the proposed solutions.

Alongside this code base, one other deliverable expected is the configuration of a set of racetracks that, due to their design, test important parameters of the robot, such as cornering capabilities, line detection ambiguity, obstacle avoidance, straight line speed and stability. Alongside the design of the tracks, their characteristics (materials, shape, colors and lane design) and the configurations of the robot (sensors, actuators and computing hardware) also make up part of the deliverables.

## 1.4 Thesis Outline

Over the next chapters, the following contents will be presented.

First, in chapter 2, an analysis of the background and the state of the art is made. This analysis shows the current mobile robots used in education, based on Arduino and similar platforms and the different deployments of mobile robots in line following projects in an educational scope. By doing this, it is perceivable the current status of the world of mobile robotics in an introductory educational panorama as well as the different implementations being developed to solve the line following problem in a similar setup to the one presented in this document.

Secondly, in chapter 3, it is detailed the analysis of the Waveshare Alphabot2-Ar used in this work. The components present in this robot are listed, the assembly of the robot and its components is explained and the functioning of the components themselves is explained.

Thirdly, in chapter 4, the implementation of the laboratory guides and respective solutions is exposed. For each laboratory guide, it is presented the theoretical introduction that supports the guide, the practical activities requested by the students as well as the questions the students must answer.

For chapter 5, the results of the application of the code into the robot are presented. To do so, the code of the final laboratory is tested around the two test tracks, where time and precision measurements are made to evaluate how adequate the solution is.

Finally, in chapter 6, the conclusions of the work are presented. In these conclusions, a final evaluation of the chosen mobile robot is made, indicating its strengths and weaknesses, specially when it comes to a pedagogical use. Conclusions regarding the laboratory guides and the consequent code solutions are also made, where it is discussed whether the laboratory guides are adequate for the intended use and also whether the coded solutions have a satisfying behaviour and are acceptable to be used as benchmark on which other written solutions can be compared to. Aside from these conclusions, the main achievements of the work developed are exposed, specially when it comes to the comparison between the code developed by Waveshare to run the robot and the code developed during this work to control the robot. Lastly, possible future work that can use the achieved solution is calculated, specially regarding natural extensions to the behaviour of the robot. A special focus on machine learning and artificial intelligence is made.

Alongside the development of this document, laboratory guides were developed and, delivered alongside this document, are also the coded solutions for each of the laboratories. The laboratory guides can be found in [17–20] while the coded solutions can be found in [21–23]. The base code files, for the laboratory guides, that are handed to the students can be found in [24–26].

# Chapter 2

# Background

Before starting to work with the present mobile robot, the Waveshare Alphabot2-Ar, it was necessary to know and understand whether this robot fits the current standard of mobile robots used in education nowadays. For this, the robot needed to have a similar sensor set and actuator set to the ones used in current introductory robotic classes in reference universities but it also needed to have a platform that was easily taught and understood by the students while still being powerful enough and diverse enough to support the behaviours that the mobile robot is intended to have in the classes being taught in $IST$ that make use of this robot.

This led to the necessity to also understand not only how the paradigm is nowadays when it comes to mobile robotics in introductory classes in higher education but also to understand how are current mobile robots being developed, setup and programmed to do the main task that this robot is asked to do: line following. For that, 2 theoretical thesis are proposed that are backed up by several papers presented below.

## 2.1 Theoretical Overview

For the mobile robot being used to be adequate for an educational and pedagogical use, it must comply with certain parameters and specifications. Above all else, it must not deviate too much from the norm that guides other universities in similar courses but must also fit and be ideal to use in the courses that plan on utilizing it within $IST$.

So, this meant it was necessary to understand what other schools are doing in terms of their curriculum structure for these courses and what robots they are using to fulfill these curricular plans.

In order to establish a baseline, it was necessary to understand what types of courses were being taught in other schools, the curriculum these courses have and the robotic platforms used. Understanding the technical and theoretical concepts is also important, as to see if they match the ones taught in $IST$, in these courses. Seeing if the robotic platforms present similarities also allows to reach conclusions on what benefits our mobile robot has versus the other ones but also what downsides this one has that may lead to future upgrades to this mobile robotic platform. For this analysis of the robotic platform, all aspects of the robot must be considered, alongside the interfaces used to communicate with the robot.

Not only this, but the robot must be diverse enough in terms of functionality to be used in a variety of scenarios and to perform a multitude of tasks. This ensures the attractiveness of the robot remains high for the duration of the school year. It also ensures different projects can be run on the robot and different theoretical concepts can be evaluated using the robot. In this particular case, the main functionalities being presented with this robot are line following and obstacle detection. With this in mind, it becomes necessary to know how other robots with similar hardware and software are performing these 2 tasks and understand whether those ways are the ideal ways to show and teach the students.

## 2.2 Theoretical Model 1

The first theoretical model compiled states that, for robotic introductory courses, the theoretical contents being taught in other schools are in line with the contents proposed in the developed laboratory guides and that the commonly used type of mobile robot is pretty similar to the one being proposed to be used in this work, the Waveshare Alphabot2-Ar.

The similarities within the course structure itself consist in the fact that these robotic introductory courses are usually divided into theoretical and laboratory classes; targeted at students with few or no experience at all handling robots and where the laboratory work is executed upon the results of the work from the previous laboratory session, with evaluations in the end of each laboratory guide.

Concerning the curriculum, the contents taught include perception (localization using sensors - in the case of this document, the sensors used are infrared obstacle detection, infrared line detection and ultrasonic obstacle detection - and filtering methods - in the case of this document, data normalization and weighted average) and action (reactive control - in the case of this document, using the $PID$ algorithm and obstacle detection and avoidance algorithms).

Finally, concerning the robot, a platform that moves on wheels, that processes data within its structure and that is used in other schools is on par with the robot used here. The sensor set normally comprises sonar, infrared and a camera. Aside from the camera, all are present in the Alphabot2-Ar, with the addition of an $IR$ receiver. Aside from these sensors, the actuators usually comprise 2 or more wheels, which is on par with the Alphabot2-Ar. Usually, other peripherals such as screens and network boards may or may not be present in robots commonly used (depends on the specific robots used), but joysticks and buzzers are not common at all, even though the Alphabot2-Ar has one of each. The papers presented below support this researched thesis.

An analysis of the Mobile Robot Programming Laboratory course at $CMU$, done by Lalonde et al. [27], shows a multitude of similarities in the course outline, when comparing to the material developed by this document to be used in the $ACIC$ [28] course in $IST$, including course logistics, robotic platform, course curriculum and theoretical/practical content taught in the laboratory classes.

Beginning with the course logistics, that, upon presenting similarities with the $ACIC$ course, create a basis for a valid comparison on the other aspects, Lalonde et al. [27] declares that ”Mobile Robot

Programming Laboratory is a one-semester course that uses a problem-driven syllabus to give students hands-on experience with mobile robot programming. The class is kept to a manageable target size of 30 students, with the support of 2 teaching assistants. It is designed for students with little or no experience with robots, and it is available to senior undergraduates in Computer Science or new graduates in Robotics.". This is similar to $ACIC$, that also consists of a one-semester course that gives students hands-on experience with mobile robot programming (this last part will begin in the school year 2021/2022). It is also a problem driven course, where the practical component is used to develop a solution to problems present on laboratory guides. This solution is then improved upon with the next laboratory guide, where new problems are presented and must be solved using the previous solution as base.

Similar to the Mobile Robot Programming Laboratory course at Carnegie Mellon University, also $ACIC$ at $IST$ also has circa 30 students in size and is designed as an introductory course for students beginning their Masters' degree in Computer Science and Engineering, with a minor in Robotics or Computational Systems. Similar to the Mobile Robot Programming Laboratory course, where Lalonde et al. [27] states that teams of 3 students are formed to be maintained throughout the semester and that work together in the laboratory guides, the same occurs for $ACIC$. A weekly lecture is also part of both courses, which "aims to enrich students' theoretical knowledge of important concepts in robotics." [27].

Regarding the robotic platform, the Mobile Robot Programming Laboratory course uses a small, wheel-driven, mobile robot. This robot presents, as sensors, a webcam and 16 sonars. In the case of the Alphabot2-Ar, it does not have a camera (although one can be attached to the Arduino board and used) but it does have a line sensing infrared sensor that, for the maze activity in which the Carnegie Mellon robot participates, the Alphabot2-Ar can also manage to engage. The difference being that, instead of using a camera to detect "gold in front of the robot (blue cardboard attached to a metal base)" [27] , which identifies the final contest location, it can use its line sensing hardware to sense a final location placed in a contrasting color to the background on which is drawn.

Since the Alphabot2-Ar also uses ultrasonic sensors, the wall detection can also occur, although the difference in number of sensors used by the Carnegie Mellon robot and the $IST$ robot makes the Carnegie Mellon one better at the task of identifying walls surrounding the robot. However, a different interpretation of the maze activity can be developed to target specifically the Alphabot2-Ar. To do so, the maze must be drawn over a background instead of being composed by a 3D structure.

The use of the sonars is similar in both robots, being used to measure distances and detecting obstacles. Over the sonar sensors present in the Carnegie Mellon robot, the Alphabot2-Ar also has infrared obstacle detecting sensors to detect obstacles that are too close for the ultrasonic one to be accurate. In terms of the actuators, both robots present 2 motorized wheels.

The main difference comes with the processing unit of the robots, where the Alphabot2-Ar uses Arduino (a Raspberry Pi unit is also available) while the Carnegie Mellon one uses a laptop running Windows XP. Both the Arduino and the laptop support serial communication and network connectivity, thanks to network adapter boards. The Alphabot2-Ar has, as peripherals, a screen (smaller than the laptop screen used in the $CMU$ robot), a joystick, a buzzer and an $IR$ receiver. All these peripherals can be emulated or added to the laptop in the $CMU$ robot. However, an in-depth analysis has shown that using a laptop as the processing unit in these type of courses is not the norm. Instead, these introductory courses tend to rely on Arduino and Raspberry-Pi boards, as exemplified by the uses in [29–31].

As for the course curriculum and theoretical/practical content, the Mobile Robot Programming Laboratory course presents an approach where the technical concepts are divided into the categories of perception, action and cognition. In the category of perception, the work relies on having students understanding the different sensors and how to correctly obtain data from them. In here, students must ”deal with the inherent uncertainty in sensing, learn about the physics governing the sensors used, and implement filtering methods to get high-accuracy localization despite noisy raw data.” [27].

Similarly, the laboratory guides developed in this work also focus on obtaining data from sensors and treating that data. To do this, students will learn about $ADC$, $I2C$ with serial interface, data normalization and weighted averages. In the case of the work asked in the laboratory guides, students use the infrared line sensors to detect a line over a contrasting background and use the infrared obstacle detection sensors and the ultrasonic obstacle detection sensors to detect obstacles and measure distances.

The second topic approached by the $CMU$ course, and that the developed laboratory guides also acknowledge and approach, is the action topic. In the action topic, the gathered data must, after having already been treated, be used to guide the movement of the robot. Here, both the $CMU$ course and the laboratory guides developed approach the subject using the same method, the $PID$ algorithm. The action both robots are being asked to perform is similar, where the $CMU$ course expects the robot to go through a maze and the $IST$ one asks the mobile robot to go around a track, delimited by a line that the robot must follow.

Finally, one topic approached by the $CMU$ course but not by the work developed in this document is the cognition topic, theoretical and practical, that is included in the course curriculum. In this topic, work that uses $AI$ is developed to make the robot take autonomous decisions. A similar situation is proposed in the future work section of this document. An AI solution to control the behaviour of the robot was not implemented in this work because there simply was not enough memory or storage in the robot to store data from the sensors regarding each detail of the track, alongside the fact that the robot does not have a high enough processing power to create a model of the track based on the data to allow for comparisons of the data recorded by the sensors to data that matches each specific part of the track (alongside real-time control of all the components that make up the Alphabot2-Ar). When mentioning data, the considered data is the speed of the robot and the information regarding $IR$ reflectivity obtained by each of the 5 $IR$ line detecting sensors. Without this data stored somewhere, it is impossible for the robot to know its position on the track and, therefore, make changes in its behavior based on $AI$ and machine learning algorithms.

Further advanced concepts are proposed by the $CMU$ course, such as robot observability, property mapping, reinforcement learning and multi-robot coordination. However, due to the fact that the Alphabot2-Ar does not have a camera shipped with it and it uses a less powerful Arduino board, when compared to the Windows XP laptop that the $CMU$ robot used, these topics are not approached neither on these laboratory guides nor in the $ACIC$ course. The fact that these topics deviate from the purpose of the $ACIC$ course also deter them from being taught and included in the laboratory work.

In the University of Manchester [29], $UK$, a similar course to $ACIC$ is also deployed, where the students use a mobile robot to cover "topics such as mobile robot kinematics (legged and wheeled), sensing & perception, planning & navigation and estimation & filtering" [29].

In this course, a theoretical lecture approach is combined with practical lectures, similar to $ACIC$. Also like $ACIC$, the $MRAS$ undergraduate course also has a low cost, small-scale robotic platform to be used by the students. The robot used is the Mona robot [29], developed in-house and that presents tremendous similarities to the Alphabot2-Ar.

Justifying the choice for a wheeled mobile robot, Arvin et al. [29] present a similar argument to the one that compelled the choice in this document, where the fact that the difference between legged mobile robots and wheeled mobile robots comes down to only the means of locomotion, and that since all other subsystems can be considered as independent when it comes to educational purposes, a wheeled mobile robot offers a low cost alternative for teaching. The uses this Mona robot, when it comes to the curriculum of the $MRAS$ course, does not deviate too much from the topics approached on the laboratory guides developed. They consist in experiments that involve sensors and actuators to learn about "open- and closed-loop control, obstacle detection and avoidance and more complex swarm algorithms." [29]. The same is true for the laboratory guides developed for $ACIC$, where obstacle detection and avoidance is present as a theoretical and practical subject to be worked on, as is open- and closed-loop control.

Absent from the laboratory guides is the topic of swarm algorithms. This is because multi-robot collaboration is not a topic to be approached by $ACIC$. So, as analyzed, the course structure and curriculum for the use of mobile robots in $ACIC$ and $MRAS$ present numerous similarities.

The other aspect that also confirms our hypothesised theoretical model is the similarity between robot models, in this case the Alphabot2-Ar and the Mona robot. This starts with the processing board, where both robots are based on the Arduino platform, with support for $I2C$ and $SPI$ serial communication protocol interfaces (similar to Alphabot2-Ar) [29]. This choice further reinforces the aspect that the chosen robot, the Alphabot2-Ar, is an ideal choice for educational purposes.

Other similarities between both robots are the actuators and sensors, where both of them use 2 motorized wheels as actuators, connected to $PWM$ ports that allow for the individual control of rotational speed. The sensors are also identical, with both robots using $IR$ obstacle detection sensors [29]. However, while Alphabot2-Ar has an ultrasonic obstacle detection sensor that is used to also calculate distance, the Mona robot uses the same $IR$ obstacle detection sensors to measures distance as well [29]. This is a worse choice, when compared to the use of ultrasonic sensor, due to the more difficult calculations needed and the lower reliability of the data. However, in the absence of an ultrasonic sensor, in the case of the Mona robot, this presents a valid alternative. In addition to the ultrasonic sensor, the Alphabot2-Ar also has $IR$ line detecting sensors, used to navigate by following a line over a contrasting background. The addition of these sensors gives a big advantage for the Alphabot2-Ar over the Mona, since it can navigate a given path on flat surface, without any obstacles.

Similar peripherals like $LED$s and $ADC$s are used by both robots, however the Alphabot2-Ar also has an $OLED$ screen, a joystick and a buzzer. Both robots also present support for XBee network communication module boards and both robots have the capability to measure battery voltage. The same Arduino software development platform can be used on both robots, alongside the same coding languages.

Finally, the third aspect that validates the hypothesized theoretical model is the fact that both $MRAS$ and the laboratory guides developed present similar activities to teach and evaluate similar theoretical and practical concepts. In $MRAS$, the first activities approach the ”core functionality of the robot with regards to the characterisation of the motors and sensors.” [29]. Secondly, activities that test open and closed loop controls are also developed [29]. Both of these approaches are also present in the laboratory guides developed, where it is asked to the students the development of solutions that guide the robot by using the complete sensor and actuator set of the Alphabot2-Ar and that then use these solutions to implement algorithms to control the motion of the robot in an optimized way.

## 2.3 Theoretical Model 2

The second theoretical model compiled states that, for a solid line tracking functionality, using data normalization, weighted average and then applying the $PID$ algorithm to that data ensures the most robust method applicable in a low-cost, small-form educational-oriented mobile robot with the sensor and actuator set that the Alphabot2-Ar has, alongside similar processing power.

In the presented work, the data from the infrared line detecting sensors passes by an $ADC$ before being sent to the Arduino board for processing. The $ADC$ used is a 10-bit one, so the data is received by the Arduino with a 1024 resolution. For each of the 5 $IR$ sensors, one value is received, which represents the amount of reflected infrared light that the sensor received. A stronger value means a higher presence of color white underneath the sensor, while a lower value means that, beneath the sensor, the colour black is predominant.

This data alone is enough to calculate a rough approximation of the robot’s position regarding a black line drawn over a white background. And that rough approximation can then be used, alongside a simple algorithm to guide the robot’s behavior (turn left, right or keep forward).

However, that solution does not seem appropriate or adequate for an environment where the goal of using mobile robots, like the one presented here, is to teach students regarding efficient and precise ways to guide the robot’s movement.

As such, data normalization, weighted average and the $PID$ algorithm were chosen to process the data, find the robot’s position and act upon it, as they demonstrated to be efficient, simple and yet adequate enough to be part of the $ACIC$ curriculum.

Data normalization is used to translate the [0; 1023] interval of the resolution of the $ADC$ into a scale that is equal for each of the 5 sensors. This needs to occur because, for each sensor, a calibration is done where, in the designed track, the sensor will indicate its highest valued and lowest valued reading. These values then serve as limits for the values read and, since every sensor will present their own limits, a normalized value between a preset scale for each sensor is going to allow an equal reading for each sensor when they are sensing the lowest and highest value data. The equation used for the data normalization is the following:

$$normalizedValue = \frac{(valueRead(sensor) - minimumCalibratedValue(sensor)) * scaleMaximumValue}{maximumCalibratedValue(sensor) - minimumCalibratedValue(sensor)} \tag{2.1}$$

Weighted average is used to calculate the position of the robot in regards to the line, by attributing different weights to each sensor. By doing this, it is possible to assign a numeric value to the position of the robot in regards to line. The equation used for the weighted average is, for each sensor j, with j between [0; 4], and with final value representing the normalized value for that sensor:

$$weightedAverage = \frac{\sum_{j=0}^{4}(finalValue(j) * (j * 100 - 200))}{\sum_{j=0}^{4} finalValue(j)} \tag{2.2}$$

In this formula, the (j * 100 - 200) part will translate into the [-200; 200] interval that equals the weights used in the algorithm, depending on the number of the sensor (0 to 4 = -200 to 200). The value returned by the formula is the position of the robot in regards to the line.

By then having that numeric value of the position and applying the $PID$ algorithm to determine the speed value necessary for each wheel, in order to center the robot, it is possible to obtain a much smoother and more accurate operation of the robot, when comparing to the simpler approach presented earlier. The classical $PID$ equation, that is also used for the implementation of the $PID$ algorithm in this work, is the following:

$$pidValue = proportional * KP + integral * KI + derivative * KD \tag{2.3}$$

In this equation, the proportional value equals the error, the integral value is the sum of the error over time and the derivative value is the difference between the error present in the current measure and the error present in the previous measure. KP, KI and KD are constants used as multipliers, which alter the effect of the proportional, integral and derivative values. The papers presented below support this researched thesis.

Engin and Engin [32] developed their work over a low-cost wheeled mobile robot with the same intent of having the robot following a line. Their target audience also consists of students beginning their learning in such diverse topics as "... microcontroller hardware and software, interfacing technologies, automatic control theory, and sensor technologies etc" [32]. And to teach these topics, their work also relies on laboratory sessions where students can learn embedded systems. Finally, the robot used also presents many similarities to Alphabot2-Ar, including the 2 motorized wheels, the $PCB$ acting as chassis and the use of optical sensors for line detection.

When it comes to the actual data processing, a different approach is used by Engin and Engin [32]. The difference resides with the fact that a quadratic line detection algorithm was used, as opposed to a weighted average [32]. This approach uses a quadratic line to detect the position of the line in regards to the sensors [32]. Similar to both algorithms are the facts that the sensors closer to the line report higher values, the attribution of weights to each sensor in order to have a numerical value for each sensor and a line position value that is limited by the biggest and lowest value attributed to the sensors (usually the one on the most left has the lowest weight value and the one on the most right has the highest weight value). The only big difference between both algorithms is the way the position is calculated. In the quadratic algorithm, a quadratic equation involving the weights/coordinates is used alongside the reported values for each sensor [32]. This equation is then analyzed in order to extract the peak of the quadratic curve which then indicates the coordinate for the position of the line. This particular approach required "...three consecutive sensors with higher output readings than the other five sensors..." [32].

The similarities between the weighted average procedure and this quadratic approach meant both can be considered similar ways to extract the line's position in regards to the sensors. Both approaches present similar results and there are no obvious advantages on using either approach except for the fact that the quadratic one's behavior improves as more sensors are deployed. Engin and Engin [32] used a robot with 8 sensors, in comparison to the Alphabot2-Ar's 5. This difference led to the adoption of the weighted average procedure.

In regards to the $PID$ algorithm used, both approaches implement the algorithm in a similar way. An error between the center of the sensors and the line is used to calculate the speed change the robot needs to assume in order to correct the error and make it 0. As Engin and Engin [32] explain, the working of the algorithm is as follows: "First, the controller calculates the current position, and then calculates the error based on the current position. It will then command the motors to take a hard turn, if the error is high or a small turn, if the error is low. Basically, the magnitude of the turn taken will be proportional to the error. This is a result of the proportional control. Even after this, if the error does not decrease approximately to zero, the controller will then increase the magnitude of the turn further and further over time till the robot centers over the line. This is the result of the integral control. In the process of centering over the line, the robot may overshoot the target position and move to the other side of the line where the above process is followed again. Thus the robot may keep oscillating about the line in order to center over the line. To reduce the oscillating effect over time, the derivative control is used. The proportional term is only a gain amplifier, and the derivative term is applied in order to improve the response to disturbance, and also to compensate for phase lag at the controlled object." [32].

The $PID$ algorithm implemented in the laboratory guides is executed in the exact same way, with the proportional value controlling how much the magnitude of the turn is, the integral value commanding the future turn values in order to smoothly correct the error and the derivative value used to reduce the oscillation of the robot and make it correct the position and turn the error to 0. To support this algorithm, KP, KI and KD constants must be used as multipliers, which alter the effect of the proportional, integral and derivative values. These constants must be tuned to find the best behavior for the robot. The method used by Engin and Engin [32] to tune theirs is similar to the one used during the development of this work and the one that is proposed in the laboratory guides.

Balaji et al. [33] exposes the traditional way of doing line following, before explaining their implementation. The bare algorithm, so to speak, is similar to the bare one presented earlier in this document. Balaji et al. [33] expose it as "...the outputs from the digital line sensors are given to a microcontroller which is programmed to run the motors of the robot in various speeds depending on where the robot is positioned with respect to the centre of the line [1] - [6]. The motor speed settings are selected by trial and error for various sensor readings." [33]. This approach, as stated earlier in this document and reinforced by [33], presents an incomplete and inaccurate way to process line following.

To solve this, Balaji et al. [33] chose a similar weighted average plus $PID$ algorithm approach. The robot used is, similar to this document, a 2 wheeled mobile robot with a ball castor (it has only one as opposed to the Alphabot2-Ar, which has 2). The sensors used to detect the line are the same $IR$ sensors, although the robot used by Balaji et al. [33] has 10 as opposed to the Alphabot2-Ar's 5.

In regards to the weighted average, the same position error logic is used. Each sensor is associated with a given weight value. This weight value is then multiplied by the value read by that specific sensor. This is done for every sensor and the products are added and then divided by the sum of all weights and the position is obtained (if the target/center position value is then subtracted, the error value is obtained).

Similar to the work this document is presenting, data normalization is also performed at [33], while it wasn't at [32]. The normalization process is identical to the one used in the laboratory guides, and even the rotation of the robot for calibration is done in the same way. The procedure, as stated by Balaji et al. [33] is as follows: "To normalize the sensor values, the robot is placed on the track and rotated such that all the sensors pass over the black and white surfaces. The maximum and minimum readings for all the sensors are noted down and stored in non-volatile memory (Data Flash) of the microcontroller. When the positional error value is to be calculated, the sensor readings are scaled to a normalized value between 0 and 1000 such that 0 corresponds to the minimum value (White) and 1000 corresponds to the maximum value (Black). Thus, the sensor readings are normalized such that when all the sensors are placed over any surface with uniform reflectiveness, a uniform value is obtained from all of them." [33].

The justification for the use of normalized values for the sensors is also similar to the one given here, where "appreciable difference between the readings of various sensors due to various factors like placement, manufacturing differences, etc." [33] result in different maximum and minimum readings for each sensor, but that represent the same no line/line interval.

As for the application of the $PID$ algorithm, a similar approach to the laboratory guides was used, however, the algorithm used is modified to be used specifically with mobile robots in a line following paradigm.

First, the classical $PID$ equation is used. In addition to this equation, which is the one used in the laboratory guides, Balaji et al. [33] introduces a new speed variation parameter. This is justified by "For stability considerations, the maximum speed of the robot had to be set depending on the characteristics of the track on which the robot runs. This reduces the average speed of the robot because the speed needs to be reduced so that the robot is stable when negotiating sharp curves. In order to avoid this problem and to improve performance, the speed of the robot needs to be dynamically changed depending on curvature of the track. To achieve this, an additional parameter has been introduced which is henceforth called the speed reduction parameter. The speed reduction parameter is similar to the modified integral control parameter but it reduces the speed of both motors." [33]. This modification is inserted as a parameter that is used when calculating the speed for each motor. A similar approach was not used in the laboratory guides because the goal of the laboratory sessions is not to make the robot go as fast as possible but to have a stable behavior which the students can identify. Given that, for the maximum tested speeds on the designed racetracks, the robot's behavior was as expected, there was no need for the addition of this variable, whose sole purpose is to enhance the robot's speed around the track at an acceptable curve stability. But, other than that, the $PID$ approach is identical to the one used in this document.

# Chapter 3

# Robot Analysis

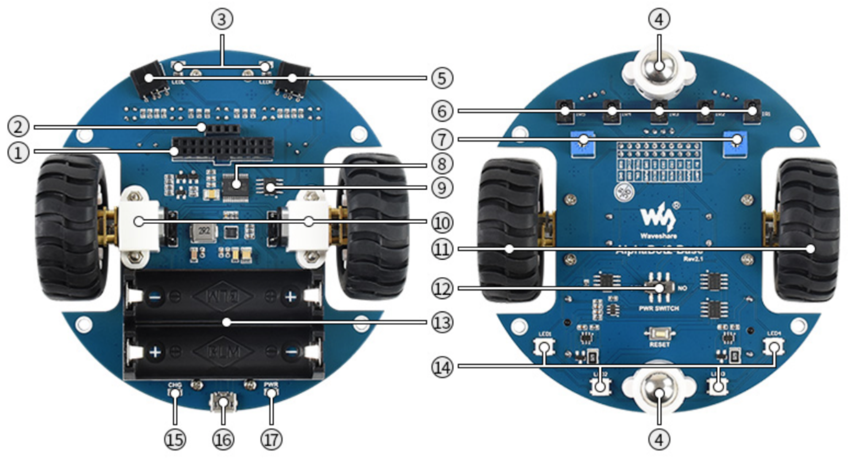
The robot being used to develop the laboratory guides and which will be used in the laboratory sessions is the Alphabot2-Ar, made by Waveshare. This robot was available in $IST$, having been chosen for a previous assignment, and an in-depth analysis was necessary to see if it was suitable for use in a classroom environment, despite having already been chosen for this work.

Below you will find the results of the analysis, which dictated the suitability of the robot to be used in the development of the laboratory guides. This analysis also showed the robot to be indicated for an educational use and that the theoretical/practical topics that make up the $ACIC$ curriculum can be fully deployed using this robot. Alongside the hardware analysis, mentions are made to the example software programs provided by Waveshare and that were used to test the components and understand better how to interact with them using Arduino code language [3, 34].

This in-depth analysis was also made because it puts, in one space, a complete guide to the functioning of each of the components that make up the Alphabot2-Ar. This means that, if any doubt arises regarding the functioning of each of the components of the robot, this work constitutes the first place where a possible answer for the doubt can be found. Of course, this work is only so big and cannot mention every aspect of every component, but it does mention the important aspects for the correct functioning of the mobile robot. This analysis also allows the Professors that use the laboratory guides and the Alphabot2-Ar in their classes, alongside the code solutions for each of the laboratory guides, to base their work preparations and course curriculum and theoretical contents on the information provided in the robot analysis, as well as in the laboratory guides.

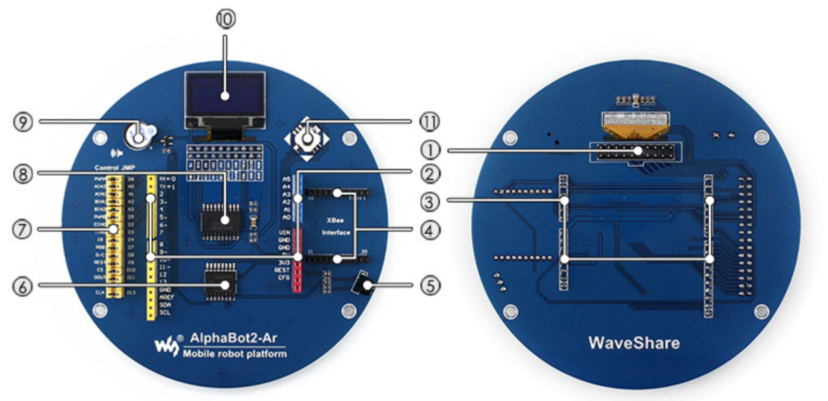
## 3.1 Robot Structure

The Alphabot2-Ar has a two-level circular $PCB$ chassis with 11 cm in diameter and 6 cm in height. The base level of the chassis is called Alphabot2-Base and is common between the Alphabot2-Ar and Alphabot2-Pi (Raspberry-Pi version) robots. The top chassis board is called the Alphabot2-Ar and the Arduino board attaches to this $PCB$. In Figures 3.1 and 3.2 you may find a diagram of both $PCB$ boards and the components present on each board. The robot comes with some pre-assembled parts



1. **AlphaBot2 control interface:** for connecting sorts of controller adapter board
2. **Ultrasonic module interface**
3. **Obstacle avoiding indicators**
4. **Omni-direction wheel**
5. **ST188:** reflective infrared photoelectric sensor, for obstacle avoiding
6. **ITR20001/T:** reflective infrared photoelectric sensor, for line tracking
7. **Potentiometer** for adjusting obstacle avoiding range
8. **TB6612FNG** dual H-bridge motor driver
9. **LM393** voltage comparator
10. **N20 micro gear motor** reduction rate 1:30, 6V/600RPM
11. **Rubber wheels** diameter 42mm, width 19mm
12. **Power switch**
13. **Battery holder:** supports 14500 batteries
14. **WS2812B:** true color RGB LEDs
15. **Battery charging indicator**
16. **5V USB battery charging port**
17. **Power indicator**

Figure 3.1: Components present in the AlphaBot2-Base. [35]

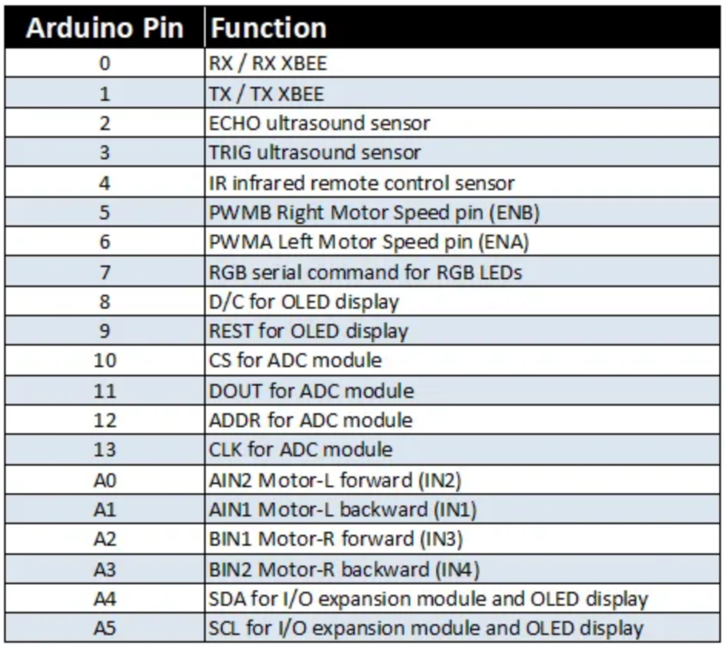


1. **AlphaBot2 control interface:** for connecting AlphaBot2-Base
2. **Arduino expansion header:** for connecting Arduino shields
3. **Arduino interface:** for connecting Arduino compatible controller
4. **Xbee connector:** for connecting dual-mode Bluetooth module, remotely control the robot via Bluetooth
5. **IR receiver**
6. **PC8574:** I/O expander, SPI interface
7. **Arduino peripheral jumpers**
8. **TLC1543:** 10-bit AD acquisition chip
9. **Buzzer**
10. **0.96inch OLED** SSD1306 driver, 128x64 resolution
11. **Joystick**

Figure 3.2: Components present in the AlphaBot2-Ar. [35]

but some assembly is still required. Laboratory guide 0 [17] provides detailed instructions on how to assemble the remaining parts.

In [36], it is possible to find the schematics regarding the assembly of all the components of the Alphabot2-Base and, at [37], it is present the schematics of the assembly of all the components that make up the Alphabot2-Ar. In Table 3.1, there is a list of all the Arduino pins and the components connected to each pin.



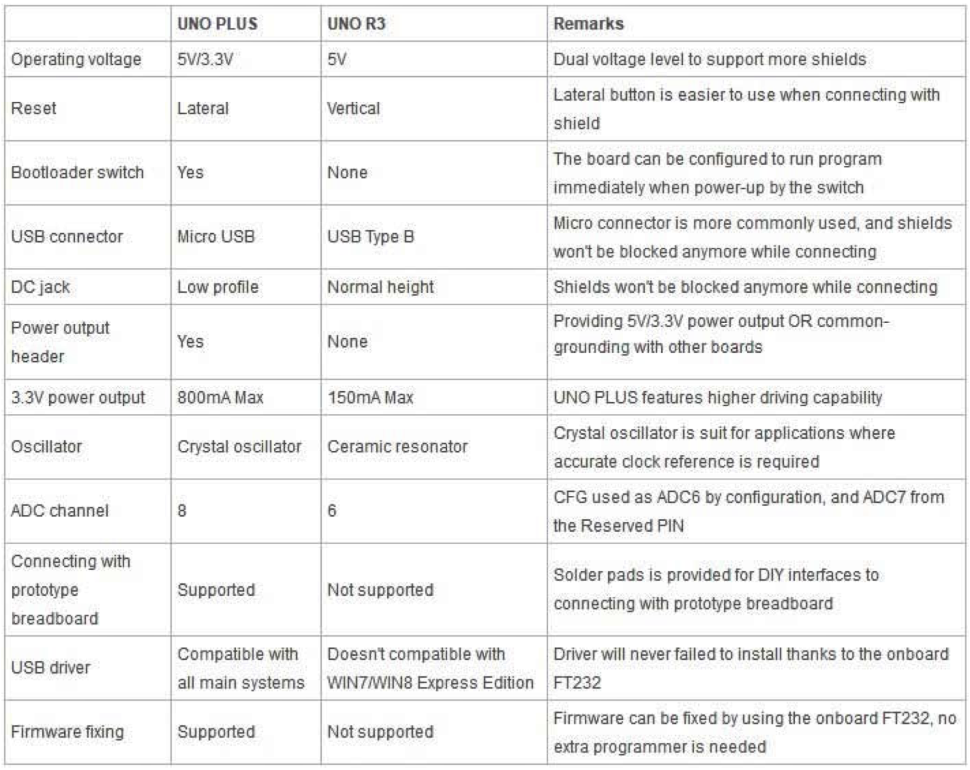
| Arduino Pin | Function |
|---|---|
| 0 | RX / RX XBEE |
| 1 | TX / TX XBEE |
| 2 | ECHO ultrasound sensor |
| 3 | TRIG ultrasound sensor |
| 4 | IR infrared remote control sensor |
| 5 | PWMB Right Motor Speed pin (ENB) |
| 6 | PWMA Left Motor Speed pin (ENA) |
| 7 | RGB serial command for RGB LEDs |
| 8 | D/C for OLED display |
| 9 | REST for OLED display |
| 10 | CS for ADC module |
| 11 | DOUT for ADC module |
| 12 | ADDR for ADC module |
| 13 | CLK for ADC module |
| A0 | AIN2 Motor-L forward (IN2) |
| A1 | AIN1 Motor-L backward (IN1) |
| A2 | BIN1 Motor-R forward (IN3) |
| A3 | BIN2 Motor-R backward (IN4) |
| A4 | SDA for I/O expansion module and OLED display |
| A5 | SCL for I/O expansion module and OLED display |

Table 3.1: Arduino pins and the components attached to each pin. [38]

## 3.2 Arduino UNO Plus

The Alphabot2-Ar uses an Arduino UNO Plus as the processing board. This board integrates all the $I/O$ and processes the data sent by the sensors and sends commands to the actuators [39]. It also controls the peripherals and processes their signals. The Arduino UNO Plus is an ”improved & enhanced alternative solution for Arduino UNO R3” [40]. Comparing to the Arduino UNO R3, it offers several improvements, all listed in Table 3.2.

The processor used by the Arduino UNO Plus is the ATMEGA328P-AU. It has 32k bytes of Flash memory, 1k byte $EEPROM$ and 2k bytes $SRAM$. This provides enough memory and storage for executing the code being developed in the laboratory guides, with an enough storage margin for different implementations of the solutions. This board also has a built-in $ADC$ with 8 channels as well as support for serial communication protocols such as $I2C$ and $SPI$, used for communication with external modules. This board unifies the communication with all the $I/O$, including the sensors and actuators and the $USB$ communication with the computer. In order to develop code to operate the Arduino board and, consequently, the Alphabot2-Ar, the Arduino $IDE$ was used, alongside the Arduino Language Reference. Other $IDE$s that support Arduino boards can be used, as well as other languages supported by



| | UNO PLUS | UNO R3 | Remarks |
|---|---|---|---|
| Operating voltage | 5V/3.3V | 5V | Dual voltage level to support more shields |
| Reset | Lateral | Vertical | Lateral button is easier to use when connecting with shield |
| Bootloader switch | Yes | None | The board can be configured to run program immediately when power-up by the switch |
| USB connector | Micro USB | USB Type B | Micro connector is more commonly used, and shields won't be blocked anymore while connecting |
| DC jack | Low profile | Normal height | Shields won't be blocked anymore while connecting |
| Power output header | Yes | None | Providing 5V/3.3V power output OR common-grounding with other boards |
| 3.3V power output | 800mA Max | 150mA Max | UNO PLUS features higher driving capability |
| Oscillator | Crystal oscillator | Ceramic resonator | Crystal oscillator is suit for applications where accurate clock reference is required |
| ADC channel | 8 | 6 | CFG used as ADC6 by configuration, and ADC7 from the Reserved PIN |
| Connecting with prototype breadboard | Supported | Not supported | Solder pads is provided for DIY interfaces to connecting with prototype breadboard |
| USB driver | Compatible with all main systems | Doesn't compatible with WIN7/WIN8 Express Edition | Driver will never failed to install thanks to the onboard FT232 |
| Firmware fixing | Supported | Not supported | Firmware can be fixed by using the onboard FT232, no extra programmer is needed |

Table 3.2: Comparison between the Arduino UNO Plus and R3. [40]

Arduino, such as C and C++.

## 3.3 TLC1543 Analog-Digital Converter

Present in the Alphabot2-Ar $PCB$ is the TLC1543 Analog-Digital Converter. The function of this $ADC$ is to convert analog signals sent by the $IR$ line detecting sensors and by the battery (which is also connected to the $ADC$) into digital ones. The TLC1543 is a 10-bit resolution $ADC$ with 11 analog input channels [41]. According to its specification manual, the TLC1543 has ”three inputs and a 3-state output [chip select (CS), input-output clock (I/O CLOCK), address input (ADDRESS), and data output (DATA OUT)] that provide a direct 4-wire interface to the serial port of a host processor. (...) In addition to a high-speed A/D converter and versatile control capability, these devices have an on-chip 14-channel multiplexer that can select any one of 11 analog inputs ...” [41]. A functional block diagram of the $ADC$, alongside a table of the terminal functions can be found in Figure 3.3 and Table 3.3

The way the $ADC$ works for the $IR$ line detecting sensors is as follows: A constant reference voltage of 3.3V flows through the $ADC$. Each of the infrared sensors sends a signal varying from 0V (when it detects the line with the maximum intensity) to 3.3V (when no line is detected). The $ADC$ then compares the voltage each sensor sends with the reference voltage and converts the final result to a digital one. Since the TLC1543 is a 10-bit $ADC$, the final converted values range from 0 (when it detects the line with the maximum intensity) to 1023 (when no line is detected). This value is then received by the Arduino board.

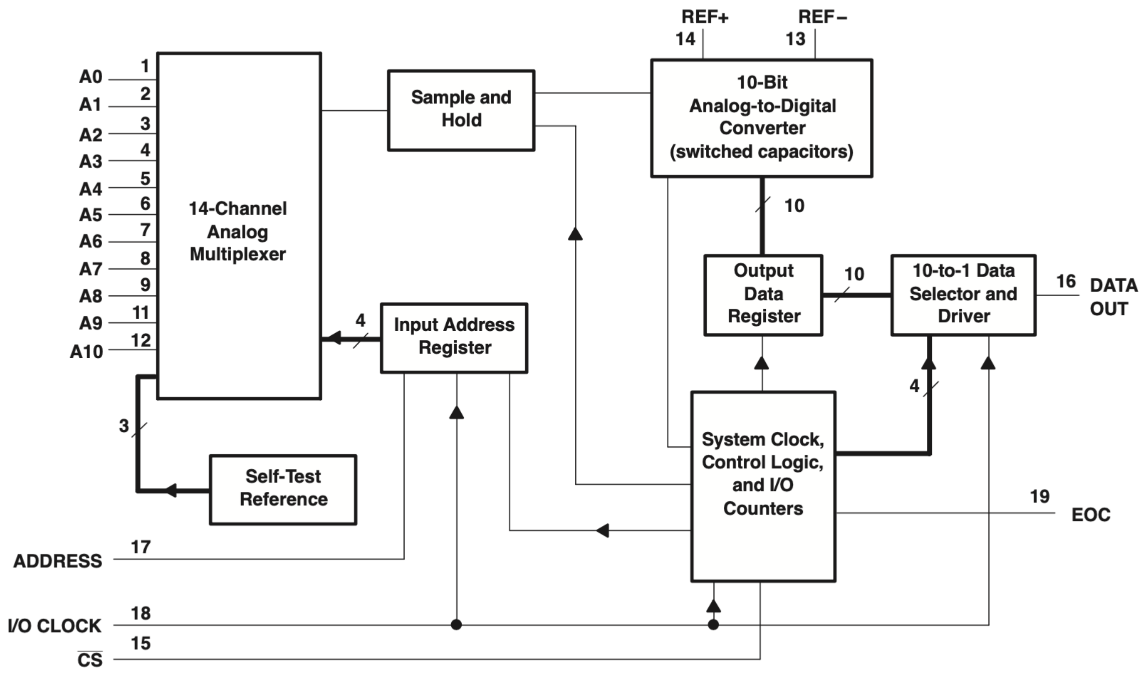


Figure 3.3: Functional Block Diagram. [41]

| TERMINAL | | I/O | DESCRIPTION |
|---|---|---|---|
| NAME | NO. | | |
| ADDRESS | 17 | I | Serial address input. A 4-bit serial address selects the desired analog input or test voltage that is to be converted next. The address data is presented with the MSB first and shifts in on the first four rising edges of I/O CLOCK. After the four address bits have been read into the address register, this input is ignored for the remainder of the current conversion period. |
| A0-A10 | 1-9, 11, 12 | I | Analog signal inputs. The 11 analog inputs are applied to these terminals and are internally multiplexed. The driving source impedance should be less than or equal to 1 kΩ. |
| $\overline{\text{CS}}$ | 15 | I | Chip select. A high-to-low transition on this input resets the internal counters and controls and enables DATA OUT, ADDRESS, and I/O CLOCK within a maximum of a setup time plus two falling edges of the internal system clock. A low-to-high transition disables ADDRESS and I/O CLOCK within a setup time plus two falling edges of the internal system clock. |
| DATA OUT | 16 | O | The 3-state serial output for the A/D conversion result. This output is in the high-impedance state when $\overline{\text{CS}}$ is high and active when $\overline{\text{CS}}$ is low. With a valid chip select, DATA OUT is removed from the high-impedance state and is driven to the logic level corresponding to the MSB value of the previous conversion result. The next falling edge of I/O CLOCK drives this output to the logic level corresponding to the next most significant bit, and the remaining bits shift out in order with the LSB appearing on the ninth falling edge of I/O CLOCK. On the tenth falling edge of I/O CLOCK, DATA OUT is driven to a low logic level so that serial interface data transfers of more than ten clocks produce zeroes as the unused LSBs. |
| EOC | 19 | O | End of conversion. This output goes from a high to a low logic level on the trailing edge of the tenth I/O CLOCK and remains low until the conversion is complete and data are ready for transfer. |
| GND | 10 | I | The ground return terminal for the internal circuitry. Unless otherwise noted, all voltage measurements are with respect to this terminal. |
| I/O CLOCK | 18 | I | Input/output clock. This terminal receives the serial I/O CLOCK input and performs the following four functions: 1) It clocks the four input address bits into the address register on the first four rising edges of the I/O CLOCK with the multiplex address available after the fourth rising edge. 2) On the fourth falling edge of I/O CLOCK, the analog input voltage on the selected multiplex input begins charging the capacitor array and continues to do so until the tenth falling edge of I/O CLOCK. 3) It shifts the nine remaining bits of the previous conversion data out on DATA OUT. 4) It transfers control of the conversion to the internal state controller on the falling edge of the tenth clock. |
| REF+ | 14 | I | The upper reference voltage value (nominally $V_{CC}$) is applied to this terminal. The maximum input voltage range is determined by the difference between the voltage applied to this terminal and the voltage applied to the REF- terminal. |
| REF- | 13 | I | The lower reference voltage value (nominally ground) is applied to this terminal. |
| $V_{CC}$ | 20 | I | Positive supply voltage |

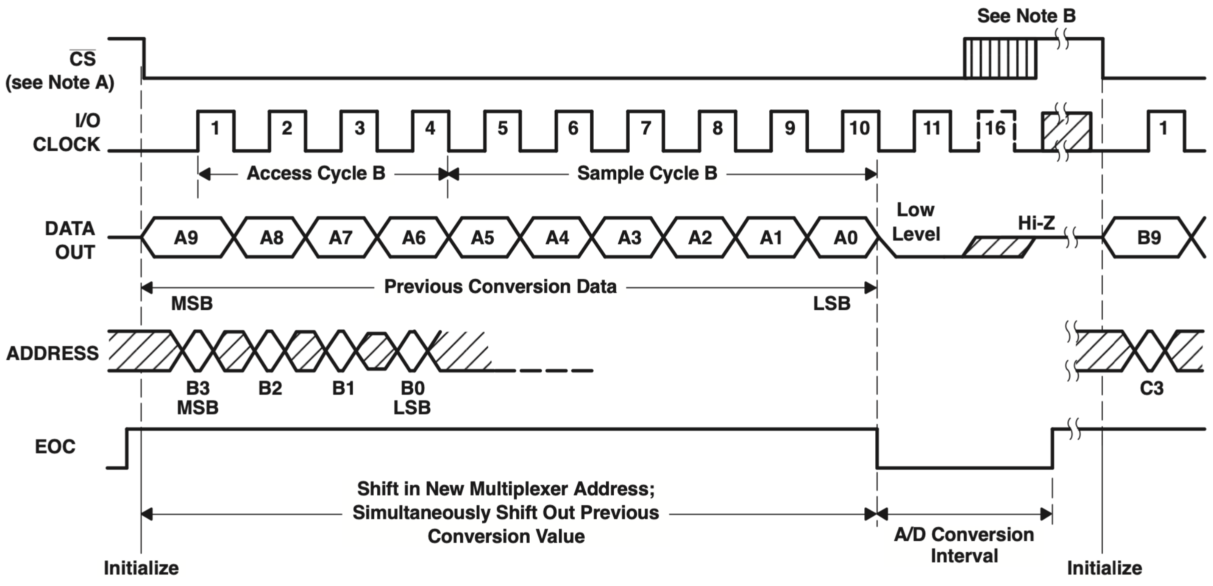
Table 3.3: TLC1543 Terminal Functions. [41]

In order to access the $ADC$ with the Arduino board, there are 4 pins connecting the $ADC$ to the Arduino: CHIP SELECT (CS – Output pin), DATA OUT (Input pin), ADDRESS (Output pin) and CLOCK (Output pin). A High to Low transition on the CS pin resets the internal counters and controls and enables DATA OUT, ADDRESS, and CLOCK [41]. DATA OUT pin outputs the 10-bit result of the previous conversion from the $ADC$ to the Arduino board [41]. ADDRESS pin is the pin from the $ADC$ that receives the 4-bit address sent by the Arduino board. Both the ADDRESS and DATA OUT pins send and receive information bit by bit, so any information must be converted to and from binary before being sent and after being received. For the ADDRESS pin, the addresses for the components connected to the $ADC$ must be converted to a 4-bit binary number and for the DATA OUT pin, the 10 bits received must then be composed into an integer, ranging from 0 to 1023. Both ADDRESS and DATA OUT operate with $MSB$ first, meaning the $ADC$ receives and sends data through these pins starting always with the $MSB$s [41]. A High to Low transition on the CLOCK pin advances one operation cycle in the $ADC$. For each bit received, processed or sent by the $ADC$, one operation cycle must be started and concluded (High to Low transition in the CLOCK pin).

In the operating mode present in the laboratory guides (fast mode - mode 3 [41]), the $ADC$ operates with each conversion cycle (receive address and output previous conversion plus Analog-Digital conversion operation cycles) taking 16 operation cycles [41]. ”There are six basic serial-interface timing modes that can be used with the device. These modes are determined by the speed of I/O CLOCK and the operation of CS (...). These modes are (1) a fast mode with a 10-clock transfer and CS inactive (high) between conversion cycles, (2) a fast mode with a 10-clock transfer and CS active (low) continuously, (3) a fast mode with an 11- to 16-clock transfer and CS inactive (high) between conversion cycles, (4) a fast mode with a 16-clock transfer and CS active (low) continuously, (5) a slow mode with an 11- to 16-clock transfer and CS inactive (high) between conversion cycles, and (6) a slow mode with a 16-clock transfer and CS active (low) continuously.” [41].

These timing modes exist so that the user can select the type of the desired conversion. The first main division is between a fast conversion and a slow one. A fast conversion converts and presents the results of the conversion quicker than the slow mode conversion, but sacrifices on conversion resolution and accuracy. By choosing a slower conversion, the conversion takes a bit longer but presents a more accurate result with a more precise resolution. ”The device is in a fast mode when the serial I/O CLOCK data transfer is completed before the conversion is completed. With a 10-clock serial transfer, the device can only run in a fast mode since a conversion does not begin until the falling edge of the tenth I/O CLOCK.” [41].

Within the fast conversion modes, there are some differences between each mode, with the most obvious being the number of $I/O$ clocks necessary to finish the conversion. One other separating factor for these fast conversion modes is the $MSB$ presented at the end of the conversion, that also initiates the serial communication. One of the modes has a CS falling edge and the others have the $EOC$ falling edge. For the use case of the $IR$ line detecting sensors and also the battery voltage calculation, the chosen mode to be used is the fast mode - mode 3 [41], where it has between 11 to 16 clock transfer cycles. This mode was chosen due to its balance between a fast operation, since it is a fast mode, and a reliable and accurate conversion process and result, due to its 11 to 16 number of I/O clocks, as compared to 10 in other fast-mode modes. In Figure 3.4 you may find depicted a conversion cycle for the fast-mode mode 3.



A. To minimize errors caused by noise at $\overline{CS}$, the internal circuitry waits for a setup time plus two falling edges of the internal system clock after $\overline{CS}\downarrow$ before responding to control input signals. Therefore, no attempt should be made to clock in an address until the minimum $\overline{CS}$ setup time has elapsed.

B. A low-to-high transition of $\overline{CS}$ disables ADDRESS and the I/O CLOCK within a maximum of a setup time plus two falling edges of the internal system clock.

Figure 3.4: TLC1543 Timing for 11- to 16-Clock Transfer Using CS (Serial Transfer Interval Shorter Than Conversion). [41]

## 3.4 ICR 14500 Batteries

The Alphabot2-Ar is powered by 2x ICR 14500 3.7V 800mAh rechargeable batteries connected in series to the Alphabot2-Base $PCB$. This effectively doubles the voltage provided to the system, for a total of 7.4V. However, these batteries actually charge to 4.2V when full, so the total voltage when full is 8.4V (For all the calculations we will consider the theoretical voltage capacity of the battery of 7.4V instead of the actual maximum of 8.4V, due to discrepancies between this maximum value) [42].

As the battery is drained, the voltage of the batteries decreases. It decreases till what's called the Safe Discharge Voltage – minimum battery voltage to ensure the battery works properly [42]. That voltage, for this type of battery, usually ranges between 2.75V to 3V. Since the provided batteries may present some differences, we will consider the Safe Discharge Voltage as 3V [42]. This means the combined provided voltage of the batteries must not decrease below 6V. The battery voltage is provided, simultaneously, to the electric motors, Arduino board and TLC1543 $ADC$ (all of them receive the same voltage).

The fact that the electric motors are powered by the batteries provides an increase in power over the standard 5V provided by the Arduino board to the motors.

The Arduino board is also powered by the batteries. The 2 batteries (from this point forwards only mentioned as battery) are connected to the VIN pin of the Arduino. This port is designed to receive between 7V and 12V and then regulates this voltage to output a consistent and constant 5V (voltage necessary for the correct functioning of the Arduino board), through a technique called voltage divider.

Voltage divider technique takes a higher voltage and reduces it to a lower voltage by using a pair of resistors. Depending on the values of the resistors, the conversion can produce a higher or lower new voltage value. This same technique is used to connect the battery to the $ADC$, which allows the reading of the level of the battery voltage.

Inserted between the $ADC$ and the battery is a pair of 10k Ohms resistors, as visible in [37]. According to the formula for voltage dividers:

$$V_{out} = \frac{V_s * R_2}{R_1 + R_2} \tag{3.1}$$

where Vout corresponds to the voltage leaving the voltage divider circuit, Vs corresponds to the voltage going into the circuit and R1 and R2 stand for the 2 resistors placed in the circuit, a Vs of 7.4V and an R1 and R2 of 10k Ohms corresponds to a Vout of 3.7V. This means that going into the $ADC$ is a voltage of 3.7V, when the voltage provided is 7.4V. The $ADC$ presents the battery voltage in the 10-bit scale of [0; 1023]. However, the 1023 value does not equal the 8.4V theoretical maximum voltage nor the 7.4V either. Since the reference voltage flowing through the $ADC$, and that is used to compare with the signals sent by the infrared line detecting sensors, is actually 3.3V, the maximum that the $ADC$ can convert from analog to digital is 3.3V. So, the 1023 value actually equals 3.3V. So that allows you to know that the battery has at least 3.3V, which corresponds to 6.6V, if the voltage dividing is undone.

So this $ADC$ allows to read the voltage level of the battery up to 6.6V, which is enough since it is a higher value than the safe discharge voltage of 6V (voltage at which the batteries should be charged). Since the voltage dividing operation cuts the voltage flowing from the battery to the $ADC$ in half, the voltage provided to the $ADC$ is, when battery is full, 4.2V. When nominal, 7.4V, voltage is provided to the system, the $ADC$ only receives 3.7V. To convert the scale from [0; 1023] to [0V; 6.6V], the formula is:

$$V_{battery} = \frac{ADC_{value} * ADC_{referenceV}}{1023} * 0.5 \tag{3.2}$$

where the $ADC$ referenceV equals 3.3, 1023 the resolution of the adcValue and 0.5 equals the voltage divider (10000/(10000 + 10000)).

## 3.5 PCF8574 I/O Expander

Installed into the Alphabot2-Ar $PCB$ is the PCF8574 8-Bit $I/O$ Expander. This $I/O$ expander is used to increase the number of $I/O$ devices connected to the Arduino board. Connected to this $I/O$ expander are the 2 infrared obstacle detecting sensors, the joystick and the buzzer.

It communicates with the Arduino using $I2C$ communication protocol. The $I2C$ communication protocol works in a Master/Slave configuration, where one or more Slaves can be connected to one Master on the same bus. This protocol uses 2 pins to send and receive data: $SCL$ and $SDA$. In this particular case the Arduino board is the master and the devices connected to the $I/O$ expander are the slaves. To communicate with any of the $I/O$ devices, the Arduino must send "a start condition, a high-to-low transition on the SDA I/O while the SCL input is high. After the start condition, the device address byte is sent, MSB first, including the data direction bit (R/W). This device does not respond to the general call address. After receiving the valid address byte, this device responds with an acknowledge, a low on the SDA I/O during the high of the acknowledge-related clock pulse. The address inputs (A0–A2) of the slave device must not be changed between the start and the stop conditions. The data byte follows the address acknowledge. If the R/W bit is high, the data from this device are the values read from the P port. If the R/W bit is low, the data are from the master, to be output to the P port. The data byte is followed by an acknowledge sent from this device." [43].

This means that $I2C$ communication with this device may be used to send data or receive data. The only component that receives data is the buzzer, while the 2 infrared sensors and the joystick only send data back to the Arduino.

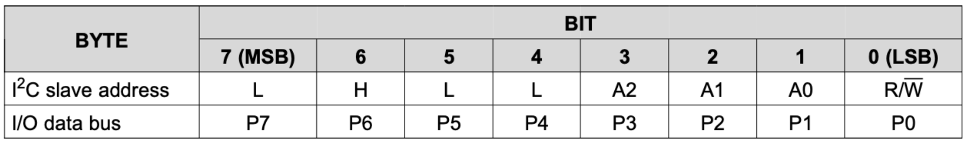
To control this communication, the Wire.h library is used [44]. This library automates many of the procedures necessary to communicate with the device into "simple to call" functions. In order to access the devices, specific addresses must be used. First, the $I2C$ address for the expander itself must be calculated. To do that, first it becomes necessary to analyze the PCF8574 address map presented in Table 3.4.

| Pin connectivity | | | Address of PCF8574 | | | | | | | | Address byte value | | 7-bit hexadecimal address without $R/\overline{W}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A2 | A1 | A0 | A6 | A5 | A4 | A3 | A2 | A1 | A0 | $R/\overline{W}$ | Write | Read | |
| $V_{SS}$ | $V_{SS}$ | $V_{SS}$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | - | 40h | 41h | 20h |
| $V_{SS}$ | $V_{SS}$ | $V_{DD}$ | 0 | 1 | 0 | 0 | 0 | 0 | 1 | - | 42h | 43h | 21h |
| $V_{SS}$ | $V_{DD}$ | $V_{SS}$ | 0 | 1 | 0 | 0 | 0 | 1 | 0 | - | 44h | 45h | 22h |
| $V_{SS}$ | $V_{DD}$ | $V_{DD}$ | 0 | 1 | 0 | 0 | 0 | 1 | 1 | - | 46h | 47h | 23h |
| $V_{DD}$ | $V_{SS}$ | $V_{SS}$ | 0 | 1 | 0 | 0 | 1 | 0 | 0 | - | 48h | 49h | 24h |
| $V_{DD}$ | $V_{SS}$ | $V_{DD}$ | 0 | 1 | 0 | 0 | 1 | 0 | 1 | - | 4Ah | 4Bh | 25h |
| $V_{DD}$ | $V_{DD}$ | $V_{SS}$ | 0 | 1 | 0 | 0 | 1 | 1 | 0 | - | 4Ch | 4Dh | 26h |
| $V_{DD}$ | $V_{DD}$ | $V_{DD}$ | 0 | 1 | 0 | 0 | 1 | 1 | 1 | - | 4Eh | 4Fh | 27h |

Table 3.4: PCF8574 Address Map. [45]

For the present installation, all A2, A1 and A0 pins must be set to Vss (or LOW) and the $R/W$ bit is not used (it is automatically decided by the Wire.h library [44]). This means that the line of the table being used is the first one, which equals an address for the $I/O$ expander of 0100000, which is 0x20 in hexadecimal (the same address indicated in [37]). The different A2, A1 and A0 values allow for the installation of multiple PCF8574 $I/O$ expanders in the same Arduino boards, with each of them having a different configuration of the A2, A1 and A0 pins, for a theoretical maximum of 8 $I/O$ expanders connected simultaneously. With each of the expanders allowing a total of 8 input devices, a maximum of 64 devices connected simultaneously to one single Arduino board is theoretically possible.

After having the $I2C$ address of the expander, it now becomes necessary to know the address of each of the devices connected to the $I/O$ expander. Bear in mind that the joystick, despite being one single component, has one address for each of the possible positions which it can be clicked (up, down, left, right and center). These 5 positions plus the 2 infrared sensors plus the buzzer total 8 pins, which means this $I/O$ expander is completely full, meaning no more components can be connected to it. To find out the addresses for each of the pins of the expander, it becomes necessary to look to [37] once again in order to understand the which device is connected to which pin and to Table 3.5, which shows how the addresses are assembled.



| BYTE | BIT | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 (MSB) | 6 | 5 | 4 | 3 | 2 | 1 | 0 (LSB) |
| I$^2$C slave address | L | H | L | L | A2 | A1 | A0 | R/$\overline{W}$ |
| I/O data bus | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

Table 3.5: PCF8574 Interface Definition. [43]

Now it becomes possible to assemble the addresses for all components. Please note that the addresses represent the component being active, meaning that if, upon requesting a value from the $I/O$ expander, it returns the address of one of the components, that means that component is currently active. The same goes for sending data to the $I/O$ expander. If the Arduino sends the address of the buzzer, the buzzer will become active. This also allows a simulated activation of each of the position of the joystick as well as the 2 infrared sensors (by sending the address of that component to the $I/O$ expander).

To assemble the address of a device, one needs simply to put the bit corresponding to that device equaling 0 and the remaining bits equaling 1. And why not the other way round? ”When using the pins as inputs, the pins are set to HIGH by the MCU, which turns on a weak 100 uA internal pull-up to Vcc. They will read as HIGH if there is no input or if the pin is being driven HIGH by an external signal but can be driven LOW by an external signal that can easily override the weak pull-up.” [46].

This means that the pins, whenever a reading from the devices starts, are all set to HIGH (= 1). Any device that is active will short a that particular’s device circuit, turning its address bit into a LOW (= 0). So, to define the address that states that a given component is active, all that’s necessary is to set the bit corresponding to the pin of the device (P0 - P7) to 0 and the remainder to 1. This applies both for reading and writing, meaning that any device that is on will return 0 in its address bit and to turn any device on just send an address with the address bit for that device to 0 (for example, to turn on the buzzer).

To activate 2 components simultaneously, it is simply necessary to turn both address bits to 0, and so on for more components to turn on. This happens when both $IR$ obstacle detecting sensors detect an obstacle at the same time (the address returned by the $I/O$ expander has the address bits for both $IR$ sensors set to 0).

## 3.6 Infrared Receiver

Attached to the Alphabot2-Ar $PCB$ is an infrared receiver. This $IR$ receiver is used to receive commands sent by the $IR$ remote control. These commands can then be used to control the behavior of the mobile robot and change all sorts of parameters in the execution of the Arduino code. Each button of the remote control is associated with a given address code. This address code is unique to each key and it is utilized to see which key was pressed.

The infrared receiver could be controlled without a specific library. But the strict timings to communicate and access the receiver make it difficult to use without a properly configured external library. As such, to receive information from the $IR$ receiver, the library IRremote.h should be used [47]. To identify the address for each of the buttons, it is only necessary to call the IrReceiver.decode() function, to see if any $IR$ command has been received by the $IR$ sensor and then save the IrReceiver.decodedIRData.command value [47]. To unlock the receiver and allow it to receive new commands, execute IrReceiver.resume() [47]. After decoding every button from the remote control, one can use the address codes to then understand which remote control key was pressed and define a specific action for that button press.

To test the component and to learn how to interact with it in an Arduino environment, the example program IR.ino was used [3, 34].

## 3.7 XBee Board

An XBee board is included with the Alphabot2-Ar robot. It can be installed in a specific socket present in the Alphabot2-Ar $PCB$. This XBee board is used to enable Bluetooth communication with external devices.

The functioning of this board is pretty similar to that of the $IR$ receiver. To connect to an external Bluetooth device, the external Bluetooth device needs simply to pair to the Arduino board, using Bluetooth $LE$, and the Arduino board is able to send and receive data from and to the external device. Then it only becomes necessary to specify the address of a given command and, in the Arduino code, specify the behavior for the robot when that given address code is received.

To communicate with the XBee adapter, the serial communication protocol is used. The decoding of the data is done automatically by the Arduino. Given the fact that not all devices/computers present in $IST$ laboratories allow Bluetooth communication, this module was not utilized in the development of this work. It is useful, however, to develop the work suggested in the Future Work section of this paper, where communication between the Alphabot2-Ar and an external device, specifically a computer, must occur wirelessly and in real time.

To test the component and to learn how to interact with it in an Arduino environment, the example program Bluetooth.ino was used [3, 34].

## 3.8 Buzzer

Connected to the PCF8574 $I/O$ expander is a buzzer. This buzzer bips whenever active and stops bipping once deactivated. In order to access this device, the Wire.h library mentioned earlier must be used [44]. Since this device is not a sensor but an actuator, it does not send data to the Arduino board but rather receives data in the form of commands. Its 2 possible commands are ON and OFF. ON equals the buzzer bipping and OFF equals not bipping.

In order to set the buzzer working, first the Wire.h library must be configured [44]. To do so, initiate the program with Wire.begin() [44]. Then, since a transmission is going to take place from the Arduino to the $I/O$ expander, the Wire.beginTransmission(device) function must be called, where device is the address for the $I/O$ expander (0x20 as seen earlier) [44]. Then, to send the command, the address of the buzzer turn on command must be defined and the same for the buzzer turn off command.

The buzzer turn on command consists of all 8 bits (corresponding to all 8 $I/O$ pins of the expander) turned to 1 except bit 5 (corresponds to pin P5) which must be turned to 0, meaning the device connected to pin P5 is active [37].

To turn off the buzzer, set all pins to 1, as to deactivate all $I/O$ connected to the $I/O$ expander. This means that the address for buzzer on is 11011111 (= 0xDF) and the address for buzzer turned off is 11111111 (= 0xFF).

Then after having defined the addresses, it is only necessary to send the address to the $I/O$ expander, via the Wire.write(message), with message being the address [44].

After finishing the transmission, call Wire.endTransmission() to terminate it [44].

## 3.9 Joystick

Also connected to the PCF8574 $I/O$ expander is the joystick. The joystick is used to interact with the robot and corresponds to a sensor, in the sense that it transmits data to the Arduino rather than receiving it.

Using the joystick requires the Wire.h library as well, but this time it is used for reading rather than writing. Once again, initialization via the Wire.begin() function must be done, but then to receive data from the expander, it is only necessary to call the Wire.requestFrom(device, bytes) function [44]. In this call, the device is the address of the expander (0x20) and bytes is equal to 1 (= 8 bits), since the expander is an 8-bit one. After making the call, using the Wire.available() function allows the verification if any data was received and, if so, using the Wire.read() functions returns the data sent back by the expander [44].

This data corresponds to which devices were active at the time of the call of the requestFrom function. This means the returned data will also have a similar address structure to the one used with the send mechanism of the Wire library [44]. For the 8 bits of the response, each bit corresponds to one $I/O$ pin (P0 - P7). Every bit that is set to LOW (= 0) corresponds to an active device at the time of the call and every bit set to HIGH (= 1) corresponds to a device that was not active at the time of the call. As such, to set the addresses for each of the positions of the joystick (to later use to identify which button was pressed and perform a given action), it is only necessary to look at [37] and see which pin, from P0 to P7, each joystick button occupies.

Since the schematic refers to Key A, B, C and D for the directional buttons and CTR for the center one, it is not immediate to know which letter corresponds to which directional button. To identify them correctly, it is only necessary to read from the expander using the Wire functions presented above and print the result. Then, press a given directional button and read the returned code and the letter is then identified. For joystick up, the address code is 11111110 (= 0xFE), for joystick down is 11110111 (= 0xF7), for joystick left is 11111011 (= 0xFB), for joystick right is 11111101 (= 0xFD) and finally joystick center is 11101111 (= 0xEF).

The request to the expander for readings must be done constantly in order to see if any buttons were pressed in real time.

To test the component and to learn how to interact with it in an Arduino environment, the example program Joystick.ino was used [3, 34].

## 3.10 OLED Screen

In the front of the Alphabot2-Ar $PCB$ is a small, 0.96 inch $OLED$ two-color display, with a resolution of 128x64. This $OLED$ screen can be used to display all kinds of information and can be controlled in real time.

To manage the contents of the screen, a specific external library should be used, that manages the communication with the display drivers. Several such libraries are available to use but most of them are not optimized for use in a mobile robot. This means that these libraries occupy too much memory and their operation is slow and heavy, resulting in a sub-optimal performance for the rest of the components that constitute the Alphabot2-Ar mobile robot, due to high processing and memory demands from the display managing library. This happens because these libraries focus on offering higher quality renderings and support the printing of non-text content. This poses a good alternative for projects that aim to take advantage of using only the display, but for the work being developed in this paper, it was not going to work. So, it became essential to find an external library that was compatible with the installed display, lightweight, fast and focused on printing text over other content. The choice made was the SSD1306Ascii library [48].

This library presents a great performance for use with the installed display and allows the functioning of the other components, such as the sensors and motors, without interruptions due to low memory available or high processing power being consumed. Since the $OLED$ display is connected to the Arduino via $I2C$ bus and utilizes the $I2C$ communication protocol, the SSD1306Ascii library provides a specific implementation of the library for displays connected via $I2C$, the SSD1306AsciiAvrI2c.h library [48]. This implementation is optimized for $I2C$ connected displays and provides an even better performance.

This library is rather simple to utilize and only requires one variable to manage the entire display.

First the address code for the display must be defined. In [37], the address is defined as 0x3C and 0x3D. The 0x3D should be used to define the screen as a 128x32 resolution display. However, since the installed display is a 128x64 resolution display, the 0x3C address code should be used. After having defined the screen address, all that's left to do is to create a variable of type SSD1306AsciiAvrI2c [48]. Then, to initialize the variable, in the setup() function of the Arduino, call the oled.begin(&Adafruit128x64, OLEDI2C, OLEDRESET) function [48]. In here, the arguments are the type of display, the $I2C$ address of the display and the reset pin of the display (pin 9 of the Arduino board). To clear the display, call the function clear() on the defined variable and, to print, call the print(message) function on the defined variable, with message being the content to print [48].

This library also allows for font changing and font size changing but still manages to be pretty lightweight by allowing only the printing of ASCII characters.

To test the component and to learn how to interact with it in an Arduino environment, the example program oled.ino was used [3, 34].

## 3.11 HC-SR04 Ultrasonic Obstacle Detecting Sensor

The first of the sensors installed into the Alphabot2-Ar is the HC-SR04 ultrasonic obstacle detecting sensor. This sensor is installed into the Alphabot2-Base $PCB$ and it is a plug and play component. Once installed in the front of the $PCB$, this component faces outside the robot and is ready to be utilized. This component is plugged directly into the Arduino board, meaning there is no need to access neither the $ADC$ nor the $I/O$ expander. The pins utilized are the 2 and 3. Plugged into pin 2 is the receiver sensor, also known as ECHO and plugged into pin 3 is the emitter sensor, also known as TRIG (trigger). The TRIG sensor is the one that emits the ultrasonic waves and the ECHO sensor is the one that detects them.

The way this ultrasonic sensor works is by having the TRIG sensor emit ultrasonic waves which are then reflected by objects and obstacles in the vicinity of the robot. The ECHO sensor then senses the reflected sound waves. This allows, through the measurement of how much time passed since the first wave was emitted by the TRIG sensor and the first reflected wave was received by the ECHO sensor, the calculation of the distance at which an obstacle is from the robot. The way this calculation is done is by having the following formula:

$$distance = \frac{duration * 0.034}{2} \tag{3.3}$$

where the duration parameter represents the time measured between the first wave was emitted and the first reflected wave was detected. 0.034 is the equivalent of the speed of sound (340m/s) in cm/microsecond. The unit cm is used because it is the most useful when working with a robot of this scale in a race track of the scale used to develop this work. The unit microsecond is used because the duration parameter is presented in microseconds. Finally, the 2 is used to divide the final distance value by 2, as the distance travelled during the duration time frame is equivalent of the sounds wave going and coming back, meaning it travelled twice the distance, hence the division by 2.

To operate this sensor in the Arduino environment, it is necessary to use the digitalWrite(pin, value) function, that can either send a LOW or HIGH value, depending on whether the pin should be deactivated or activated. The pin parameter should be the TRIG one. The receiver, also known as ECHO, is activated when it detects the reflection of these ultrasonic waves. By using the pulseIn(pin, state) function, the Arduino waits for the pin indicated as argument to go from the state indicated as argument to the other state. It then returns the duration in microseconds. In the case of the ultrasonic sensor, the ECHO has a HIGH state as soon as the TRIG begins emitting ultrasonic waves and that state goes to LOW as soon as the ECHO detects an ultrasonic wave.

The accuracy of the ultrasonic sensor is the interval [2; 400] cm. Any value lower than 2 cm or higher than 400 cm should be disregarded as it does not correspond to a correct distance measurement.

To test the component and to learn how to interact with it in an Arduino environment, the example programs Ultrasionc_Ranging.ino, Ultrasionc-Infrared-Obstacle-Avoidance.ino and Ultrasionc-Obstacle-Avoidance.ino were used [3, 34].

## 3.12 ST188 Infrared Obstacle Detecting Sensors

The second of the installed sensors for obstacle detection are the 2x ST188 $IR$ obstacle detecting sensors. These sensors are, like the ultrasonic ones, installed in the Alphabot2-Base $PCB$.

These sensors are activated as soon as they detect an obstacle, either individually or both of them simultaneously. They are both connected to the PCF8574 $I/O$ Expander, so accessing them must be done using the Wire.h library. These sensors have, each of them, a potentiometer associated. These potentiometers are located on the underside of the Alphabot2-Base $PCB$ and serve to tune the sensibility of the sensor. Rotating them changes the distance to which the sensors react and the sensibility can vary from 0.5 cm all the way to over 2 m. It is important to find the exact amount of sensibility desired, since changing this during the operation of the robot cannot be done with a remote control and must be done via the potentiometers. In Figure 3.5, it is possible to find the potentiometers' configuration used for this work.

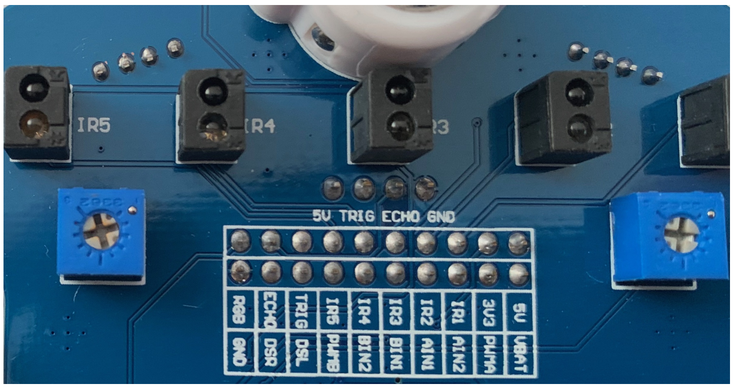


Figure 3.5: ST188 Potentiometer Configuration.

On the other hand, it is possible to alter the sensitivity on the fly, without having to shutdown the robot nor recompile and restart the Arduino code.

These sensors work in a similar way to the sensors that make the line detection. However, these infrared obstacle detecting sensors emit a signal to the Arduino when the photo sensor, part of the infrared sensor assembly, detects the reflected beam, and the change in sensitivity changes the amount of reflected light necessary to trigger the signal to the Arduino. This is contrasting to the line detecting infrared sensors that emit a signal that is proportional to the amount of reflected light that is detected. But, like the line detecting sensors, so do these obstacle detecting sensors detect more reflected light of a lighter surface than a darker one. Since these sensors are not connected to the $ADC$, one cannot receive a value that corresponds to the proportion of light reflected. Hence, only a signal saying sensor activated or not activated is sent.

Due to the way these sensors are connected to the Arduino, it is not possible to measure distances using these sensors. Firstly, these sensors use light to detect obstacles, as opposed to the ultrasonic ones that use sounds. This makes the calculations much harder since the speed of sound is far slower than the speed of light. By having a faster speed to measure a similar distance, the time the light beam takes to travel that distance is much less than the sound wave needs to travel the same distance. This makes the timing mechanisms present in the Arduino board not accurate and precise enough to allow these measurements of such small time frames between the first light beam being emitted and the first reflected light beam being detected. And also, due to the fact that these sensors are connected to the $I/O$ expander and a single sensor encompasses both the emitter and the receiver into a single unit which is connected to one pin in the expander, it is very difficult to independently control the emitter and the receiver to see how much time passed, specially since the pulseIn function cannot be used (because the sensors are not directly connected to the Arduino).

To access these sensors, it is first necessary to know the addresses. In a similar way to the joystick and buzzer, it is possible to quickly identify the address codes for each of the 2 $IR$ obstacle detecting sensors. By consulting [37], it is easy to see that the left $IR$ sensor uses pin P7 and the right $IR$ sensor uses pin P6. This is equivalent to having the left $IR$ sensor with address 01111111 (= 0x7F) and the right $IR$ sensor with address 10111111 (= 0xBF). The address for both sensors being active (both detecting an obstacle) is calculated by having both P6 and P7 pins set to 0 in their address code bits. This equals 00111111 (= 0x3F).

After having the addresses defined, since these sensors only send information and don't receive it, it is necessary to use the Wire.h library in a similar way to the joystick. The Wire.begin() function must be called, but then to receive data from the expander, it is only necessary to call the Wire.requestFrom(device, bytes) function [44]. In this call, the device is the address of the expander (0x20) and bytes is equal to 1 (= 8 bits), since the expander is an 8-bit one. After making the call, using the Wire.available() function allows the verification if any data was received and, if so, using the Wire.read() functions returns the data sent back by the expander [44]. This data corresponds to which sensors were active at the time of the call of the requestFrom function. It can be either the left $IR$ sensor, the right $IR$ sensor or both.

With this information, changes to the behaviour of the robot can then be made. If the behavior is not going as expected and the code seems to be correct, the problem can always be caused by the potentiometers not being set to the desired sensitivity.

To test the component and to learn how to interact with it in an Arduino environment, the example programs Infrared-Obstacle-Avoidance.ino and Ultrasionc-Infrared-Obstacle-Avoidance.ino were used [3, 34].

## 3.13 ITR20001/T Infrared Line Detecting Sensors

The last set of sensors installed in the Alphabot2-Ar mobile robot are the 5x $IR$ line detecting sensors. These sensors are placed on the underside of the Alphabot2-Base $PCB$. They are used to detect a color over a contrasting one. The most common use for them is to detect a line, which guides the movement of the vehicle, over a given background. To ensure maximum contrast and best possible efficacy in the behaviour of the sensors, a white background with a black line or a black background with a white line are usually the combinations used. For the work developed in this paper, a black line over a white contrasting background was used.

These sensors work by emitting a beam of infrared light, which is then reflected on the surfaces which the beam encounters. Depending on how dark or light the surface is, the less or more light it reflects. A darker surface will absorb more infrared light and reflect less while a lighter coloured surface will absorb less infrared light and reflect more. This difference in the amount of reflected light is enough for the sensor to identify a light coloured surface and dark coloured one. This distinction is done via the values returned by the $ADC$ after converting the voltage sent by the sensor. A higher voltage sent by the sensor equals a higher reflected amount and, hence, a lighter coloured surface. A lower voltage sent by the sensor to the $ADC$ equals a lower amount of reflected light and so, a darker surface. These sensors operate on a voltage of 3.3V, meaning they can send any voltage ranging from 0 to 3.3V. The $ADC$ then compares the voltage each sensor sends with the reference voltage and converts the final result to a digital one. Since the TLC1543 is a 10-bit $ADC$, the final converted values range from 0 (when it detects the line with the maximum intensity) to 1023 (when no line is detected). This value is then received by the Arduino board.

The operation of the $ADC$ has already been detailed previously. However, to read the value of each sensor, an address code for each sensor must be provided. To see which sensor has each value, it is only necessary to analyze [37]. According to [41], pins A0-A4 have the address codes 0-4. After having the address codes, all it takes to get the values in the range of [0; 1023], corresponding 0 to 0V(darkest surface detected) and 1023 to 3.3V(lightest surface detected), is to access the $ADC$ for each sensor, by sending the sensor's address to the $ADC$, as per explained in the beginning of this chapter.

After having the values for each of the sensors, it is easy to understand that the sensor which presents the lowest value and that presents a significant decrease over the remaining sensors values' is the one placed closer to the line. The obtained values should then be normalized to create a uniform base line and then a weighted average should be performed, with each sensor having a weight associated to it. This weighted average allows for a precise determination of the robot's position in relation to the line.

Finally, using the $PID$ algorithm allows the robot to adjust its speed in the smoothest and quickest possible way to correct the error of its position, in regards to the line, to 0. An in-depth analysis of the information regarding the sensor, present at [49], was also made to complete the provided information.

To test the component and to learn how to interact with it in an Arduino environment, the example programs TRSensorExample.ino and Infrared-Line-Tracking.ino were used [3, 34].

## 3.14 WS2812B RGB LEDs

One other component attached to the Alphabot2-Base $PCB$ are the 4x WS2812B $RGB$ $LED$s. These $LED$s are connected to the Arduino board via pin 7 [37]. Similar to the $OLED$ display and the $IR$ receiver, the $LED$s must also be managed with the help of an external library. This is necessary because of the extremely sensitive voltage settings that the $LED$s must have and the fast access timings. The chosen library is the Adafruit_NeoPixel.h [50].

This library is considered the standard for Arduino $LED$ management libraries. It is an easy to use and lightweight library, with support for multiple $LED$s connected in one pin, as is the case with the Alphabot2-Ar. To utilize this library, simply create a variable of type Adafruit_NeoPixel, that has to be initialized with a call to Adafruit_NeoPixel(4, PINS, NEO_GRB + NEO_KHZ800) [50]. This function takes as arguments the number of $LED$s, the pin to which they are connected and the type of $LED$s. After having initialized the variable, all that's necessary is to call the begin() function on the variable [50]. Then, to set a given colour, simply call the function setPixelColor(i, red, green, blue) on the created variable [50]. The arguments for this function are the number of the $LED$ (which in the case of the Alphabot2-Ar ranges from 0-3), the amount of red in a scale of 0-255, the amount of green in a scale of 0-255 and the amount of blue in a scale of 0-255. After setting the colour for a given $LED$, calling the function show() on the variable makes the LED emit the chosen color [50].

To test the component and to learn how to interact with it in an Arduino environment, the example program W2812.ino was used [3, 34].

## 3.15 N20 Micro Gear Motors

Finally, to conclude this robot analysis chapter, the motors are introduced. The Alphabot2-Ar possesses 2x N20 Micro Gear DC Motors connected to the Alphabot2-Base $PCB$. These motors are rated for 6V and can handle up to 600RPM. Connected to each motor is a rubber wheel with 42mm in diameter and 19mm width.

Similar to the Mona robot used in [29], so too in the Alphabot2-Ar "The rotational speed for each motor is controlled individually using pulse-width modulation (PWM). Each motor is controlled separately by an H-bridge DC motor driver, which requires approximately 74 mW to operate. As the motors are directly powered by the on-board battery, any voltage drop impacts the speed of the robot." [29].

To calculate the theoretical maximum speed of the robot, one needs only to consider that the maximum revolutions per minute the motor can handle is 600, which is equivalent to 10 revolutions per second. With this fact in mind, one needs only to know the distance equivalent to one revolution, multiply it by 10 and the speed in units/s is achieved. Bearing in mind the fact that the wheels have 42mm = 4.2cm in diameter, it is easy to calculate the perimeter of the wheels. Using the formula:

$$perimeter = \pi * diameter \tag{3.4}$$

the diameter can be calculated to be approximately 13.2cm. This means that, one revolution equals 13.2cm and 10 revolutions equal 132cm. This means that the theoretical maximum speed of the motors, without the robot and with the same tires, in m/s, is 1.32m/s.

It is important to note that this speed is just the theoretical maximum speed of the motors, when equipped with the tires present in the Waveshare Alphabot2-Ar robot. It does not take into account the weight of the robot, possible power losses and errors and code implementations that limit the performance of the motors, among other factors. As such, the maximum practical tested speed of the robot, when utilizing the Waveshare example programs, was around 60cm/s.

The motors connect to the Arduino board via the pins A0, A1, A2 and A3, alongside pins 5 and 6 [37]. The pins A0 and A1 connect the forward and backward circuit, respectively, for the left motor [37]. The pins A2 and A3 connect the backward and forward circuit, respectively, for the right motor [37]. Pins 5 and 6 are used for the $PWM$ speed pin, for the right and left motors respectively [37]. To make the robot move forward, it is only necessary to call the digitalWrite(PIN, HIGH) function to the A0 and A3 pins, with the value HIGH. This will ensure the forward circuit is closed and the motors are moving forward. To make the robot move backwards, simply close the A1 and A2 pins, by calling the digitalWrite(PIN, HIGH) function on them and the digitalWrite(PIN, LOW) on pins A0 and A3. To modulate the speed, it is only necessary to call the analogWrite(PIN, SPEED) function to pins 5 and 6. Each wheel may be set to a different speed. $PWM$ is used to set the speeds on each wheel.

”Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between the full Vcc of the board (e.g., 5 V on Uno, 3.3 V on a MKR board) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of ”on time” is called the pulse width. To get varying analog values, you change, or modulate, that pulse width.” [51]. By using $PWM$, it is possible to set a speed value to the motor as if it was plugged to an analog pin, even though it is plugged to a digital one.

To test the component and to learn how to interact with it in an Arduino environment, the example program Run_Test.ino was used [3, 34].

## 3.16 Analysis Conclusion

After having analysed every component that make up the Alphabot2-Ar, it became possible to draw some conclusions regarding the robot and its intended use in a classroom environment.

Before the conclusions were reached, every component was tested in isolation. This means that every sensor and actuator were tested using both the example programs provided by Waveshare, as well as programs developed during this work, both with and without external libraries. These tests concluded that the components worked as expected, when functioning in isolation, and that the results of their functioning met the requirements for an operation inside a classroom.

However, before deciding upon the adequation of the robot as a whole system for the intended project, it was necessary to test all the components working simultaneously. For this, the example program Line-Tracking.ino was used [3, 34]. Small alterations were made, to better fit the intended purpose of testing the solution, such as changing the speed parameters and the constants for the $PID$ algorithm.

After this analysis, that consisted in seeing if the sensors were accurate, the motors worked continuously during periods of time ranging from 20-30 min uninterruptedly, the batteries lasted the period corresponding to a laboratory class and the peripherals were working alongside the movement of the robot, the conclusion that the robot was fit for the intended use in this project was reached. This conclusion was only drawn after analysing how each component worked, how it interacted with the Arduino board and the other components, how it performed individually and also how it performed when the system is working in conjunction.

With this conclusion, the robot was chosen as the one to be used in the $ACIC$ classes beginning with the school year 2021/2022. It met the necessary requirements, of presenting a similar platform to the ones used in other reference schools and also having a pedagogical interface suitable for introductory students.

The next step was to develop the laboratory guides that would guide the work of the students, including the necessary theoretical contents for them to be able to implement solutions that work with the sensors.

# Chapter 4

# Implementation

After the analysis of the mobile robot indicated that it was suitable for an utilization in an educational environment, it was necessary to develop laboratory guides. These laboratory guides needed to meet some requirements.

They should be designed as to distribute the work load into 3-4 3-hour laboratory sessions. For each session, the laboratory guide should provide an overlook into the work being developed during the session and a theoretical introduction that covers the topics necessary to have the knowledge needed to interface with the sensors and actuators of the robot.

Then, a practical activity should be detailed by the laboratory guide, that encompasses the theoretical content provided in the guide with the actual components of the robot. These activities consist in having the robot perform some type of behaviour, such as following a line, detecting obstacles, interacting with peripherals, among several others. To have the robot perform the required activity, the students must program a solution using the Arduino $IDE$ and the Arduino Reference Language.

After having programmed and tested the solution that makes the robot perform the activity, some theoretical questions are asked, that cover both the theoretical content of the solution as well as explanations regarding the practical behavior of the robot.

In order to guide the development of a possible solution, a base code file is delivered to every student, upon which the code for the solution of the laboratory guide should be developed. These base code files include the definition of certain functions, the addition of code comments that guide the development and also the definition of variables, constants and code structure that allows the student to better understand how to implement a solution. However, this isn't mandatory. These laboratory guides were developed so that every guide builds upon the solution of the last one. This means that students can opt to continue to develop their work on the the file where they developed the solution to the previous laboratory guide or to develop the solution in the provided base code file. This approach allows students who have not been able to successfully complete the laboratory guide of the previous session to see how the correct implementation is made and to then be able to proceed the work without being left behind.

The final objective of this approach of having laboratory guides build upon the work of the last guide is to allow students to reach the final laboratory session and have a single code file that controls all the components of the robot. By doing so, they are able to understand how each component that constitutes the Alphabot2-Ar works and to have a complete solution for the operation of the robot as a complete system. By having this, students are then able to participate in a final activity, at the end of the last laboratory guide [20], where they must race their solutions to see which one is better. This race encompasses several aspects that are explored below.

For every laboratory guide, a complete solution file was developed. This complete file serves as benchmark for the Professor to compare the students' solutions and to evaluate them. It may also serve to clear any possible implementations doubts that the students may have.

Laboratory guide 0 [17] is meant to help students identify each of the components present in the Waveshare Alphabot2-Ar and show them how to assemble the components correctly in order to have the robot fully functioning. The pedagogical goal of this laboratory guide is to show students how to correctly handle, assemble and identify common components to Arduino systems.

Laboratory guide 1 [18] is meant to have students make the mobile robot follow a single black line placed over a white background, by utilizing the $IR$ line detecting sensors placed beneath the robot. To do this, students must use components such as the $ADC$ and motors and use algorithms such as the weighted average, data normalization and $PID$ algorithm. They must identify the best $PID$ multiplier parameters for specific speeds while having implemented some functions that were missing or incomplete in the provided base code file. The pedagogical goal of this laboratory guide is to have students learn the basics of real time control of a mobile robot, including learning how to utilize control algorithms to change the behavior of the robot, and to have the students learn about the operation of an $ADC$.

Laboratory guide 2 [19] is meant to have students make the mobile robot detect and avoid obstacles placed in front of it, first by reducing the robot speed and then by stopping the robot entirely. To do this, students must utilize the $I2C$ protocol for communication, through an external library, as well as components such as the $I/O$ expander and $IR$ (for stopping the robot) and ultrasonic (for reducing the robot speed) obstacle detection sensors. If the ultrasonic sensor detects an object within a distance (calculated using time intervals between ultrasonic waves are emitted and detected) smaller than a predefined threshold, the robot should slow down and when the $IR$ sensors detect an object, the robot must stop completely. The pedagogical goal of this laboratory guide is to have the students learn how communication protocols such as $I2C$ work and also how to override default behaviors with data obtained from different sources.

Finally laboratory guide 3 [20] is meant to have students make the mobile robot interact with its peripherals, both internal and external. Through the interaction with these peripherals, the robot should change its behavior depending on the input received and communicate, in real time, with several components such as an $OLED$ display and $LED$s. To do this, students must utilize some other external libraries as well as further components of the system, such as the $IR$ receiver and the batteries. They must also calculate battery voltages and the voltage flowing through the Arduino board, process commands sent in real time to the Arduino, both via the $IR$ remote control as well as the joystick, have the calibration process automatized and interact with the buzzer, $OLED$ display, $LED$s, among other components. The pedagogical goal of this laboratory guide is to teach students how to process commands sent to the Arduino in real time, while maintaining the normal behavior for all the components and also to teach students the concepts such as battery and Arduino voltage control and voltage dividing.

These laboratory guides were meant to be tested and evaluated using a group of selected students, which would simulate the target audience for these guides (since the $ACIC$ course was or will be part of their master's degree). However, due to the pandemic, this work was not developed within any of the $IST$ campi. Adding to that, restrictive measures were in place during most of the time of development of this work. This means that it was not possible to neither assemble a testing group nor have students interact with the robot in order to evaluate the laboratory guides. As such, these laboratory guides have only been tested by the author of this document, although feedback from one other former student of $ACIC$ has been received, with small changes that were implemented in the final version of the laboratory guides.

However, if it had been possible to have real students test the laboratory guides, the measured metrics would have been the time taken to complete the guides, how many times students would have visited websites not included in the references section of each guide as well as the results of the laboratory guides, both in terms of the code produced as well as the answers to the theoretical and practical questions. Adding to these metrics, any additional feedback submitted by the testing students would have been taken into account.

## 4.1 Environment Setup

Before starting the work with the Alphabot2-Ar, the environment where the robot will operate needs to be setup. This is true for when the work depicted in this document was being developed as well as for the students who are going to do the activities in the laboratory guides.

The first aspect of the environment is that it must have room to deploy a race track on which the robot will circulate. For the testing of the robot and the development of the laboratory guides, 2 racetracks were used. These 2 racetracks were designed to test different aspects of the mobile robot. The first one was designed to test the robots' ability to avoid crashes and at the same time seeing how well the robots behave when they have to deal with multiple crossing lines drawn over a single area. The second track's design simulates a real-life racetrack. As such, it has long straights to allow the robots to reach a high top speed and also many corners that vary in terms of length and degree. These corners are meant to take the robots capabilities to the limit and test the performance of the semi-autonomous solution for the laboratory guides.

When it comes to the materials the track should be made of, several were tried and experimented. When testing Styrofoam, it was visible that the traction it gave to the tires was not optimal and the worst part is that the electrical tape being used to delimit the track did not adhere well to it. The final choice was to use scenery paper. Scenery paper has a high gramage, while remaining flexible enough to be folded and stored. It also has a yellowish tone to it, as opposed to the complete white of the Styrofoam. However, during the tests, this reduction in the contrast level between using the white of the Styrofoam and the white-ish of the scenery paper did not present any visible change in the behavior of the robot. By using this paper, the robot had excellent traction, while the electrical tape adhered perfectly to it. Each of the tracks were assembled in a 2m x 2m square of scenery paper.

To delimit the track, a couple of ways were considered, including black markers and electrical tape. The markers allowed for an easier construction of the track but the width of the tip was too thin for the robot to detect successfully. The choice went then to the electrical tape. Both 15mm and 19mm wide tapes were tested and early tests showed a higher cornering accuracy from the behalf of the robot when using the wider 19mm tape. So that tape became the chosen one. In Figures 4.1 and 4.2 it is possible to find the designs for both of the racetracks.

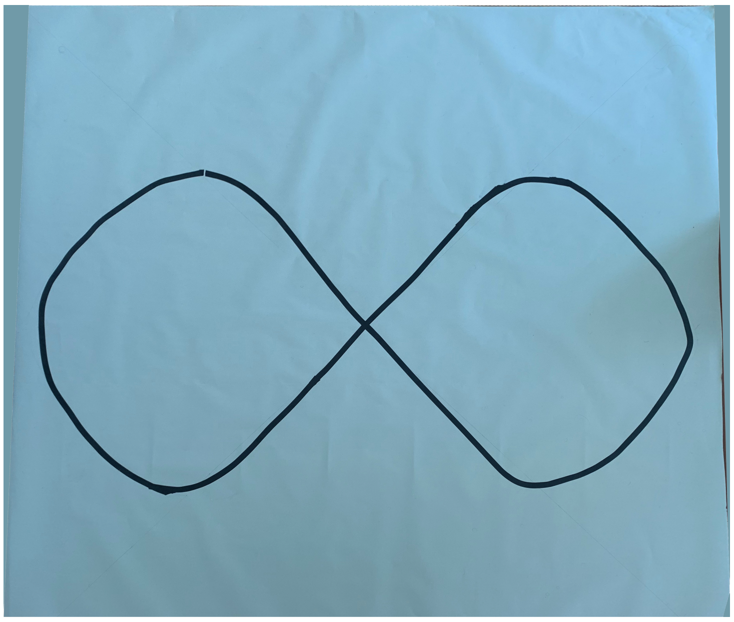


Figure 4.1: Racetrack 1.

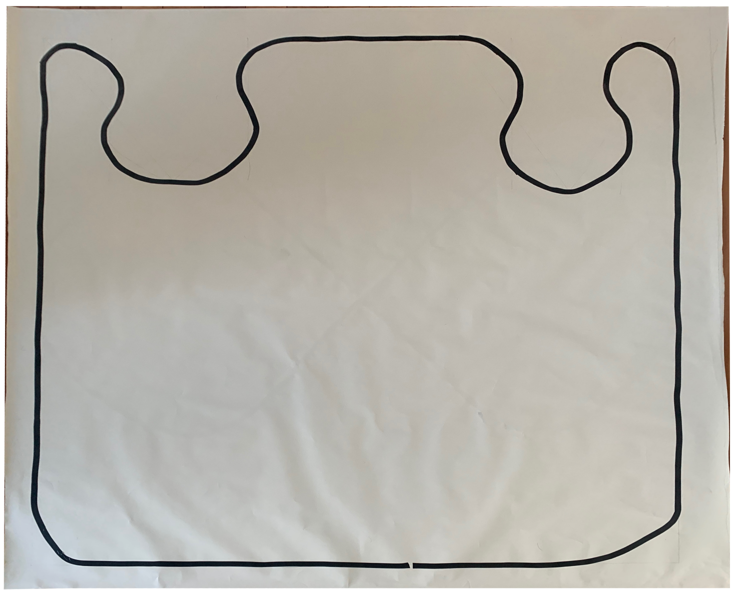


Figure 4.2: Racetrack 2.

The second aspect of the environment to bear in mind is the hardware necessary to program the mobile robot. During the development of this work, Arduino $IDE$ version 1.8.13 was used on a Apple Macbook Pro Retina 15.4' Mid-2014 laptop computer with 16GB of $RAM$, a 2.5 GHz quad-core Intel Core i7 and running on an $SSD$. This computer was running macOS Big Sur 11.3.1. The $USB$ cable provided with the robot was the one used to connect the Arduino board to the computer. In order to recharge the 2 ICR 14500 batteries, the XTAR MC2 Plus battery charger was used, connected with the provided micro-$USB$ to $USB$ cable. An internet connection is not required for programming the robot. It is simply necessary to download the external libraries used in some of the laboratory guides. To setup the Arduino $IDE$, it is only necessary to install it and, if the board is not detected when plugged in to the computer, download additional drivers from the Arduino website.

These are the requirements for the environment to be setup in order to successfully work with the Alphabot2-Ar. Of course, variations of the setup presented here can be deployed and still work for the intended purpose. Bear in mind that the setup where the laboratory guides were developed was deployed to be as similar as possible to the setup where the students will program the Alphabot2-Ar to perform the tasks asked in the laboratory guides.

## 4.2 Laboratory Guide 0

The first laboratory guide [17] is an introductory guide that serves as assembly instructions for the Alphabot2-Ar. It includes a description, with photographs, of every component present in the box of the mobile robot. It presents a diagram of the components present in both the Alphabot2-Base and Alphabot2-Ar $PCB$s. It also features a diagram of the components of the Arduino UNO Plus board. This way, the students are familiar with every part of the assembly that makes up the Alphabot2-Ar mobile robot.

After introducing all the components, assembly instructions with pictures are given. These are divided into assembly instructions for the Alphabot2-Base and Alphabot2-Ar chassis $PCB$s. After the instructions, pictures of the assembled robot are shown. In the end of the guide, there are some recommendations for the students to mind when dealing with this robot and when assembling and removing components. These warnings are meant to protect both the students and the hardware. By following the recommendations, students will guarantee a safe experience when dealing with the electronic components of the robot.

Also included in the guide are some references used when the compilation of the guide was made. Students can consult these references to find out more information about the robot or to clarify any possible doubts.

This guide is not intended to take a whole laboratory session to complete, but it is rather intended to be carried out in the beginning of the first laboratory session or at home, before the laboratory sessions begin.

## 4.3 Laboratory Guide 1

The laboratory guide 1 [18] is the first full-length laboratory guide that the students will have to complete. The development of this laboratory guide had in consideration the fact that this would be the first time many of the students working on the laboratory would be interacting with components such as Arduino boards, $ADC$s, $I/O$ expanders, among others. Due to this fact, the theoretical introduction focused on how these components work and on how they interact with the Arduino board.

The main objective of this laboratory guide is to have the robot following the line around the racetrack. By having this goal achieved so early on in the laboratory sessions, students are then able to, in the following sessions, fine tune their implementation, if they so desire, to have the robot follow the line as accurate and precisely as possible.

The goal of this laboratory work is to have the mobile robot follow a black line placed above a white surface. With the robot assembled, students will now have it use its sensors to navigate its surroundings. In this laboratory, students make use of the 5 infrared line sensors to make the robot follow a black line, placed over a white surface.

Before the actual work, they are given a theoretical introduction to the concepts needed to have the robot follow the line. A file containing the base code [24] is provided, upon which the work should be developed. The theoretical introduction first approaches the $ADC$. Since the 5 $IR$ line detecting sensors are connected to the $ADC$, students are presented with the logic behind the functioning of the $ADC$. A depiction of the pins used by the $ADC$ and their function is explained. Included in the base code file is the implementation of the readADC(address) function, that takes the address of the desired component to be accesses as argument and returns the value, in the scale of [0; 1023]. This function is already provided for the students because the operation of the $ADC$ can be quite complex for an introductory student. And since failing to correctly implement this function would lead to the robot not being able to follow the line (leaving the students stuck on the first task with the robot), it was decided that this function would be provided to them in the base code file. However, questions regarding the functioning of the readADC function and the function of the $ADC$ in general are asked in the end of the guide.

After explaining the functioning of the $ADC$, students are presented to the concepts of data normalization and weighted average. This theoretical introduction explains why data normalization is necessary for the best functioning of the line following algorithm and shows the formula necessary to calculate the normalized value for each of the [0; 1023] values returned by each of the sensors. It is also in this introduction to data normalization that the students first face the concept of calibration. For the data normalization algorithm to work, a calibration is necessary. This calibration occurs before the normal functioning of the robot of following the line. What this process does is record the highest and lowest value read for each sensor, in order to set those as the maximum and minimum values returned by the data normalization process. This maximum and minimum values work on a given scale. For the provided base code, the scale is [0; 100] where 0 corresponds to the minimum value and 100 to the maximum value.

In order to calibrate the robot, it must be rotated around the itself and over the racetrack, on a section where each of the sensors can capture both the line and the background. The ideal distance to the track that the robot should be rotated at is circa 2-3 cm. This process lasts about 10 seconds but this duration can be changed by changing the value attributed to the constant CALIBRATION_TIME, in the base code file.

After this introductory content on data normalization and the calibration process, the students are introduced to the concept of weighted average. It is explained how to calculate the weighted average and how it works to provide the robot with its current position in regards to the line. For this laboratory, it is considered that a centered robot (line directly above infrared sensor $IR3$) equals a position of 0. And for each deviation of the line in terms of sensors means a deviation of 100 in the position. As such, if the line is below the 4th infrared sensor ($IR4$), the position of the robot equals 100. If the line is below the 2nd infrared sensor ($IR2$), the position of the robot equals -100. The limit positions are -200 (if the line is below the first sensor, $IR1$) and 200 (if the line is below the last sensor, $IR5$). These values are the values used for the weights of each sensor ($IR1$ = -200 to $IR5$ = 200). The formula for calculating the weighted average is then provided. By applying this formula, the students are able to retrieve the position of the robot in regards to the line, on a scale from [-200; 200], where the central position (line centered with the robot) returns a value of 0.

After having been introduced to the concept of weighted average, that returns the robot's position in regards to the line, the students are then presented to the $PID$ algorithm. This algorithm will allow them to calculate the speed to provide to each wheel in order for the robot to recenter itself to the line in the smoothest way possible.

This algorithm is based on the concept of error value, with this error value being the difference between the desired point (0 in the presented case) and the current position (in the presented case, it can be in the range of [-200; 200]). This error value is often referred to as the proportional value. After this value is obtained, the algorithm tries to correct this error, with not only 2 other terms but also, associated to each term, a constant that serves as multiplier in a final sum. Besides the proportional value, 2 other components make up the $PID$ algorithm: The integral value and the derivative value.

The integral value sums the error value over time. Essentially it serves as a cumulative sum that aims to have a bigger effect on the algorithm if the errors remain high during the execution time of the algorithm or that reduces its effect on the algorithm if the errors stay with a low value. Overall, it is present to correct the behavior of the algorithm if the proportional and derivative components are not making the robot stay center within the line. As such, and because the integral value is a cumulative sum, it grows quite fast. Hence, it must not have a high valued constant associated to it in the final sum of the algorithm.

Finally, the derivative value consists of the difference between the error present in the current measure and the error present in the previous measure. It essentially shows the evolution of the position of the robot in each iteration and, depending on how that evolution is occurring, affects the final value outputted by the algorithm. As such, a big absolute derivative value means that the robot is changing its position quite fast, and a small value means that the robot is changing position rather slowly. This means that the derivative value acts as a damper that tries to bring the error to 0, while preventing an overshoot. The bigger the constant associated to the derivative value, the stronger the damping will be while a smaller value will produce a weaker damping and produce more overshooting.

After calculating these parameters, there is one final component to the algorithm, the constants associated to each parameter. Each constant consists of a numeric value that is multiplied to each of the parameters and that affects the role of each parameter in the final calculation of the algorithm. In order to determine the best values for each of the constants, the proposed way is as follows:

1) Set all constants to 0

2) Increase the KP constant until the robot presents a steady oscillation in its movement across the line

3) Increase the KD constant until the oscillation disappears

4) Increase the KI constant until the oscillations begin to appear noticeable again

Bear in mind that these values are speed related, so any change in the speed of the robot means that a new tuning is required. Provided with the base code file are constants that exemplify one possible combination of values for KP, KI and KD for the given speed.

With the introduction of the $PID$ algorithm, the theoretical introduction is complete.

Following the introduction, the students are presented with the references used to compile the guide. They can use them to further investigate the subject and clear any doubts they may have. The same is true for the references present in the next 2 laboratory guides.

Afterwards, the laboratory work is detailed. The first task is to analyze the given code base, in order to understand well what the code does and what functions need implementing. Then, it is asked to the students to implement the calibrate(), readLine() and pidAlgorithm() functions, which are missing or incomplete. They must be completed in order to make the robot successfully go round the track. Advice on how to complete the functions are present, both in the laboratory guide as well as in comments spread throughout the base code file.

Finally, the last task asked is to fine tune the $PID$ parameters. A possible combination of values for KP, KI and KD for a SPEED of 50 will be provided in the base code. However, students are expected to present the best possible parameters, using the suggested method or any other that can be explained. They must do this for a set of given speeds, alongside an explanation for the method used to obtain the values of those parameters.

After doing this, which completes the practical part of the laboratory guide, it is asked for the students to answer 2 theoretical questions.

The first one is regarding the main loop function and asks to explain the process behind the calculation for the final speed value for each of the motors. For this question, it is expected that the students explain the part of the code where the new speed for each wheel is obtained, this being the pidAlgorithm() function and the main loop(), where, for the left motor, the speed subtracts the obtained value and, for the right motor, it adds the value. Then, students should explain the limitations for the speed not to go over or below the maximum and minimum allowed speeds, for both motors. This limitation is due to the maximum voltage that the engine may take. Then students should mention the analogWrite() function and what it does in order to send the speed to the motors. What it does is convert the value given as input to voltage that is sent to the motors. As not all the content necessary to answer the question is written in the laboratory guide, students are encouraged to research and investigate in order to provide the most complete answer.

The second question has to do with the readADC(address) function and asks the students to give a brief explanation as to how the access to the $ADC$ works, what each of the for loops does, how is the address value being decomposed into binary and sent to the $ADC$ and how the value received is being composed into an integer. The expected answer is for the students to detail the functioning of the function, associating each code block with the explanation given in the theoretical introduction. Since the code is presented in the same order as theoretical introduction, when it comes to the functioning of the $ADC$, the answer consists in translating the blocks of code into English and explain them with the help of the theoretical introduction.

## 4.4 Laboratory Guide 2

The second complete laboratory guide that the students are presented with is the laboratory guide 2 [19]. This laboratory guide develops on the work executed in the first laboratory work and now assumes a complete and successful solution for the first laboratory session. If one was not achieved, the solution can be found inside the base code [25] file provided for this second laboratory. If the students so desire, they can perform this laboratory work on top of the solution achieved for the previous laboratory work. It is up to the students to choose the best approach.

After having the robot successfully go around the racetrack, it is now asked to the students to prevent the robot from colliding into obstacles that may be present in the track. The robot should decrease its speed as it closes into the obstacle and if it gets too close to the obstacle, make a complete stop. To do this, students will use the ultrasonic sensor and infrared obstacle detecting sensors to prevent the robot from crashing into obstacles that are in front of it. The ultrasonic sensor should make the robot slow down once an object is detected that is closer to the robot than a predefined given distance. The 2 infrared obstacle detecting sensors, located at the front of the robot, should stop the robot when they detect an obstacle. For this work to be carried on, a theoretical introduction is given, once again.

This time, the theoretical introduction relies first on $I2C$ communications. This is necessary since the 2 $IR$ obstacle detecting sensors are connected to the Arduino via the $I/O$ expander, that is connected to the Arduino board via an $I2C$ bus and communicates via the $I2C$ communication protocol. An explanation is then provided on how the $I2C$ protocol works and on how to use the Wire.h library to communicate with the devices present in the expander.

After this, an introduction on how the $I/O$ expander works is given, in particular the fact that since an obstacle close to the robot would trigger the sensors, they would be active whenever an obstacle is detected and their address would be returned if a request is made while an obstacle is being detected by the sensors.

After this, a theoretical introduction into distance calculation is made. This introduction is necessary because students will be asked to calculate distances, using the ultrasonic obstacle detecting sensors, in order to slow down the speed of the robot. For this introduction, students are presented with the digitalWrite and pulseIn functions and on how to use them to measure the time between the TRIG sensor emits a sound wave and the ECHO sensor detects the reflected wave. After having this time, and with the formula for calculating distance presented earlier, it is possible to calculate the robot's distance to an obstacle.

After this theoretical introduction, students are presented with the references used for the guide and the general working precautions.

As for the laboratory work, first the students are asked to analyze the given base code file. After that, they must implement the obstacleDetectedInfraRed(), distanceMeasureUltraSound() and loop() functions, that are missing or incomplete. A suggestion is also given, in regards to the functionality of reducing the speed of the robot, as it closes in to an obstacle. For the loop() function, it has to be adapted so that the robot reacts to obstacles detected by the ultrasonic and infrared sensors. In the case of the ultrasonic sensor, the goal is to make the robot reduce its speed once it detects an obstacle that is closer to the robot than DISTANCE cm, being DISTANCE the constant defined in the base code file. The speed of the robot must decrease in a proportional and constant way as the robot closes in at the obstacle (Suggestion given - use a speed value that is proportional to obstacleDistance / DISTANCE). Students need to make sure that the robot does not stop completely, but instead, only goes down to minimum moving speed. For the robot to stop completely, the infrared sensors must detect an obstacle. This means that, even if the ultrasonic sensor does not detect any obstacle, if the infrared ones do, the robot must stop completely and immediately. The objective is to have the infrared sensors detect an object that is 3 to 4 cm away. The sensibility of the sensors can be fine-tuned using the potentiometers located at the bottom of the robot. When the sensors are active (detecting an obstacle), a green $LED$ next to each sensor will light up, meaning the corresponding sensor is detecting something.

After this, some theoretical questions are asked. The first question is regarding the minimum speed value for the robot to move and why is that value different than 0. For this question, it is expected that students answer it by mentioning the fact that the minimum speed value is the value below which the robot does not move. It is also expected that they mention this value exists so that the speed reduction caused by the ultrasonic sensors detecting an obstacle cannot go below it and they should also mention that it is different than zero due to the fact that, for the weight of the robot, only above this speed can the motors actually move the robot.

The second question is related to the fact that there are potentiometers for the infrared sensors and why do the sensors react differently to obstacles, at the same distance, when the potentiometers' value is changed. The expected answer should touch on the fact that the potentiometers exist so that the sensitivity of the infrared sensors can be changed, the fact that these regulate how much reflected light is enough to trigger the sensor and the fact that the sensors react differently to objects at the same distance, when the potentiometer's value is changed, because the adjustment of the potentiometer alters the amount of reflected light necessary to trigger the sensor. This means the same 2 objects, at the same distance to the robot, will always reflect the same amount of light.

The third question asks if it is possible, with the robot assembled as is, to measure distances using the infrared obstacle detecting sensors and to explain the reasoning. The expected answer from the students should touch on the fact that it is not possible to measure distances using these sensors. Firstly, these sensors use light to detect obstacles, as opposed to the ultrasonic ones that use sounds. This makes the calculations much harder since the speed of sound is far less than the speed of light. By having a faster speed to measure a similar distance, the time the light beam takes to travel that distance is much less than the sound wave needs to travel the same distance. This makes the timing mechanisms present in the Arduino board not accurate and precise enough to allow these measurements of such small time frames between the first light beam being emitted and the first reflected light beam being detected. And also, due to the fact that these sensors are connected to the $I/O$ expander and a single sensor encompasses both the emitter and the receiver into a single unit which is connected to one pin in the expander, it is impossible to independently control the emitter and the receiver to see how much time passed, specially since the pulseIn function cannot be used (because the sensors are not directly connected to the Arduino).

The next 2 questions require some practical work, as they ask students to present the values (SPEED, KP, KI and KD) that correspond to the maximum stable speed at which the robot can navigate the track, while also being able to detect obstacles, using the ultrasonic sensor, and reducing speed before crashing into the obstacle. The other question asks to present the value that corresponds to the lowest possible distance at which the robot still can reduce its speed and have the infrared sensors stop the robot completely before it crashes into the obstacle (using the KP, KI and KD defined in the previous laboratory class). Students should tune them, if necessary, to present a stable behavior following the line. These 2 questions require some hands-on with the robot and the track. For this second question, the parameters should be determined for 4 different speed values.

## 4.5 Laboratory Guide 3

The third and final laboratory guide is the laboratory guide 3 [20]. This laboratory guide develops, once again, on the solution achieved on the previous laboratory work. For this laboratory guide, the robot can already follow the line and also react to obstacles, decreasing speed with the ultrasonic sensor and stopping completely when the $IR$ sensors detect an object. Once again, students can choose to use their own code as base or work with the provided base code [26] file.

The goal of this laboratory work is to have the mobile robot interact with its peripherals, both internal and external. Through the interaction with these peripherals, the robot should change its behavior depending on the input received and communicate, in real time, with several components such as an $OLED$ display and $LED$s. The list of peripherals includes the $OLED$ display, the buzzer, the joystick, the $LED$s and the infrared remote control.

For the implementation to be well done, the robot must go around the track with the same precision as the 2 previous laboratory works and the same obstacle detection and avoidance capabilities from the laboratory guide 2, while at the same time being able to receive commands via the joystick and the remote control and communicating in real time with the $OLED$ display, the $LED$s and the buzzer.

The ultrasonic sensor and infrared obstacle detecting sensors should continue to prevent the robot from crashing into obstacles that are in front of it. The 5 infrared line detecting sensors should continue to keep the robot centered within the line.

In this laboratory, the $OLED$ display is used to display the battery status, view distances in the distance measurement mode and display the current values for the robot's speed, distance, kp, ki and kd parameters. The joystick is used to enable and disable the buzzer (similar to a car's horn), the distance measurement mode, the $LED$s and the display of the battery status. Finally, the infrared remote control will serve for a number of purposes: change the color of the $LED$s (between red, green and blue); enable and disable the $LED$s; increase, decrease and reset the distance parameter; increase, decrease and reset the speed parameter; display the current speed, distance, kp, ki and kd parameters; store 5 different presets for the speed, kp, ki and kd parameters for the robot to change in real time; reset the speed, kp, ki and kd parameters to their default values; enable the buzzer (similar to a car's horn); display the battery status and enable and disable the distance measurement mode.

The theoretical introduction for this laboratory guide begins with the use of external libraries. So far, the only external library needed was the Wire.h library, used to communicate with the $IR$ obstacle detecting sensors, through the $I/O$ expander. In order to control the $OLED$ display, the $RGB$ $LED$s and the infrared remote control receiver, external libraries should be used. The use of these libraries makes it easier to interact with these components and prevents errors from happening when dealing with the components.

The Wire.h library will still be used in this laboratory guide but instead of using it only to read from the $I/O$ expander, the students will now also use it so send information to the $I/O$ expander. This library will control the $I/O$ expander, to which are connected, besides the 2 infrared obstacle detecting sensors, the joystick and the buzzer. The joystick, like the infrared sensors, requires only the use of the Wire.h library to read from the $I/O$ expander, while the buzzer requires the sending of information to the $I/O$ Expander, via the Wire.h library. The use of the library for the joystick will be similar to the use of the library for the infrared obstacle detection sensors. The only change in the procedure is the addresses returned by the $I/O$ expander. Then, for the buzzer, the functions from the Wire.h library used to communicate with it are introduced and explained, in terms of their arguments and their functioning.

To control the $OLED$ display, the SSD1306Ascii library is used. Explained in the guide are the procedures to perform in order to utilize the library, whose functioning has already been explained in the robot analysis chapter.

To control the $RGB$ $LED$s, a library called Adafruit_NeoPixel.h should be used, and so the procedures to use it are also depicted.

Finally, to control the infrared remote control receiver, the IRremote.h library should be used and so the procedures are, once again, depicted. To follow the instructions to use the external libraries, an Internet connection must exist in order to download the content of the external libraries.

The next theoretical content introduced are the battery calculation procedures. To calculate the battery level from the 2x ICR 14500 batteries, an explanation similar to the one provided in the robot analysis chapter is provided. However, one other calculation is asked of the students. That calculation is the calculation for the voltage flowing through the Arduino board. Ideally, and to preserve the Arduino board, the voltage flowing through it should be 5V and should not drop too much below it. Since the batteries are connected to the Vin pin, a 5V flowing through the board is guaranteed. The Vin pin should receive between 7 to 12V in order to divide the voltage into a guaranteed 5V directly going through the pin, after being divided by the resistors associated with the Vin pin. Since 5V would equal 2.5V in each battery, and the safe discharge is set at 3V, a drop in this voltage shouldn't occur unless the batteries are extremely low. But since this measurement allows to see if there is any error with the Arduino board and, if the batteries discharge too much, it is at least possible to still see if enough voltage is being provided to the Arduino board, so it is a good idea to read this value. The function calculateArduinoVoltage() is already implemented, and the student should study it, alongside the web page explaining the function, located in the reference section of the guide, in order to understand how this measurement is being done.

Finally, 2 native Arduino functions are introduced. These functions are functions the students may find useful to use during their implementation of the solution to the laboratory guide.

The first function is the delay(amount) function. For some of the tasks asked in this laboratory guide, having a function that can stop the execution of the code for a certain amount of time might be useful. Say, to display information on the $OLED$ for only a certain amount of time or to prevent double reading signals from the infrared remote control, a stop of the execution is handy. That is the purpose of the delay(amount) function. It takes as an argument a single value, indicating the time, in milliseconds, that the execution must stop. After that amount of time has passed, the execution resumes as normal.

The other function is the millis() function. For some of the tasks asked in this guide, there is the need to control for how long a piece of code is executed. To do that, one of the easiest ways is by using the millis() function. The millis() function returns the amount of time that the Arduino has been powered up, in milliseconds. By adding to this value a specific amount of time, it is possible to, subsequently, compare a millis() execution to a previous variable consisting of millis() + someAmountOfMilliseconds. This way, it is possible to tell if someAmountOfMilliseconds have passed and, if so, proceed with the rest of the execution of the code. This function is particularly useful for the auto calibration feature discussed in the guide.

After this theoretical introduction, the references used and the recommendations are presented.

The first task asked in the laboratory work is to, once again, read the provided base code file in order to understand well what the code does and what functions need implementing. Then, students must implement the missing functions, which are incomplete or missing entirely.

The first feature the students should develop is the auto-calibration feature, where the calibration procedure is changed from a manual one, where the robot is rotated above the line, to one where the robot does it automatically. This should be done in the setup() function of the Arduino, right after the pin mode settings and initialization of all the variables. To automate this process, first the students must print in the $OLED$ display a message asking the user to push the joystick center button in order to start the calibration. To do this, it is necessary to implement the joyStickPressed() function. After the button is pressed, a message in the display should be presented saying that the robot is calibrating. Then, for CALIBRATION_TIME, the robot should automatically calibrate by rotating on itself above the line, bearing in mind that all sensors need to cross the line and also the background, so that all of them can record their minimum and maximum values. The robot must start and finish the calibration in the same place, so the rotations around itself should always be complete ones. A tip to do this would be to use the millis() function and rotate the robot on one direction and the other for the same amount of time. After the calibration is over, the robot should stop its movement, set the rotation of both its motors to forward and display a message on the screen asking the user to push the joystick center button to start the movement of the robot around the track. After the user presses the joystick, the $OLED$ has to be cleared. The $OLED$ display should be cleared whenever the robot is doing its normal operation and should only be on when the distance measurement mode is active or during short periods of time, with the robot stopped, when the user asks for the battery status or the status of the parameters.

The second feature asked of the students is to receive commands via the joystick and the infrared remote control. With every execution of the main loop(), students should check for infrared or joystick commands being pushed by the user and received by the Arduino. So, in the beginning of the loop, they need to check for both types of commands and store them in a variable to later compare with known addresses for commands. Having the joyStickPressed() function already implemented, all that's left for them to do is see if there is any new infrared command and, if there is, store it. Then, a list of commands and their asked feature is presented. Students must implement those in order to completely finish the practical work of the guide.

Finally, some theoretical questions are asked. The first is regarding the content of the calculateArduinoVoltage() function and the logic behind it. Students are encouraged to visit the references and other information sources in order to successfully answer the question. Students have already some comments present in the function that should help them to successfully understand what is going on inside the function. The only thing that remains to mention in the answer is for them to indicate what each function call does inside this calculateArduinoVoltage() function. A research of the Arduino Reference Language is enough to provide the answer.

The second theoretical question is regarding the fact that some of the pins of the Arduino need the digitalWrite() function and others the analogWrite() function and then they are asked to explain the differences between the functions, the pins and relate them to the concept of Pulse Width Modulation [51]. To successfully answer this question, students should mention the fact that some of the pins are digital ones, meaning they simply have 2 states: current flowing (5V) and current not flowing (0V). These pins should be used with the digitalWrite function. Other pins are analog pins, meaning they can receive any voltage value between 0V and 5V. As such, they should use the analogWrite function. To relate this with the concept of $PWM$, students should mention that "Pulse Width Modulation, or $PWM$, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between the full Vcc of the board (e.g., 5 V on Uno, 3.3 V on a MKR board) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width." [51]. A simpler answer may be provided, as long as it mentions that $PWM$ is a technique that simulates analog results using digital functions, by creating a wave. This wave is created by fluctuating the amount of voltage (between no voltage and maximum voltage) over a period of time till a wave resembling an analog one is created.

Finally, one last task is asked of the students. Since this is the last activity of the last laboratory guide, students are encouraged to compare the solutions each of them arrived to. As such, all the groups are asked to compete for the prize of best programmed and configured robot. To do this, they should build a track that starts in one place but ends in a different place. They must make sure it has curves and straights. In the end of the track, they need to place an obstacle. 5-7 cm before the obstacle, make a color marking next to the electrical tape serving as finishing line. The team whose robot completes the track the fastest and stops the closest to the target, without knocking over the obstacle wins. If the robot does not stop with the front of the robot over the line, it has not finished the course. They may use the remote control to alter the behavior and use whatever speed, distance, kp, ki and kd values the team believes is best.

This way, a healthy competition is created and a fun experience that culminates the programming of all the interfaces of the robot is presented. By adding this playful component, it is expected that students will further enhance the quality of their solution and their programming in order to present better results than their peers.

# Chapter 5

# Results And Evaluation

As the laboratory guides were being developed, so were Arduino code files that could serve as solutions to each of the laboratory guides. These code files had the provided example codes from Waveshare as base but most of the content was completely redone. Even the logic behind the implementation of the solution was completely re-thought. This way, an optimized and custom solution specifically made for these laboratory guides could be created.

In order to optimize the developed code, the use of external libraries was reduced to the minimum, leaving the lightweight default functions and implementations of Arduino to handle most of the tasks. Throughout the code files, memory usage was reduced to a minimum as well, in order to have as much memory as possible. In order to prevent reduced performance due to display content updates, whenever any command that requires the use of the display is called, the motion of the robot stops while the information is displayed in the $OLED$ display. Usage of loops is also highly selective, only being used where proved necessary. Conditional statements that prevent unnecessary code executions were also deployed throughout the code file. And to clean the code file and make it easy to read and also to prevent code duplication, all major functions of the robot were programmed into functions that can be called from anywhere in the program. Global variables that can be used by all functions were also implemented, to prevent the creation of unnecessary new variables. Finally, all the code is thoroughly commented and all constant values are defined as constants in the beginning of the file. Overall, all major good programming procedures were put in place to guarantee a code file that is of high quality, high legibility and high performance.

With this in mind, and after many hours of tuning and perfecting the code files, it is my believe that the provided code files can be used as benchmark for evaluating the students' solutions and also to clear any doubts that may arise from the development of the students' solutions.

After this, it became necessary to evaluate the solutions and measure how well they behaved in action. Since each code solution built upon the last laboratory solution, the last solution is a compilation of the previous 2 plus other features and enhancements. And since this solution is what is considered as a complete platform for use of the Alphabot2-Ar and all its components, it made sense to evaluate this solution and this solution alone in terms of its performance. If it behaves well, then the other 2 previous laboratory solutions will behave well as well. If it behaves poorly on tasks exclusive to this laboratory solution and well on the common tasks, then the other 2 solutions also should behave well on these tasks. If it performs poorly all around, then so do the previous 2 solutions, corresponding to the solutions to the previous 2 laboratory guides.

## 5.1 Evaluation Metrics

The following measuring metrics that will evaluate the solution and its effectiveness:

• Time taken to complete a lap

• Success at preventing a crash into the other robot

• Being able to complete a set of laps without derailing the track

For the time taken to complete a lap, the evaluation will consist on the improvement, in terms of lap time, when the SPEED parameter is increased (and subsequently the KP, KI and KD values changed as well). The measurements are made on both tracks, as to understand the solutions' ability to perform with a consistent speed value across tracks that push different aspects of the robot to the limit.

Then, to evaluate crash avoidance performance, the solution must reduce its speed upon detecting an obstacle with the ultrasonic sensor and the robot must stop completely when the $IR$ obstacle detecting sensor detects any obstacle. It must demonstrate clear efficacy at slowing down, to a complete stop if necessary, in order to prevent the crash.

As for the other metric, which states that the robot is able to complete a set of laps without derailing the track, the robot must present an accurate line path, without leaving it to be considered appropriate. This must occur for all the speeds tested for lap times. To do so a robot must never deviate from the line and if it does, it must be able to find the track again and resume the track course (for this to be possible, the deviation must be a small one). Of course small errors can occur and the sensors may not present reliable data with every single call. This is why, for every test, 10 laps will be lapped to measure the performance metrics. This allows the robot to have some degree of margin in its behavior.

In the end, for the evaluation metric of time taken to complete a lap to have a positive result, the lap times around the track must decrease as the speed parameters increase till the theoretical top speed value is reached. The theoretical top speed is the speed value to which, if the value is increased, the lap times do not decrease in an amount considered to be relevant. This happens because, for the robot to have a stable behavior, the $PID$ algorithm needs to reduce the robot's speed in the corners and also in the straights, if the robot starts to wiggle due to small lane curves present in the track. For these reasons, there is a speed value which, if increased, the time the robot takes does not decrease significantly. If the robot reaches that speed in testing and is able to complete 10 laps at that speed, while being able to prevent crashed and not derailing, with the lap times being faster than ones with a smaller speed value, then the solution is well implemented and can be considered effective as a platform to control the behavior of the robot and its components.

Aside from the evaluation metrics listed above, there are some concepts which were analyzed and studied for the final evaluation of the solution. They do not affect the conclusions regarding the effectiveness of the solution but give a better insight into how other aspects may affect the functioning of the robots. Such parameters are:

- Response time of the robot (both for the software and hardware)
- Real-time control of the robot
- Fault tolerance for the sensors and actuators
- Battery consumption and energy autonomy

## 5.2 Data Gathering

To gather data regarding the evaluation metrics, both designed racetracks were used. For each racetrack, a set of 10 laps were run for time measurements as well as another set of 10 laps for obstacle avoidance. This happened for 6 different speed parameters. These parameters include the value of the constants SPEED, KP, KI and KD for the laboratory guide 3 code solution. For each SPEED value, the parameters were calculated using the method defined in the laboratory guide 0.

For each speed value, the first 10 laps of each track were timed in order to have an average lap time. Deviations from the track were counted but no obstacle detection was tested. After these 10 laps, another 10 took place where times were not being measured but instead the obstacle detection and crash avoidance systems were being evaluated, for each of the speeds. To do this, the tracks were divided into 10 different sections and the obstacle, consisting of a small box the size of the Alphabot2-Ar, was placed in the middle of each section, consecutively. Each time the robot crashed into the box was counted and every deviation of the line was also counted. In the end the total of deviations (that were not automatically recovered by the robot) from the 20 laps was divided by 2 in order to present an average value for 10 laps.

This procedure was repeated on both tracks for each of the 6 speed configurations. Between each speed parameter change, the robot was fully recharged. The 6 speed configurations are present in Table 5.1.

Table 5.1: Alphabot2-Ar speed configurations used in this work.

| Speed | KP | KI | KD |
|---|---|---|---|
| 50 | 0.5 | 0.005 | 6 |
| 75 | 0.6 | 0.005 | 6 |
| 100 | 0.8 | 0.0001 | 8 |
| 115 | 1.3 | 0.0005 | 15 |
| 120 | 1.3 | 0.0005 | 15 |
| 125 | 1.5 | 0.001 | 20 |

Finally, for the other 4 metrics that were not used to evaluate the effectiveness of the solution but that were important to measure, the battery life was analyzed over the course of the testing sessions.

One single test, that consisted in seeing how many laps of the ”8” track the robot could do, at speed configuration 100, showed the robot could endure, before reaching the Safe Discharge Voltage, a little more than 0h45m of uninterrupted laps (considering the robot started with a full battery).

For the real-time control aspect of the robot, 10 testing laps where the robot was sent commands via the $IR$ remote control were made. Also during these laps, obstacles were put in front of the robot from time to time to test the reliability of the sensors and actuators while the robot was being issued constant commands. The results showed that the robot behaved as expected and no real-time control fault was detected, meaning the robot always responded to the commands, while still being able to go around the track without deviations and always responding to the obstacles.

For the fault tolerance aspect, the sensors and actuators did present some occasional faults but the system, as whole, is programmed in a way that occasional faults do not cause the system to fail. This happens because the requests for new data are constant and faulty data is immediately replaced by new data, which is to be considered correct due to the low fault rate of the components of the robot. This means that the generality of the data is to be considered when evaluating how well the system deals with faults, rather than occasional faulty data.

Finally, for the response time of the robot, no measurable delays were encountered in the solution, so the response time is considered more than adequate for the purpose this solution is destined to.

## 5.3 Testing Results

Tables 5.2 and 5.3 present the measurement results regarding the metrics considered for the effectiveness of the solution, for each of the tracks. Bear in mind that, when referring to recovered derailings, the situation considered is when the robot is, in its entirety, not above the line and manages to return to the line. No unrecovered derailings were measured for any of the tested speeds.

Table 5.2: Measurement results for the racetrack 1, "8".

| Speed | Avg. Lap Time | Crashes Into Box | Derailings |
|---|---|---|---|
| 50 | 19.145s | 0 | 0 |
| 75 | 12.679s | 0 | 0 |
| 100 | 9.492s | 0 | 0 |
| 115 | 8.755s | 0 | 0.5 |
| 120 | 8.559s | 0 | 0 |
| 125 | 8.497s | 0 | 0 |

The results presented in Tables 5.2 and 5.3 contain values measured over the course of 10 laps, with the derailing parameters being halved, since they were measured over 20 laps. The configured testing speeds only go till 125, because from the speeds 115 to 125, there was no noticeable difference in lap times, both in racetrack 1 and 2. Due to this fact, it was possible to conclude that the $PID$ algorithm had reached its speed limit when it came to the maximum speed possible to go around the corners. This means that the speed decreased to the same amount to go around the corners but also that, for the straights, the $PID$ algorithm was also reducing the speed. This would happen because, for these

Table 5.3: Measurement results for the racetrack 2, ”Professional”.

| Speed | Avg. Lap Time | Crashes Into Box | Derailings |
|---|---|---|---|
| 50 | 30.777s | 0 | 0 |
| 75 | 20.355s | 0 | 0 |
| 100 | 16.203s | 0 | 0 |
| 115 | 14.612s | 0 | 0 |
| 120 | 14.423s | 0 | 1.5 |
| 125 | 14.331s | 0 | 2 |

high speed tests, every tiny curve in the tape for the straight lines would be followed by the mobile robot, causing the robot to wiggle too much and lose stability. To prevent this, the $PID$ algorithm would reduce the robot’s speed so that it could go around the track in a stable way. This also means that the speed was reduced, in speeds 115, 120 and 125, to similar speeds all around the track. Hence why the time results were so close between them. For this reason, it was considered that 125 was the maximum speed configuration to be tested. Possible higher speed configurations can be determined and tested and could, possibly, present a stable behavior around the track and improve the lap times presented by speed 125. However, since that was not the goal when testing the solution, that work is proposed to be done as a future work or by the students working with the laboratory guides provided by this work. The important thing to keep in mind is that these values are mere examples and many more different ones can be developed and tested. These are tested values, though. This means that these are values that are assured to give the robot a fast, stable and precise behavior around racetracks.

The obtained results are significant to the conclusion regarding the use of this mobile robot to develop the laboratory works being proposed because they show how the robot behaves when presented with the tasks developed in the laboratory guides. The fact that the robot never crashed into the box, had a significant time decrease as the speed increases and that the amount of derailings can be considered insignificant, given the amount of laps, indicate that both the robot and the coded solutions for the labs are appropriate for the use proposed in the guides. The real-time performance and response to commands complement this conclusion, which is further discussed in the next chapter.

# Chapter 6

# Conclusions

Mobile robots have created a whole new way for machines to cooperate with humans. From exploring the lunar surface to moving heavy items across a factory's floor autonomously, mobile robots have become essential in multiple areas of our society. Through more than fifty years of evolution, these robots have become increasingly more reliable and safer, as well as more useful.

The goal of this work is to not only analyse the mobile robot in discussion but to also prepare content that allows students, who are approaching the subject for the first time, to have a complete yet practical introduction to the topic of mobile robots. Through the use of known technologies such as Arduino and also a simple and inexpensive sensor set, comprising infrared obstacle detection sensors, ultrasonic sensors and line-detecting sensors, it is expected that the results obtained can be easily scalable to more complicated mobile robots with better $I/O$ devices.

With the creation of this solution and these laboratory guides, it is expected that the educational use of mobile robots can be improved and made more attractive towards students. The improvements happen because professors and teaching staff will have a more stable base to present to students. The reduction in errors and the smoother behavior of the robot ensure that the students will have an easier and more precise learning experience when debugging and testing their experiments. As the platform will be based around Arduino and a pre-built system, the students will also have an easier time when learning how to interact with mobile robots similar to Alphabot2-Ar, by analyzing the solution presented by this work. The increase in the attractiveness of the subject happens because the final achieved solution and guides enable students to deal with an optimized solution. As such, fun and engaging games and races can be played and done by the students, with their robots, by having them develop even better navigational software on top of the framework. Concepts such as real-time movement control and decision making are introduced in a more attractive way which increases the students' engagement towards said concepts.

After the development of this work, several conclusions were reached.

Starting with the first one, it is believed that this robot presents a suitable platform for educational use in a classroom environment. The analysis made to the components, the measured tests, the similarities between this and other platforms used for university introductory robot class teaching and the opensource and pedagogic nature of the components, specially the Arduino board, make this robot the perfect choice for the desired intent. Even though other alternatives exist in the market, this robot is believed to be the best choice when it comes to number of different sensors and peripherals and also in terms of the price/quality ratio.

The second conclusion reached is the fact that the developed laboratory guides are on par with the curriculum for the $ACIC$ course and the contents taught and the way the laboratory sessions develop is similar to the laboratory sessions and course curriculums for robotic and embedded systems courses in reference universities. The analysis of several papers regarding how introductory robotics courses were structured in top universities showed a deep level of similarities with not only the $ACIC$ course but also with the way the laboratory classes were laid out in the laboratory guides. To contribute to this similarity, there is also the fact that the laboratory guides were only developed after the analysis of the papers, as to make sure it followed the standard norm when it came to educational activities to make students interact with mobile robots. The contents of the laboratory guides, both theoretical and practical, also are similar to the contents being taught at these universities.

After having analyzing the achieved results, the final conclusion reached is that the programmed solution is suitable to be used as benchmark for evaluation and comparison reasons. It is a solution that is ready to be used for this particular mobile robot, but one that is also easily adaptable to use with other Arduino-based mobile robots. This solution encompasses all peripherals, components, sensors and actuators, except for the XBee Bluetooth communication board. This solution also guarantees, after the multiple testing sessions, a safe, reliable, accurate and fast functioning of the mobile robot around racetracks, with easy to adjust parameters and adaptable behavior for any track consisting in a line drawn over a contrasting background.

Overall, this work presented many challenges during its development. Many of the components did not have a easy-to-read and easy-to-understand documentation, the Waveshare support website did have some information but there was a lot of information missing and even the example programs provided by Waveshare were filled with bugs and logical incoherence. However, during the many months taken to develop this project, improvements and progress were a constant. All aspects of the developed solution and laboratory guides were carefully examined and tested to make sure the results of this work are suited to be used in classrooms and evaluations at $IST$.

## 6.1 Achievements

The major achievements of this work have to be the in-depth analysis of the mobile robot, which was not available before, at this level, anywhere else, and also the development of the laboratory guides and the final Arduino coded solution.

The analysis itself must be considered an achievement because it puts, in one space, a complete guide to the functioning of each of the components that make up the Alphabot2-Ar. This means that, if any doubt arises regarding the functioning of each of the components of the robot, this work constitutes the first place where a possible answer for the doubt can be found. Of course, this work is only so big and cannot mention every aspect of every component, but it does mention the important aspects for the correct functioning of the mobile robot. This analysis also allows the Professors that use the laboratory guides and the Alphabot2-Ar in their classes, alongside the code solutions for each of the laboratory guides, to base their work preparations and course curriculum and theoretical contents on the information provided in the robot analysis, as well as in the laboratory guides.

The second achievement has to be the laboratory guides themselves. They present a unified guide to the development of programs that allow the students to interact with all the components present in the Alphabot2-Ar. Not only this but they also introduce the theoretical contents necessary in order to correctly and completely implement the Arduino functions that will control the components and, ultimately, the behaviour of the robot.

Alongside these laboratory guides, also the coded solutions themselves also constitute an achievement of this paper, specially the last coded solution, corresponding to the solution to the laboratory guide 3. That code consists of one single file that controls and manages all the peripherals, components, actuators and sensors of the Alphabot2-Ar. With this coded solution, which can be considered the framework of the mobile robot, any component of the robot can be interacted with. Not only this but this solution can also serve as base for further developments to be made to the robot and using the robot. By using this code solution, the person working on the robot can be assured that the behavior of the robot is well programmed and working as expected. This allows that person to focus only and exclusively on the task in hand, improving productivity and leading to better, more accurate and clear results.

## 6.2 Future Work

As the work developed in this paper proved to be pretty in depth when it comes to controlling and using all the components in the intended way, any future work that may be developed should focus on non intended ways to use the components. Firstly, any work that utilizes the XBee Bluetooth module could use the laboratory guide 3 coded solution as base and simply implement the communication with the module and the response to received commands.

However, one more interesting future work that could be developed has to do with the XBee board as well as the Arduino board. While the provided solution does allow for a pretty seamless operation of the robot, it does not allow the robot to vary its speed parameter while navigating around the track in a dynamic way. It can change its speed but the user must command it and, until changed again, it remains the same. This poses some issues when it comes to achieving the best possible lap times. The reason this wasn't implemented was because there simply was not enough memory or storage in the robot to store data from the sensors regarding each detail of the track, alongside the fact that the robot does not have a high enough processing power to create a model of the track based on the data to allow for comparisons of the data recorded by the sensors to data that matches each specific part of the track (alongside real-time control of all the components that make up the Alphabot2-Ar).

When mentioning data, the considered data is the speed of the robot and the information regarding $IR$ reflectivity obtained by each of the 5 $IR$ line detecting sensors. Without this data stored somewhere, it is impossible for the robot to know its position on the track and, therefore, know the ideal speed to be in.

If it were possible to implement an algorithm that allows the speed of the robot to dynamically change, increasing in the straights and reducing in the corners, the achievable lap times would be considerably better. In order to do this, the best possible imagined implementation would be to use a machine learning algorithm.

This algorithm would make the robot collect data regarding the track it is travelling on, and send that data onto a Bluetooth computer nearby, using the XBee Bluetooth board. This data would then be compiled into a full representation of the track. Having this full representation of the track and the data sent when the robot was going through the track (which include all the corners, straights and small details of the track), it is possible to create a machine learning algorithm that, when compiled into a neural network and the neural network is uploaded into the Arduino, would allow for the Arduino to know if it was currently on a straight or on a curve and the distance to the next curve and straight, by analyzing the data recorder by the sensors at any given time. With this information, the robot would be able to control its speed dynamically, knowing when to increase it and when to decrease it in order to have the fastest lap times.

One other approach would be to send the data in real time to a computer with high processing power but have the data regarding the next move of the robot sent to the robot, instead of having the robot making the calculation. By doing this, all that is necessary is to ensure that there is little latency in the communication between the robot and the computer and it is possible to have a seamless experience. This reduction in the computations that the robot must make helps the implementation of such a functionality, while allowing for the Arduino board to only deal with controlling the components of the mobile robot. The computer itself would receive the data sent by the robot, compile a map of the track, identify the location of the robot in the track and then send to the robot the appropriate changes in the robot's behavior so that the robot can go around the track with increased speed and accuracy.

The issue of the amount of speed that the robot should increase its parameter and also the automatic tuning of the KP, KI and KD parameters, as the speed increases, are another challenges to be tackled in this future work. To develop this work, one could could resort to projects such as [52–56], alongside technologies such as TensorFlow Lite for micro-controllers [57]. With these papers and platform, a possible future work surrounding this topic of machine learning algorithms to control the behaviour of the robot could be successfully implemented.

# Bibliography

[1] F. Rubio, F. Valero, and C. Llopis-Albert. A review of mobile robots: Concepts, methods, theoretical framework, and applications. *International Journal of Advanced Robotic Systems*, 2019.

[2] G. Demetriou. *Mobile Robotics in Education and Research*. 2011. ISBN 978-953-307-716-1. doi: 10.5772/26295.

[3] Waveshare alphabot2-ar wiki page. `https://www.waveshare.com/wiki/AlphaBot2`, . Accessed: 2021-05-14.

[4] B. Crnokić, M. Grubisic, and T. Volaric. Different applications of mobile robots in education. *International Journal on Integrating Technology in Education*, 6:15–28, 09 2017. doi: 10.5121/ijite.2017.6302.

[5] Aalto university intelligent robotics research areas page. `http://irobotics.aalto.fi/research`. Accessed: 2021-05-14.

[6] Ensta homepage. `http://cogrob.ensta-paris.fr/projects.html`. Accessed: 2021-05-14.

[7] Isr about page. `https://welcome.isr.tecnico.ulisboa.pt/about-us/overview/`, . Accessed: 2021-05-14.

[8] Isr storeslam project page. `https://welcome.isr.tecnico.ulisboa.pt/projects/accurate-agent-localization-and-mapping-methods-for-structured-indoor-retail-store-environments/`, . Accessed: 2021-05-14.

[9] Isr voamais project page. `https://welcome.isr.tecnico.ulisboa.pt/projects/computer-vision-for-the-operation-of-unmanned-aerial-vehicles-in-maritime-and-wildfire-scenarios/`, . Accessed: 2021-05-14.

[10] Isr cemra project page. `https://welcome.isr.tecnico.ulisboa.pt/projects/cemra-autonomous-mobile-robot-programming-using-ros-on-a-remote-web-based-real-robot-lab/`, . Accessed: 2021-05-14.

[11] Isr limoman project page. `https://welcome.isr.tecnico.ulisboa.pt/projects/developmental-learning-of-internal-models-for-robotic-manipulation-based-on-motor-primitives-and-multisensory-integration/`, . Accessed: 2021-05-14.

[12] Boston dynamics solutions page. https://www.bostondynamics.com/solutions. Accessed: 2021-05-14.

[13] Roomba home page. https://www.irobot.pt/roomba. Accessed: 2021-05-14.

[14] S.-E. Oltean. Mobile robot platform with arduino uno and raspberry pi for autonomous navigation. *Procedia Manufacturing*, 32:572–577, 2019. ISSN 2351-9789. doi: https://doi.org/10.1016/j.promfg.2019.02.254. URL https://www.sciencedirect.com/science/article/pii/S2351978919302896. 12th International Conference Interdisciplinarity in Engineering, INTER-ENG 2018, 4–5 October 2018, Tirgu Mures, Romania.

[15] R. Siegwart and I. R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Company, USA, 2004. ISBN 026219502X.

[16] Mobile industrial robots agv vs amr page. https://www.mobile-industrialrobots.com/en/insights/get-started-with-amrs/agv-vs-amr-whats-the-difference/. Accessed: 2021-05-14.

[17] Laboratory guide 0. https://drive.tecnico.ulisboa.pt/download/1695923671602091, . Applications and Computation for the Internet of Things, Master's Degree in Computer Science and Engineering, Instituto Superior Técnico (temporary web page). Accessed: 2021-06-07.

[18] Laboratory guide 1. https://drive.tecnico.ulisboa.pt/download/1695923671602085, . Applications and Computation for the Internet of Things, Master's Degree in Computer Science and Engineering, Instituto Superior Técnico (temporary web page). Accessed: 2021-06-07.

[19] Laboratory guide 2. https://drive.tecnico.ulisboa.pt/download/1695923671602096, . Applications and Computation for the Internet of Things, Master's Degree in Computer Science and Engineering, Instituto Superior Técnico (temporary web page). Accessed: 2021-06-07.

[20] Laboratory guide 3. https://drive.tecnico.ulisboa.pt/download/1695923671602102, . Applications and Computation for the Internet of Things, Master's Degree in Computer Science and Engineering, Instituto Superior Técnico (temporary web page). Accessed: 2021-06-07.

[21] Code solution to the laboratory guide 1. https://drive.tecnico.ulisboa.pt/download/1695923671602086, . Applications and Computation for the Internet of Things, Master's Degree in Computer Science and Engineering, Instituto Superior Técnico (temporary web page). Accessed: 2021-06-08.

[22] Code solution to the laboratory guide 2. https://drive.tecnico.ulisboa.pt/download/1695923671602097, . Applications and Computation for the Internet of Things, Master's Degree in Computer Science and Engineering, Instituto Superior Técnico (temporary web page). Accessed: 2021-06-08.

[23] Code solution to the laboratory guide 3. https://drive.tecnico.ulisboa.pt/download/1695923671602103, . Applications and Computation for the Internet of Things, Master's Degree in

Computer Science and Engineering, Instituto Superior Técnico (temporary web page). Accessed: 2021-06-08.

[24] Student base code file to the laboratory guide 1. `https://drive.tecnico.ulisboa.pt/download/1695923671602089`, . Applications and Computation for the Internet of Things, Master's Degree in Computer Science and Engineering, Instituto Superior Técnico (temporary web page). Accessed: 2021-06-08.

[25] Student base code file to the laboratory guide 2. `https://drive.tecnico.ulisboa.pt/download/1695923671602098`, . Applications and Computation for the Internet of Things, Master's Degree in Computer Science and Engineering, Instituto Superior Técnico (temporary web page). Accessed: 2021-06-08.

[26] Student base code file to the laboratory guide 3. `https://drive.tecnico.ulisboa.pt/download/1695923671602104`, . Applications and Computation for the Internet of Things, Master's Degree in Computer Science and Engineering, Instituto Superior Técnico (temporary web page). Accessed: 2021-06-08.

[27] J.-F. Lalonde, C. Bartley, and I. Nourbakhsh. Mobile robot programming in education. pages 345–350, 05 2006. doi: 10.1109/ROBOT.2006.1641735.

[28] Applications and computation for the internet of things, master's degree in computer science and engineering, instituto superior técnico - web page. `https://fenix.tecnico.ulisboa.pt/cursos/meic-a/disciplina-curricular/1971853845332813`. Accessed: 2021-05-14.

[29] F. Arvin, J. Espinosa, B. Bird, A. West, S. Watson, and B. Lennox. Mona: an affordable open-source mobile robot for education and research. *Journal of Intelligent & Robotic Systems*, 94: 761–775, 2019.

[30] J. Gonçalves and P. Costa. Differential mobile robot controller study: A low cost experiment based on a small arduino based prototype. In *2017 25th Mediterranean Conference on Control and Automation (MED)*, pages 945–950, 2017. doi: 10.1109/MED.2017.7984241.

[31] R. Balogh. Educational robotic platform based on arduino. In *Proceedings of the 1st international conference on Robotics in Education, RiE2010. FEI STU*, pages 119–122, 2010.

[32] M. Engin and D. Engin. Path planning of line follower robot. In *2012 5th European DSP Education and Research Conference (EDERC)*, pages 1–5, 2012. doi: 10.1109/EDERC.2012.6532213.

[33] V. Balaji, M. Balaji, M. Chandrasekaran, M. A. khan, and I. Elamvazuthi. Optimization of pid control for high speed line tracking robots. *Procedia Computer Science*, 76:147–154, 2015. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2015.12.329. URL `https://www.sciencedirect.com/science/article/pii/S1877050915038302`. 2015 IEEE International Symposium on Robotics and Intelligent Sensors (IEEE IRIS2015).

[34] Alphabot2-ar demo code. https://www.waveshare.com/wiki/File:AlphaBot2-Demo.7z. Accessed: 2021-05-14.

[35] Alphabot2-ar homepage. https://www.waveshare.com/product/arduino/robots/mobilerobots/alphabot2-ar.htm, . Accessed: 2021-05-14.

[36] Alphabot2-base schematic. https://www.waveshare.com/w/upload/9/91/AlphaBot2-Base-Schematic.pdf, . Accessed: 2021-05-14.

[37] Alphabot2-ar schematic. https://www.waveshare.com/w/upload/0/00/AlphaBot2-Ar-Schematic.pdf, . Accessed: 2021-05-14.

[38] Alphabot2-ar open eletronics review page. https://www.open-electronics.org/alphabot2-the-opensource-robot/, . Accessed: 2021-05-14.

[39] Arduino uno plus page. https://www.waveshare.com/uno-plus-package-a.htm. Accessed: 2021-05-14.

[40] Alphabot2 robot building kit for arduino. http://static6.arrow.com/aropdfconversion/f6dcd26fed52bb291be6e4dbcc75635d5011d6b4/110060864_web.pdf, . Accessed: 2021-05-14.

[41] Tlc1543 - 10-bit analog-to-digital converters with serial control and 11 analog inputs. https://www.ti.com/lit/ds/symlink/tlc1543.pdf. Accessed: 2021-05-14.

[42] Lithium-ion battery 14500 data sheet. http://ultran.ru/sites/default/files/catalog/svetodiody/brend/datasheets/lir14500.pdf. Accessed: 2021-05-14.

[43] Pcf8574 remote 8-bit i/o expander for i2c-bus. https://www.ti.com/lit/gpn/pcf8574, . Accessed: 2021-05-14.

[44] Arduino wire library. https://www.arduino.cc/en/reference/wire. Accessed: 2021-05-14.

[45] Pcf8574 remote 8-bit i/o expander for i2c-bus - nxp. https://www.nxp.com/docs/en/data-sheet/PCF8574_PCF8574A.pdf, . Accessed: 2021-05-14.

[46] Pcf8574 i2c digital i/o expander: Arduino, esp8266 and esp32, basic i/o and interrupt – part 1. https://www.mischianti.org/2019/01/02/pcf8574-i2c-digital-i-o-expander-fast-easy-usage/, . Accessed: 2021-05-14.

[47] Irremote library. https://www.arduino.cc/reference/en/libraries/irremote/. Accessed: 2021-05-14.

[48] Ssd1306ascii library. https://www.arduino.cc/reference/en/libraries/ssd1306ascii/. Accessed: 2021-05-14.

[49] Waveshare - tracker sensor. https://www.waveshare.com/wiki/Tracker_Sensor. Accessed: 2021-05-14.

[50] Adafruit neopixel library. https://www.arduino.cc/reference/en/libraries/adafruit-neopixel/. Accessed: 2021-05-14.

[51] Pulse width modulation. https://www.arduino.cc/en/Tutorial/Foundations/PWM. Accessed: 2021-05-14.

[52] Artificial intelligence (ai) based line following robot. https://create.arduino.cc/projecthub/Muhammad-sheraz/artificial-intelligence-ai-based-line-following-robot-b20235. Accessed: 2021-05-14.

[53] Line follower robot using cnn. https://towardsdatascience.com/line-follower-robot-using-cnn-4bb4f297c672. Accessed: 2021-05-14.

[54] You can run machine learning on arduino. and any other mcu out there too! https://eloquentarduino.github.io/2019/11/you-can-run-machine-learning-on-arduino/. Accessed: 2021-05-14.

[55] H. Hu. An educational arduino robot for visual deep learning experiments. In *2018 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 1310–1314, 2018. doi: 10.1109/FSKD.2018.8687137.

[56] F. M. Lopez-Rodriguez and F. Cuesta. An android and arduino based low-cost educational robot with applied intelligent control and machine learning. *Applied Sciences*, 11(1), 2021. ISSN 2076-3417. doi: 10.3390/app11010048. URL https://www.mdpi.com/2076-3417/11/1/48.

[57] Tensorflow lite for microcontrollers. https://www.tensorflow.org/lite/microcontrollers. Accessed: 2021-05-14.

[58] Structured folder containing all developed materials. https://drive.tecnico.ulisboa.pt/download/1695927966745303. Applications and Computation for the Internet of Things, Master's Degree in Computer Science and Engineering, Instituto Superior Técnico (temporary web page). Accessed: 2021-06-08.

# Appendix A

# Developed Materials

Alongside the development of this document, some other materials were also produced. These materials include:

- 4 laboratory guides:
  - Laboratory guide 0 [17]
  - Laboratory guide 1 [18]
  - Laboratory guide 2 [19]
  - Laboratory guide 3 [20]
- 3 laboratory code solutions:
  - Code solution for the laboratory guide 1 [21]
  - Code solution for the laboratory guide 2 [22]
  - Code solution for the laboratory guide 3 [23]
- 3 base code files to be delivered to the students for the development of their laboratory work:
  - Base code file for laboratory guide 1 [24]
  - Base code file for laboratory guide 2 [25]
  - Base code file for laboratory guide 3 [26]

A complete folder that depicts these files in a structured way, alongside tested values for the parameters SPEED, KP, KI and KD, for each of the laboratory code solutions, can be found in [58].