Car Race

Hussein Giva
Instituto Superior Técnico, Universidade Lisboa
Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal
hussein.giva@tecnico.ulisboa.pt

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. They benefit from a generalist and didactic robot that enhances their learning experience. The first work 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. The final objective of this work is the creation of laboratory guides that allow students to create programs that fully and correctly control the robot.
First, an analysis of the robot is made and the contents that make up the developed laboratory guides 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. The results prove the efficacy of the developed code in line following, detection and avoidance of obstacles and in providing a robust and complete interaction with the robot's peripherals.

Author Keywords

Mobile robot; Alphabot2-Ar; Arduino; Line following; Obstacle detection

INTRODUCTION

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. 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 basis to operate the components of the robot correctly and the practical part to show the students the implementation and results of their work.
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.

May 2021, Lisbon, Portugal.

BACKGROUND

For the mobile robot being used to be adequate for an educational use, it must not deviate too much from the norm that guides other universities in similar courses while also being fit and ideal to use in the courses that plan on utilizing it within Instituto Superior Técnico (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. Understanding the technical and theoretical concepts is also important, as to see if they match the ones taught in IST, in these courses.
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 two tasks and understand whether those ways are the ideal ways to show and teach the students.

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 paper, the sensors used are infrared obstacle detection, infrared line detection and ultrasonic obstacle detection - and filtering methods - in the case of this paper, data normalization and weighted average) and action (reactive control - in the case of this paper, using the Proportional, Integral and Derivative (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 has sonar and infrared sensors is on par with the robot used here. The paper presented below supports this researched thesis.

An analysis of the Mobile Robot Programming Laboratory course at Carnegie Mellon University (CMU), done by Lalonde et. al [19], shows a multitude of similarities to the Applications and Computation for the Internet of Things ($ACIC$) [5] course in IST. $ACIC$, as the Mobile Robot Programming Laboratory course, also consists of a one-semester course that gives students hands-on experience with mobile robot programming. It is also a problem driven course, where the practical component is used to develop a solution to problems presented on laboratory guides. This solution is then improved upon in the next laboratory guide, where new problems are presented and must be solved using the previous solution as base. Regarding the robotic platform, the Alphabot2-Ar does not have a camera 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)" [19], 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 it is drawn. Since the Alphabot2-Ar also uses ultrasonic sensors, 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. The use of the sonars is similar in both robots, being used to measure distances and detecting obstacles. In terms of the actuators, both robots present two motorized wheels. The main difference comes with the processing unit of the robots, where the Alphabot2-Ar uses Arduino while the Carnegie Mellon one uses a laptop running Windows XP. As for the course curriculum and theoretical/practical content, 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 Analog-Digital Converter (ADC), Inter-Integrated Circuit ($I2C$) with serial interface, data normalization and weighted averages. 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 developed laboratory guides 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 expects the mobile robot to go around a track, delimited by a line that the robot must follow.

Theoretical Model 2

The second theoretical model compiled states that, for a consistent 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. 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 Infrared (IR) 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 readings. 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:

$$normVal = \frac{(val(sensor) - minCalVal(sensor)) * scaleMaxVal}{maxCalVal(sensor) - minCalVal(sensor)} \quad (1)$$

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:

$$weightedAverage = \frac{\sum_{j=0}^{4}(finalValue(j) * (j * 100 - 200))}{\sum_{j=0}^{4} finalValue(j)} \quad (2)$$

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 having that numeric value of the position and then applying the PID algorithm to determine the speed value necessary for each wheel, in order to center the robot, it is possible to attain a smooth and accurate operation for the robot. The classical PID equation, that is also used for the implementation of the PID algorithm in this work, is the following:

$$pidVal = proportional*KP + integral*KI + derivative*KD \quad (3)$$

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 paper presented below supports this researched thesis.

Engin et al. [18] 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" [18]. 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 two motorized wheels, the Printed Circuit Board (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 et al. [18]. The difference resides with the fact that a quadratic line detection algorithm was used, as opposed to a weighted average [18]. 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. The similarities between the weighted average procedure and this quadratic approach mean both can be considered similar ways to extract the line's position in regards to the sensors. 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. The PID algorithm implemented works 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 is used to reduce the oscillation of the robot and make it behave without more curves than the ones needed to 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.

ROBOT ANALYSIS

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. The top chassis board is called the Alphabot2-Ar and the Arduino board attaches to this PCB. The robot comes with some pre-assembled parts but some assembly is still required. Laboratory guide 0 provides detailed instructions on how to assemble the remaining parts. In [4], it is possible to find the schematics regarding the assembly of all the components of the Alphabot2-Base and, at [3], it is present the schematics of the assembly of all the components that make up the Alphabot2-Ar.

Arduino UNO Plus

The Alphabot2-Ar uses an Arduino UNO Plus as the processing board. This board integrates all the Input/Output (I/O) and processes the data sent by the sensors and sends commands to the actuators [6].

The processor used by the Arduino UNO Plus is the ATMEGA328P-AU. It has 32k bytes of Flash memory, 1k byte Electrically-Erasable Programmable Read-Only Memory ($EEPROM$) and 2k bytes Static Random-Access Memory ($SRAM$). This board also has a built-in ADC with 8 channels as well as support for serial communication protocols such as $I2C$ and Serial Peripheral Interface (SPI), used for communication with external modules. In order to develop code to operate the Arduino board and, consequently, the Alphabot2-Ar, the Arduino Integrated Development Environment (IDE) was used, alongside the Arduino Language Reference.

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 [14]. 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 ..." [14].

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. 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 [14]. DATA OUT pin outputs the 10-bit result of the previous conversion from the ADC to the Arduino board [14]. ADDRESS pin is the pin from the ADC that receives the 4-bit address sent by the Arduino board. Both ADDRESS and DATA OUT operate with Most Significant Bit (MSB) first [14]. 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 [14]), the ADC operates with each conversion cycle (receive address and output previous conversion plus Analog-Digital conversion operation cycles) taking 16 operation cycles [14].

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) [9]. As the battery is drained, the voltage of the batteries decreases till what's called the Safe Discharge Voltage – minimum battery voltage to ensure the battery works properly [9]. 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 [9]. 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 two 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 [3]. According to the formula for voltage dividers:

$$V_{out} = \frac{V_s * R_2}{R_1 + R_2} \tag{4}$$

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 two 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:

$$V_{battery} = \frac{ADC_{value} * ADC_{referenceV}}{1023} * 0.5 \tag{5}$$

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

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 two infrared obstacle detecting sensors, the joystick and the buzzer. It is communicates with the Arduino using $I2C$ communication protocol. 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." [10]. 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 two infrared sensors and the joystick only send data back to the Arduino. To control this communication, the Wire.h library is used [7]. 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. This is done using the table in [11]. 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. To do this, it is necessary to look to [3] and to [10]. To assemble the address of a device, one needs simply to put the bit corresponding to that device equaling 0 and the remainder equaling 1, according to Table 1.

BYTE	BIT							
	7 (MSB)	6	5	4	3	2	1	0 (LSB)
I^2C 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 1. PCF8574 Interface Definition. [10]

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. Each button of the remote control is associated with a given address code. To receive information from the IR receiver, the library IRremote.h should be used

[8]. To identify the address for each of the buttons, it is 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 [8]. To unlock the receiver and allow it to receive new commands, execute IrReceiver.resume() [8]. To test the component and to learn how to interact with it in an Arduino environment, the example program IR.ino was used [16, 2].

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 must be used [7]. The buzzer's 2 possible commands are ON and OFF. In order to set the buzzer working, first the Wire.h library must be configured [7]. To do so, initiate the program with Wire.begin() [7]. 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) [7]. Then, to send the command, the address of the buzzer turn on and turn off commands must be sent to the I/O expander, via the Wire.write(message), with message being the address (0xDF to turn on and 0xFF to turn off) [7]. After finishing the transmission, call Wire.endTransmission() to terminate it [7].

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 [7]. 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 [7]. This data corresponds to which direction the joystick was pressed at the time of the call of the requestFrom function. The returned data has a similar address structure to the data sent with the send mechanism of the Wire library [7]. For the 8 bits of the response, each bit corresponds to one I/O pin (P0 - P7). The bit, in the returned data, that is set to LOW (= 0) corresponds to the direction pressed in the joystick at the time of the call. To test the component and to learn how to interact with it in an Arduino environment, the example program Joystick.ino was used [16, 2].

OLED Screen

In the front of the Alphabot2-Ar PCB is a small, 0.96 inch Organic Light Emitting Diode ($OLED$) two-color display, with a resolution of 128x64. To manage the contents of the screen, a specific external library should be used, that manages the communication with the display drivers. The choice made was the SSD1306Ascii library [13]. 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 [13]. To use, first the address code for the display must be defined. In [3], the address is defined as 0x3C. After having defined the screen address, all that's left to do is to create a variable of type SSD1306AsciiAvrI2c [13]. Then, to initialize the variable, in the setup() function of the Arduino, call the oled.begin(&Adafruit128x64, OLEDI2C, OLEDRESET) function [13]. 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 [13]. To test the component and to learn how to interact with it in an Arduino environment, the example program oled.ino was used [16, 2].

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 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 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:

$$distance = \frac{duration * 0.034}{2} \tag{6}$$

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 2 is used to divide the final distance value by two, 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 two. 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 TRIGGER 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. 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 [16, 2].

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 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. 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. To access these sensors, it is first necessary to know the addresses for the sensors. After having the addresses defined, done in a similar way to the joystick and buzzer, 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. Using the Wire.read() functions returns the data sent back by the expander [7]. 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. 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 [16, 2].

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. 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 robot, over a given background. 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. 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. 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. 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 [3]. According to [14], 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], is to access the ADC for each sensor, by sending the sensor's address to the ADC. 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 the line. The attained 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 [15], 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 [16, 2].

WS2812B RGB LEDs

One other component attached to the Alphabot2-Base PCB are the 4x WS2812B Red, Green and Blue (RGB) Light Emitting Diode (LED)s. These LEDs are connected to the Arduino board via pin 7 [3]. Similar to the $OLED$ display and the IR receiver, the LEDs must also be managed with the help of an external library. The chosen library is the Adafruit_NeoPixel.h [1]. 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) [1]. This function takes as arguments the number of LEDs, the pin to which they are connected and the type of LEDs. After having initialized the variable, all that's necessary is to call the begin() function on the variable [1]. Then, to set a given colour, simply call the function setPixelColor(i, red, green, blue) on the created variable [1]. 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 [1]. To test the component and to learn how to interact with it in an Arduino environment, the example program W2812.ino was used [16, 2].

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 [17], 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." [17]. 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:

$$perimeter = \pi * diameter \tag{7}$$

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, without the robot and with the same tires, in m/s, is 1.32m/s. The motors connect to the Arduino board via the pins A0, A1, A2 and A3, alongside pins 5 and 6 [3]. The pins A0 and A1 connect the forward and backward circuit, respectively, for the left motor [3]. The pins A2 and A3 connect the backward and forward circuit, respectively, for the right motor [3]. Pins 5 and 6 are used for the Pulse Width Modulation (PWM) [12] speed pin, for the right and left motors respectively [3]. 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. To test the component and to learn how to interact with it in an Arduino environment, the example program Run_Test.ino was used [16, 2].

IMPLEMENTATION

Laboratory Guide 0

The first laboratory guide 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 features a diagram of the components present in both the Alphabot2-Base and Alphabot2-Ar PCBs. It also features a diagram of the components of the Arduino UNO Plus board. After introducing all the components, assembly instructions with pictures are given. After the instructions, pictures of the assembled robot are shown. In the end of the guide, there are some safety recommendations for the students to mind when dealing with this robot and when assembling and removing components. Also included in the guide are some references used when the compilation of the guide was made.

Laboratory Guide 1

The laboratory guide 1 is the first full-length laboratory guide that the students will have to complete. In this lab, 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 is provided, upon which the work should be developed. The theoretical introduction approaches how the ADC works and how to access it (Since the 5 IR line detecting sensors are connected to the ADC), data normalization (where they are introduced to the sensor calibration), weighted average and the PID algorithm (which allows students 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). The practical work revolves around implementing incomplete or missing functions from the provided base code file (calibrate(), readLine() and pidAlgorithm()). It is also asked to the students to fine tune the PID algorithm constant parameters. Afterwards, two theoretical questions are asked, one regarding the main loop function and the process behind the calculation for the final speed value for each of the motors and the other one regarding the readADC(address) function, 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.

Laboratory Guide 2

The second complete laboratory guide that the students are presented with is the laboratory guide 2. 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 threshold distance. The two 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. The theoretical introduction approaches $I2C$ communications (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), how the I/O expander works and how distance calculation is made (using the digitalWrite and pulseIn functions). The practical work revolves around implementing incomplete or missing functions from the provided base code file (obstacleDetectedInfraRed(), distanceMeasureUltraSound() and loop()). Afterwards, some theoretical questions are asked. The first regarding the minimum speed value for the robot to move and why is that value different than 0. The second one regards 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 third one asks if it is possible, with the robot assembled as is, to measure distances using the infrared obstacle detecting sen-

sors and to explain the reasoning. The next two 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 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, for different presented speeds.

Laboratory Guide 3

The third and final laboratory guide is the laboratory guide 3. This laboratory guide develops, once again, on the solution achieved on the previous laboratory work. The goal of this laboratory work is to have the mobile robot interact with its peripherals. The list of peripherals includes the $OLED$ display, the buzzer, the joystick, the LEDs 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 two 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 LEDs and the buzzer. 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 LEDs and the display of the battery status. Finally, the infrared remote control will serve for a number of purposes: change the color of the LEDs (between red, green and blue); enable and disable the LEDs; 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 approaches the use of external libraries (Wire.h, SSD1306Ascii, Adafruit_NeoPixel.h and IRremote.h), battery calculation procedures (both the voltage of the batteries as well as the voltage flowing through the Arduino) and the delay() and millis() Arduino functions (used in some functions of the code solution). The practical work revolves around implementing incomplete or missing functions from the provided base code file (auto-calibration and receive commands via the joystick and the infrared remote control). Afterwards, some theoretical questions are asked. The first regarding the content of the calculateArduinoVoltage() function and the logic behind it. The second one 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 [12]. The third one asks the students to compete for the prize of best programmed and configured robot. To do this, they should race their solutions. They may use the remote control to alter the behavior of the robot and use whatever speed, distance, kp, ki and kd values the team believes is best.

RESULTS

Since the laboratory guide 3 solution may be considered as a complete platform program 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 two 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 two 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.

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 are changed as well). The measurements are made on both tracks, as to understand the solution's 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). 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 PID top speed value is reached.

Data Gathering

To gather data regarding the evaluation metrics, two designed racetracks were used, each on a 2m x 2m white paper sheet.

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

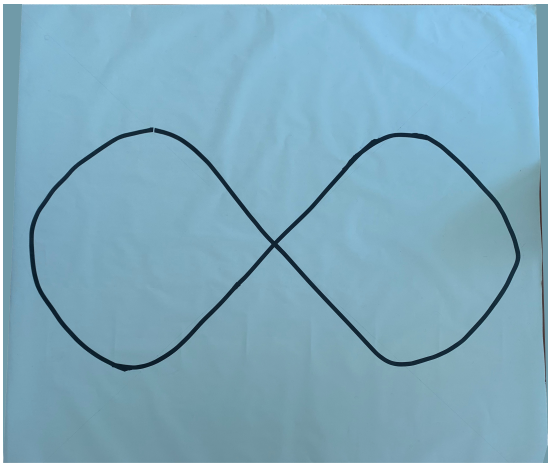

Figure 1. Racetrack 1.

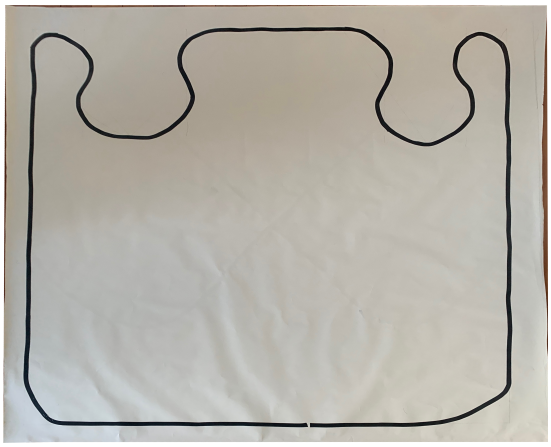

Figure 2. Racetrack 2.

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 two 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 2.

Testing Results

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

Table 2. Alphabot2-Ar speed configurations used in this work.

The following tables 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 detected in any of the tested speeds.

Speed	Avg. Lap Time	Crashes Into Box	Derrailings
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

Table 3. Measurement results for the racetrack 1, "8".

Speed	Avg. Lap Time	Crashes Into Box	Derrailings
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

Table 4. Measurement results for the racetrack 2, "Professional".

The results presented in Tables 3 and 4 contain values measured over the course of 10 laps, with the derrailing parameters being halved, since it was 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 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.

CONCLUSION

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 open-source and pedagogic nature of the components, specially the Arduino board, make this robot the perfect choice for the desired intent. 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 they followed the standard norm when it came to educational activities to make students interact with mobile robots. 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. This solution 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, while interacting with all the components of the robot.

REFERENCES

1. Adafruit neopixel library. **https://www.arduino.cc/reference/en/libraries/adafruit-neopixel/**. Accessed: 2021-05-14.
2. Alphabot2-ar demo code. **https://www.waveshare.com/wiki/File:AlphaBot2-Demo.7z**. Accessed: 2021-05-14.
3. Alphabot2-ar schematic. **https://www.waveshare.com/w/upload/0/00/AlphaBot2-Ar-Schematic.pdf**. Accessed: 2021-05-14.
4. Alphabot2-base schematic. **https://www.waveshare.com/w/upload/9/91/AlphaBot2-Base-Schematic.pdf**. Accessed: 2021-05-14.
5. Applications and computation for the internet of things - ist page. **https://fenix.tecnico.ulisboa.pt/cursos/meic-a/disciplina-curricular/1971853845332813**. Accessed: 2021-05-14.
6. Arduino uno plus page. **https://www.waveshare.com/uno-plus-package-a.htm**. Accessed: 2021-05-14.
7. Arduino wire library. **https://www.arduino.cc/en/reference/wire**. Accessed: 2021-05-14.
8. Irremote library. **https://www.arduino.cc/reference/en/libraries/irremote/**. Accessed: 2021-05-14.
9. Lithium-ion battery 14500 data sheet. **http://ultran.ru/sites/default/files/catalog/svetodiody/brend/datasheets/lir14500.pdf**. Accessed: 2021-05-14.
10. Pcf8574 remote 8-bit i/o expander for i2c-bus. **https://www.ti.com/lit/gpn/pcf8574**. Accessed: 2021-05-14.
11. 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.
12. Pulse width modulation. **https://www.arduino.cc/en/Tutorial/Foundations/PWM**. Accessed: 2021-05-14.
13. Ssd1306ascii library. **https://www.arduino.cc/reference/en/libraries/ssd1306ascii/**. Accessed: 2021-05-14.
14. 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.
15. Waveshare - tracker sensor. **https://www.waveshare.com/wiki/Tracker_Sensor**. Accessed: 2021-05-14.
16. Waveshare alphabot2-ar wiki page. **https://www.waveshare.com/wiki/AlphaBot2**. Accessed: 2021-05-14.
17. Arvin, F., Espinosa, J., Bird, B., West, A., Watson, S., and Lennox, B. Mona: an affordable open-source mobile robot for education and research. *Journal of Intelligent & Robotic Systems 94* (2019), 761–775.
18. Engin, M., and Engin, D. Path planning of line follower robot. In *2012 5th European DSP Education and Research Conference (EDERC)* (2012), 1–5.
19. Lalonde, J.-F., Bartley, C., and Nourbakhsh, I. Mobile robot programming in education. (05 2006). 345–350.