

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Teaching ROS2 with a Minimalistic Mobile Robot

Alexandre Cruz Miranda

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Armando Jorge Miranda de Sousa

October 10, 2022

© Alexandre Miranda, 2022

Gonçalo da Mota Laranjeira Torres Leão collaborated with the supervising team during this work.

Resumo

Atualmente existe uma vasta quantidade de ferramentas e de informação sobre robótica e Inteligência Artificial ([AI](#)). Contudo, a maioria possui uma dificuldade de aprendizagem relativamente elevada, o que não as torna tarefa fácil para principiantes e difíceis de dominar na sua totalidade. Esta barreira inicial infelizmente pode causar a perda de interesse por estes temas para os iniciantes.

As instituições universitárias, especialmente as de Engenharia, têm um papel importante no ensino destas disciplinas, uma vez que os seus alunos provavelmente terão um contacto ainda maior com sistemas robóticos inteligentes no seus futuros empregos. Tendo isto em mente, é importante que estas instituições adaptem os seus cursos de forma a proporcionar uma formação simples e eficaz em robótica e [AI](#). Os recursos disponíveis, tais como o acesso a robôs físicos e a forma como os cursos são ensinados terão um impacto directo sobre a eficácia das bases essenciais de robótica e robótica inteligente. No entanto, o número de estudantes que uma instituição possui exige uma quantidade significativa de robôs, e o custo total pode ser demasiado elevado para estas o conseguirem suportar.

Esta dissertação tinha o propósito de fazer um pequeno estudo sobre robótica e sobre "Robot Operating System" ([ROS](#)), com o objectivo de compreender a importância e quais os tópicos mais valiosos destes temas. Tinha também o objectivo de desenvolver um pequeno robô móvel de baixo custo para ser utilizado no ensino de robótica baseada em [ROS](#). Finalmente, foi criado um pequeno tutorial para ensinar como começar a utilizar o robô desenvolvido, e também para ensinar a executar algumas aplicações de demonstração desenvolvidas, que tinham a finalidade de validar o conceito. Ao mesmo tempo, este guia ajuda a solidificar alguns conceitos e comandos básicos de [ROS2](#).

Palavras-chave: ROS, Robótica, Educação, Sistemas de Visão

Abstract

Nowadays there is a vast amount of tools and information about robotics and Artificial intelligence ([AI](#)). However, most of them have a relatively high learning difficulty curve, being no easy task for beginners and difficult to fully master. This initial barrier unfortunately can cause the loss of interest in these subjects for newcomers.

University institutions, especially those in Engineering, have a major role in teaching these subjects since their students will probably have an even greater contact with intelligent robotic systems in their future jobs. Having this in mind, it is important that these institutions adapt their courses to provide a simple and effective formation in robotics and [AI](#). The available resources, such as physical robots, and the way the courses are taught will have a direct impact on how effective the essential bases on robotics and intelligent robotics are settled. However, the number of students an institution has demands a significant amount of robots, and the total cost might be too high for them to afford.

This dissertation had the purpose to make a small study regarding robotics and Robot Operating System ([ROS](#)) teaching, with the objective to comprehend the importance and what are the most valuable topics of these subjects. It also had the objective to develop a small low-cost mobile robot to be used to teach robotics based on [ROS](#). Finally, a small tutorial to teach how to start using the developed robot was created, as well as how to run some demo applications developed as a concept validation and practical examples. At the same time, this guide helps to solidify some [ROS2](#) basics concepts and commands.

Keywords: ROS, Robotics, Education, Vision Systems

Agradecimentos

Antes de mais, quero agradecer ao meu orientador e co-orientador, Armando Jorge Miranda de Sousa e Gonçalo da Mota Laranjeira Torres Leão por todo o apoio, transmissão de conhecimentos e disponibilidade que prestaram ao longo desta Dissertação.

Devo também um agradecimento ao Professor Paulo Costa pelo fornecimento do robo, e ao meu colega Vítor Ventuzelos por ter fornecido algum material e apoio que me ajudou bastante para os desenvolvimentos desta Dissertação.

Quero também agradecer à minha família, principalmente ao meu pai, por toda a ajuda e suporte dado com problemas que foram surgindo no decorrer da Dissertação.

Por fim, e não menos importante, quero deixar o agradecimento a todos os meus amigos não só pelo apoio e motivação, mas também pela paciência devido às minhas frustações em momentos mais difíceis.

Alexandre Miranda

*“If you set your goals ridiculously high and it’s a failure,
you will fail above everyone else’s success.”*

James Cameron

Contents

1	Introduction	1
1.1	Context	1
1.2	Objectives	2
1.3	Document Structure	2
2	Robot Operating System (ROS) and robotics teaching platforms	3
2.1	Robot Operating System (ROS)	3
2.1.1	ROS1 teaching material	4
2.1.2	ROS2 teaching material	5
2.2	Examples of robots suited for AI Robotics	5
2.2.1	BulbRobot	6
2.2.2	Mona	7
2.2.3	Khepera IV	7
2.2.4	Thymio	8
2.2.5	E-puck and e-puck2	9
2.2.6	Robobo	11
2.2.7	Robots Comparison and conclusions	12
2.3	Summary	14
3	Questionnaire Results	15
3.1	Classification of experience in ROS use and teaching	15
3.2	Robotics teaching course and academic level	16
3.3	Is teaching robotics with ROS important? Why?	16
3.4	Is teaching ROS2 relevant? Why?	17
3.5	Do you consider it important to work with real robots when teaching robotics? When teaching ROS? Why for robotics and why for ROS?	17
3.6	Between C++ and Python, which one do you consider more important to teach ROS with? Why?	18
3.7	Do you consider any topic(s) of the book irrelevant (or less relevant) to a first introduction to ROS? Which one(s)?	18
3.8	Classify how important you think the following topics of ROS are for teaching of ROS at an introductory level.	19
3.9	Besides the topics of the previous question, which topics do you consider the book is lacking?	20
3.10	Do you have any suggestions about books, guides or videos for ROS?	21
3.11	Summary	21

4 The Physical Robot	23
4.1 The Robot and its Hardware	23
4.1.1 List of Material	24
4.1.2 Hardware Architecture	26
4.2 Hardware and Design Modifications	28
4.2.1 Camera support	28
4.2.2 The Raspberry Pi upgrade	31
4.2.3 The Raspberry Pi camera	32
4.3 Summary	32
5 The Software	33
5.1 Requirements	33
5.2 ROS2 and Applications Development	34
5.2.1 ROS code Migration	34
5.2.2 ROS2 packages explanation	35
5.2.3 Applications created	36
5.3 Arduino Code	37
5.4 Summary	38
6 Results and Discussions	39
6.1 Physical robot discussion	39
6.2 Software and applications development	40
6.2.1 User controlled robot demo	40
6.2.2 Distance measure demo	40
6.2.3 Reactive robot demo	42
6.2.4 LiDAR	42
6.3 Summary	43
7 Conclusions and Future Work	45
7.1 Main contributions	45
7.2 Future work	45
7.2.1 Physical platform	45
7.2.2 Software and program development	46
7.2.3 Simulator	46
A Questionnaire Form	47
B Software Installation and Environment Configuration	57
B.1 Operating system installation	57
B.2 ROS1 Noetic Ninjemys Installation	58
B.3 ROS2 Foxy Fitzroy installation	58
B.4 Arduino IDE installation	59
B.5 Visual Studio Code installation	59
B.6 Define fixed tty devices	60
B.6.1 LiDAR device identifier	60
B.6.2 Arduino device Identifier	61

C ROS Migration Guide	63
C.1 Migration of Custom Messages and Services	63
C.2 Migration of Package.xml files	64
C.3 Migration of CMakeLists.txt files	64
C.4 Migration of source files	65
C.5 Migration of launch files	66
D Physical Robot Introduction using ROS2 (GitHub ReadMe)	67
References	77

List of Figures

2.1	The BulbRobot (3D software model) [1]	6
2.2	The Mona robot ¹⁰	7
2.3	The Khepera IV ¹¹	8
2.4	The Thymio robot ¹⁴	9
2.5	The e-puck robot with the basic extension board [2]	9
2.6	Examples of e-puck extension boards: an infrared distance scanner(left) and turret with three linear cameras (right) [2]	10
2.7	The e-puck2 ¹⁸	11
2.8	The Robobo robot [3]	12
3.1	Graphs of the responses of personal classification on experience in using and teaching ROS1 and 2, where 1 in the scale is no experience and 5 very experienced.	16
3.2	Graphs of the responses classifying the importance to learn some topics in ROS	20
4.1	Physical robot in its original configuration	23
4.2	Final robot with all modifications applied	24
4.3	Schematic of the Hardware Layout	27
4.4	Hardware layout of the physical robot	27
4.5	3D model of the robot in its original configuration	28
4.6	3D model of the robot with different versions of a frontal camera support in red - (a); black - (b); and dark blue - (c)	29
4.7	3D model of the robot with the final version of the support	30
4.8	Final camera support, in a virtual 3D model - (a); Split in half version for 3D printing - (b); 3D printed version, with Raspberry and camera installed - (c)	30
4.9	New location of the LiDAR board	31
5.1	Software and Hardware structure	33
6.1	Messages received from the Arduino and published by the RPi-Uino communication package	41
6.2	Distance to the hand measured by the robot when it is at the same distance but with the hand in lower position - (a); and upper position - (b)	41
6.3	Distance to the hand measured by the robot when the hand is close - (a); and when it is far - (b)	42
6.4	Terminal printed logs of the LiDAR program	43
B.1	Parent device with the vendor and product ID and serial number for the LiDAR	61
B.2	Parent device with the vendor and product ID and serial number for the Arduino	61

B.3 New names assigned for the devices, written in blue 62

List of Tables

2.1	Mobile robots and their main features	13
3.1	Average of the classifications gathered for each topic	19
4.1	List of Material of the robot	25
5.1	Channels and their respective function	38

Abbreviations and Symbols

AI	Artificial Intelligence
API	Application Programming Interface
BLE	Bluetooth Low Energy
CAD	Computer-aided Design
CMOS	Complementary Metal-Oxide Semiconductor
DC	Direct Current
EOL	End-of-Life
GDB	GNU Debugger
GPS	Global Positioning System
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
IO	Input/Output
iOS	iPhone Operating System
IR	Infrared
LED	Light-Emitting Diode
LiDAR	Light Detection And Ranging
OS	Operating System
PCL	Point Cloud Library
PLA	Polylactic Acid
RAM	Random Access Memory
RF	Radio Frequency
RGB	Red, Green, Blue
RPi	Raspberry Pi
ROS	Robot Operating System
SD	Secure Digital
SDK	Software Development Kit
SPI	Serial Peripheral Interface
TF	Transform
ToF	Time of Flight
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VNC	Virtual Network Computing
VPL	Visual Programming Language
VSCode	Visual Studio Code
Wi-Fi	Wireless Fidelity
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 Context

The use of robots and intelligent machines is growing exponentially, not only in the number of possible applications, but in complexity as well. These technologies are proving to be effective in solving some problems and in improving solutions in the industry such as manufacturing processes and transportation.

According to K. Afasari and M. Saadeh [4], the increase in Artificial Intelligence ([AI](#)) interest and development will have a significant impact in the future, leading to a substantial rise in the creation of new jobs related to [AI](#) in the near future. To respond to this demand, the education system needs to adapt in order to form engineers capable of working with [AI](#).

The ever-growing appeal in robotics and [AI](#) robotics led to an appearance of a vast set of tools and an endless quantity of information built by the constantly bigger community. One of the biggest tools used in robotics is Robot Operating System ([ROS](#)) for robot programming. These tools give access to a vast number of functionalities and have information available to consult. However, sometimes this excess of information can make the learning process more difficult if not taken the right progressive steps. Some of these tools are in fact, no easy task to start and difficult to fully master.

That is why institutions must provide an efficient and effective way to teach how to use these tools, being able to provide the support and the resources needed, and making the learning of these subjects easier. Having a way for students to have better bases in robotics and [AI](#) robotics will not only provide a better capability for development and comprehension in more complex problems, since that is the tendency of evolution in robotics, but also increase the interest in exploring more of what these tools can create.

One important resource to complement the teaching process in robotics is the use of physical robots. By using physical robots, students have a closer experience on the interaction between software/hardware and machine limitations, like memory and time constraints in the decision making [5], giving the students a more approximated experience from a real world scenario, something

that would be more difficult to achieve if they are limited to just using simulations. It also creates an incentive to learn because students want to see what they create working properly [5].

1.2 Objectives

This dissertation had the objective to create a minimalist, low-budget mobile robot by adapting an already existing mobile platform, meeting the requirements to be a capable machine for robotics teaching using [ROS1](#) and [ROS2](#) in institutions like Faculty of Engineering of the University of Porto (FEUP). To achieve this minimalist system, a camera-based robot was chosen. Vision systems are versatile because it is possible to do sensor virtualization by using image processing algorithms. Using virtual sensors allowed to reduce the number of sensors required for an autonomous robot, simplifying the electrical and mechanical complexity.

It also focused on the study and evaluation of robotics and [ROS](#) teaching. This study had the purpose to understand how important the teaching of these subjects is and what topics and sections about [ROS](#) are the most important to make the teaching more effective and easier to comprehend.

Finally, it was intended to create a small robot introduction guide to teach how to set up and configure the environment to be ready to develop applications on the created physical system. Some application demos were created and introduced in the tutorial as well, not only to validate the robot's concept but also to better comprehend the interaction between [ROS](#) and the physical robot when doing the tutorial.

1.3 Document Structure

Chapter 1 introduces the work proposal, clarifying the context and motivation of the dissertation along with its objectives. Chapter 2 explores the background knowledge required in [ROS](#), and depicts some robots developed for robotics. In Chapter 3, a detailed analysis of the results gathered by the robotics and [ROS](#) questionnaire is discussed. In Chapter 4, an analysis of the used robot, with an explanation of the modifications, hardware architecture, and material list, is provided. Chapter 5, contains the software installation and environment configuration. It also details the code and applications done for the robot. The Chapter 6 is for results and discussions. Here, discussions about the physical robot, the programs developed, and the tutorial created for it, are made. Finally, Chapter 7 summarizes the biggest contributions of the work and presents some suggestions for possible future work to be made to improve the robot.

Chapter 2

Robot Operating System (**ROS**) and robotics teaching platforms

This chapter has the purpose to explore some key concepts of the tools used, in particular, **ROS**. It is also listed some existing robots built for research or learning purposes and a comparison between all these systems is made at the end.

2.1 Robot Operating System (**ROS**)

As previously mentioned, **ROS**¹, also known as Robot Operating System, is a powerful framework for robotics development. It is an open source software that defines tools, interfaces and components to construct advanced robots. It helps developers connect the actuators, sensors and the control system using topics and messages.

In **ROS** [6], processes that performs computation are called nodes, and a system can contain multiple nodes. Nodes communicate with each other by publishing messages in a topic. Topics are buses to exchange messages, and it follows a publisher-subscriber pattern. Messages are the information transmitted. Messages can be a standard data type (integer, boolean, etc.) or a composition of other messages.

Briefly, in the publisher-subscriber pattern exists the ones that sends the information, called publishers, and the ones that receives the messages, called subscribers. A publisher does not send the information directly to a subscriber, but makes it available to everyone. The subscribers will subscribe only to the publishers that their sent messages have interest to the subscriber itself.

In **ROS**, a node can publish and subscribe to multiple topics, and it is possible to have multiple subscribers and publishers for the same topic.

ROS also supports synchronous transactions, in a topology of client-server, called services. Client-server means that a client node requests the specific information to another node, the server, and the server then responds.

As for the programming languages, the more common ones are C++ and Python.

¹<https://www.ros.org/>

[ROS](#) organizes the software system into packages². Packages are directories containing nodes, independent libraries, configuration files or any other file or folder that constitutes a useful module. This organization structure provides a piece of software that can be reused easily. It also helps to partition the [ROS](#) software in smaller portions, allowing a better development and organization, especially in development among multiple collaborators.

This framework also contains some features and tools that facilitate the development of software systems.

One of them is the ability of generic logging and playback, called [ROS](#) bags. That means that in [ROS](#), a message stream can be recorded and saved to later be replayed.

Another tool offered by [ROS](#) is the roslaunch command. Launching multiple nodes one by one is not a practical process, being very time-consuming and confusing. With roslaunch, it is possible to launch all the nodes at a time contained in a specific extensible markup language ([XML](#)) file which is read by the command.

Visualization and monitoring of programs can be done writing simple programs to subscribe to topics, reading the data retrieved by that topic. However, [ROS](#) provides a visualization program called rviz, which allows viewing of a large set of datatypes.

Multiple instantiations of a node or a cluster of nodes is also possible, by assigning a namespace when launching the program and ensuring that there are no name collisions, leading to not being required to modify the code of the files.

With [ROS](#), it is possible to work with a simulated robot that represents the real machine, making it easier and faster to test the devoped programs. It also supports hardware interfaces for most of the components like cameras and motor controllers.

Currently, there are two versions of [ROS](#), [ROS1](#) and [ROS2](#), each one containing multiple distributions. The first version of [ROS](#) is reaching its End-of-Life ([EOL](#)) state, meaning that eventually it will no longer be maintained, and [ROS2](#) will be the only maintained one.

[ROS2](#) is especially interesting due to the fact that it supports Micro[ROS](#), a variant of [ROS](#) capable to run natively on embedded microcontrollers running real-time operating systems.

2.1.1 [ROS1](#) teaching material

[ROS1](#) contains a vast quantity of information and examples dedicated to teaching purposes.

There is the official [ROS](#) documentation, called "[ROS](#) wiki"³ that explains and exemplifies some concepts and teaches how to use some tools. There is also information to build simple programs.

Regarding books, one of the most famous is "A Gentle Introduction to [ROS](#)", also known as "agit", and written by Jason M. O'Kane⁴. This book addresses initial [ROS](#) topics in a simple and very well explained way.

²<http://wiki.ros.org/Packages>

³<http://wiki.ros.org/Documentation>

⁴<https://jokane.net/agitr/>

One of the advantages of ROS is its big community and industry adhesion. That created a big online community, building a big network of educative material containing information regarding various questions and applications.

2.1.2 ROS2 teaching material

With the End-of-Life of ROS1, there is a flow of ROS1 users passing to this later version. However, this conversion is moving slowly and there is not as much teaching content available created by the community. Just as in ROS1, ROS2 has official documentation⁵ explaining concepts and commands and has tutorials to learn ROS2 using simple examples.

Regarding books, it was suggested by a questionnaire that will be discussed in a later chapter "A Concise Introduction to Robot Programming with ROS2", written by Francisco Martín Rico⁶. However, at the time of writing this document, the book is not yet available.

2.2 Examples of robots suited for AI Robotics

Robots consist of three main components: Actuators, Sensors and Control Systems. Briefly, actuators are components responsible for the movement, for example, a motor. Sensors are components that obtain information from the real world, like reading the temperature in a room. The control system is responsible to process the information obtained by the sensors, and controls the actuators. For example, a command to turn on the fan when the read temperature is above 30°C.

There are many types of robots suitable for many applications. One of the most common types of robots are the manipulator robots, very useful in the manufacturing industry in tasks like pick-and-place and cutting, screwing, and so on. Another common type of machines are the mobile robots, with increasing use in storage and transportation systems, like warehouses.

Looking at the work of F. Bellas et al.[7], the purpose was to create an educational robot to be used in a robotics subject. Their approach was to produce a low-cost robot so that institutions could afford a decent quantity of them to have multiple robots at the student's disposal, being able to work in a small group and have a stronger interaction with the real machines. They also had attention in creating a long-term duration mobile robot so it doesn't get outdated quickly, by being able to update the robot in software and hardware aspects. Another important feature they had in consideration was the use of various sensors and actuators, as well connectivity features, adding robustness and functionalities to the robot.

Another important aspect given by Francesco Mondada et al. [2] work, is that a robot should have a small size. Also, the information about the robot needs to be open, and using open source material for the robot is one way to accomplish this goal. All those three aspects provide a better experience to the student/developer, making the development less complex and confusing, facilitating the learning, and speeding the project's conclusion.

⁵<https://docs.ros.org/en/foxy/index.html>

⁶<https://www.routledge.com/A-Concise-Introduction-to-Robot-Programming-with-ROS2/Rico/p/book/9781032264653>

Next will be presented some examples of robots targeting different levels of expertise.

2.2.1 BulbRobot

The BulbRobot [1] is a simple robot based on the Whaveshare AlphaBot 2⁷. It uses a camera and a chrome-plated light bulb, making it a mirror. Having the camera on the base of the robot pointing to the mirror part of the bulb (the bulb is centered and elevated from the camera) creates an omnidirectional vision to the robot. Figure 2.1 shows the robot to better understand the construction. The bulb is fixed and elevated by a support with four legs and offers adjustable height, so it can be easier to adjust to the ideal height. The robot despite being simple in terms of hardware, the vision system provides flexibility making the BulbRobot capable of executing various applications.

The Alphabot 2⁷ is a two-layer structure (one on top of the other) and uses the Raspberry Pi ([RPi](#)) Computer. More precisely, the BulbRobot used the [RPi3 Model B+](#). The base board has two infrared ([IR](#)) sensors for obstacle avoidance, five [IR](#) sensors for line tracking, and it has an interface for an attachable ultrasonic sensor. As actuators, it has four Red, Green, Blue ([RGB](#)) Light-Emitting Diodes ([LED](#)s) and two direct current ([DC](#)) motors. The upper base is used to connect the [RPi](#) and has a joystick, an [IR](#) receiver, a servo-motor interface, and a buzzer. Besides these sensors, the robot uses a camera, as previously mentioned. The exact model is the Raspberry Pi Camera Module (B) Revision 2.0.

The price of the whole system is not referred on the article. However, the price of the Alphabot2, not including the Raspberry Pi3, is around 80€ at the time of writing this document⁸. As for the [RPi](#), its cost is around 42€⁹.

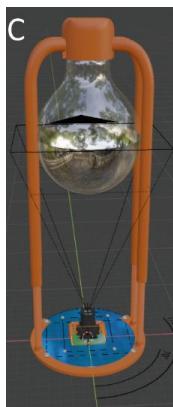


Figure 2.1: The BulbRobot (3D software model) [1]

⁷<https://www.waveshare.com/wiki/AlphaBot2-Pi>

⁸<https://www.botnroll.com/pt/kits-para-montagem/2709-alphabot2-plataforma-rob-tica-m-vel-compat-vel.html>

⁹<https://www.botnroll.com/pt/placas-controladoras-e-kits/2482-raspberry-pi-3-model-b-bcm2837b0.html>

2.2.2 Mona

Mona¹⁰ [8] is a low cost, open source mobile robot developed for both teaching and research purposes based on the ATmega328 microcontroller.

For actuators, Mona has two DC motors with magnetic encoders to provide movement to the machine. As for sensors, it is equipped with five **IR** proximity sensors. The low base quantity of hardware equipment is to make the Mona a low cost and modular robot. A variety of sensors can be added using expansion boards. An expansion board including a **RPi** Zero was designed to add more processing power to Mona.

It supports three communication methods as default: Universal Synchronous/Asynchronous Receiver/Transmitter (**USART**), Inter-Integrated Circuit (**I2C**) and Serial Peripheral Interface (**SPI**). With an add-on board, it is possible to add Wireless Fidelity (**Wi-Fi**), Bluetooth or Radio Frequency (**RF**).

Regarding software, to program Mona, it can be used Arduino. However, pure C or C++, Java, Pascal, Basic and Assembly can be used as well.

A simulation of the robot is available in Stage, an open-source software for the simulation of mobile robot groups in a 2D environment. The Figure 2.2 presents the Mona robot.

The base price of this robot is about 118€ according to the robot's article and the exchange rate at the time of writing this document (the original price is in pounds (£)). Although the price of the extension boards was not specified, the low cost of the base robot and the existence of multiple boards concludes that these might not be included in the 118€ price range.



Figure 2.2: The Mona robot¹⁰

2.2.3 Khepera IV

The Khepera IV¹¹ [9], shown in Figure 2.3, is the fourth iteration of the Khepera series. It is a small mobile robot designed for indoor applications.

This robot is equipped with twelve **IR** sensors, eight around the robot and four downward-facing, five ultrasound sensors, an Inertial Measurement Unit (**IMU**), that is, an accelerometer and

¹⁰<https://uomrobotics.com/robots/mona.html>

¹¹<http://www.k-team.com/khepera-iv>

a gyroscope, two microphones and a color Complementary Metal-Oxide Semiconductor ([CMOS](#)) camera with a resolution of 752x480. It has two DC motors with encoders to provide movement to the machine, three [RGB LEDs](#) and a speaker as actuators. For connectivity, it possesses [Wi-Fi](#) and [Bluetooth](#).

The Khepera functionality can be extended using the Universal Serial Bus ([USB](#)) or [Bluetooth](#) to connect to other devices, or using custom boards using a communication bus. The bus provides power, [I2C](#), [SPI](#), serial, [USB](#) and other more specific lines. Some examples of these boards include a gripper or a laser range finder. Besides the official boards, exists some 3rd party ones as well that provide other functionalities to the robot.

As for the processing power, the robot uses an embedded computer called Overo FireSTORM COM from Gumstix, running [GNU/Linux](#). It also has a microcontroller (dsPIC33FJ64 GS608) that has the function to control the hardware.

Finally, for software, libraries for the robot are available. As for the programming languages, it supports C/C++ and python.

Regarding the price, official retailers available on the official Khepera website sell it for 3240€¹². The extension boards are not included in the pack contents.



Figure 2.3: The Khepera IV¹¹

2.2.4 Thymio

Thymio¹³ [10] is a low cost mobile robot, presented in Figure 2.4. Regarding sensors, Thymio is equipped with seven proximity sensors, two infrared sensors pointing to the ground, an accelerometer, a thermistor, a microphone, five capacitive touch buttons and an infrared remote control receiver. It also has 39 [LEDs](#), two motors for the wheels and a speaker as actuators. The robot includes a memory-card slot, a [USB](#) connection and a trailer hook as well. A wireless version of Thymio is available and includes the wireless module and a [USB](#) dongle.

For software, Thymio uses Aseba, which is a programming environment. For PC, Aseba has an Integrated Development Environment ([IDE](#)) that supports Visual Programming Language ([VPL](#)), a script language, and Blockly. Aseba also supports [ROS](#) using the asebaros bridge.

¹²<https://www.generationrobots.com/en/402241-khepera-iv-mobile-robot.html>

¹³<https://www.thymio.org/>

The normal version robot is listed for 151.75€¹⁴, while the wireless version cost 195.42€¹⁵.



Figure 2.4: The Thymio robot¹⁴

2.2.5 E-puck and e-puck2

The e-puck¹⁶ [2] is a small mobile robot that offers a base set of sensors, actuators and capabilities. However, this robot allows expandability of features, sensors and actuators using physical extension boards, making it suitable for a wider range of applications. Figure 2.5 shows an e-puck robot.



Figure 2.5: The e-puck robot with the basic extension board [2]

The base of the e-puck contains eight infrared proximity sensors, an 3D accelerometer (with later versions also having a 3D gyroscope¹⁷), three microphones and a color CMOS camera with a resolution of 640x480. As actuators, it is equipped with two stepper motors used for the robot movement, a speaker, eight red LEDs, and a set of green LEDs around the body. It also has one stronger red LED next to the camera. Looking at the connectivity, it supports RS232 serial communication and Bluetooth.

¹⁴<https://www.generationrobots.com/en/401213-mobile-robot-thymio-2.html>

¹⁵<https://www.generationrobots.com/en/402337-wireless-thymio.html>

¹⁶<http://www.e-puck.org/>

¹⁷<https://www.gctronic.com/doc/index.php/E-Puck>

The basic version includes a default extension board in which contains an infrared remote control receiver, an RS232 connector, a speaker, a reset button and a 16 positions rotary switch. There are more extensions available for the robot. Some examples are an extension with a turret with three linear cameras, a rotating scanner equipped with infrared or three analog sensors to measure the ground color. Figure 2.6 shows some examples of the extension boards.

Regarding software, it is provided a bootloader to program the robot via Bluetooth, a library to drive the hardware and a monitor for remote control of the e-puck with a computer.

The e-puck is supported in various simulator environments, some of them being Webots and Enki.

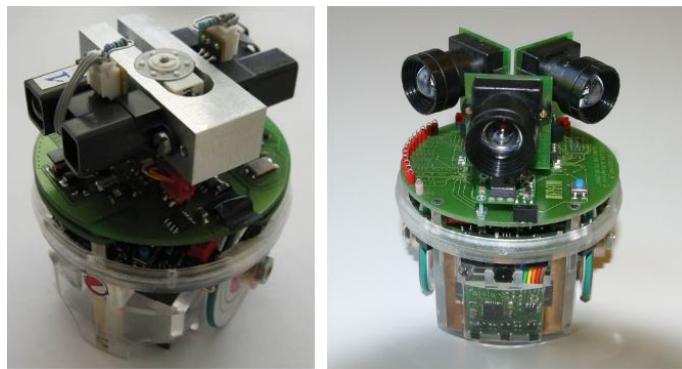


Figure 2.6: Examples of e-puck extension boards: an infrared distance scanner(left) and turret with three linear cameras (right) [2]

A second generation of the e-puck, called e-puck2¹⁸ is already available, improving some features while adding others. Figure 2.7 presents this robot.

Regarding sensors¹⁹, it is still used the same **IR** sensors and it was added a Time of flight (**ToF**) sensor, on the **IMU** it was added a 3D magnetometer, it has one more microphone (four in total) and the camera is still the same. The **IR** receiver and the 16 positions rotary switch are the same, but now are available on the robot itself and not on the extension board. As for actuators it uses the same 2 stepper motors, it still has one speaker, and the red **LEDs** are reduced to four, but four **RGB LEDs** were added. For communication, it was added **Wi-Fi** and **Bluetooth Low Energy (BLE)**.

The extension boards developed for the first generation of the e-puck are all compatible with the e-puck2¹⁹.

As for these robots prices, both versions can be found for around 878 euros, with the extension boards sold separately²⁰.

¹⁸<https://www.gctronic.com/e-puck2.php>

¹⁹<https://www.gctronic.com/doc/index.php/e-puck2>

²⁰<https://www.gctronic.com/shop.php>



Figure 2.7: The e-puck2¹⁸

2.2.6 Robobo

Robobo²¹[7, 3], shown in Figure 2.8, is a mobile robot formed by a wheeled base and a smartphone. Regarding actuators, its base is equipped with four motors, two of them responsible for the robot movement and the other two to allow movement of the smartphone, being able to tilt and pan. It also has seven **RGB LEDs**. Looking at the sensors, each motor contains an encoder. The robot also has eight infrared sensors, some of which are pointing to the ground. So, these sensors can be used for object detection, fall avoidance, and ambient light sensing. The base has a microcontroller responsible to control the existing hardware of the base, as well carry other simple tasks and calculations.

Besides the existing equipment in the base, the robot can also use several sensors and functionalities present on the smartphone, such as cameras, gyroscope, accelerometer, magnetometer, Global Positioning System (**GPS**), microphone, speakers, tactile display, connectivity via WI-FI, 4G and Bluetooth and so on. The range of capabilities will vary depending on the smartphone used. The smartphone is responsible to execute the more complex computing tasks.

Regarding software, they developed a framework in Java that allow users the creation of custom apps in Android. A framework using Swift is also available for iPhone Operating System (**iOS**) devices. The framework supports programming languages for beginners, intermediate and advanced users. For beginners, block programming is available, using Scratch or their own **IDE** using Blockly. For intermediate and advanced users, the programming languages Python and **ROS** are supported by the robot.

A simulated version of the robot is also available and can be used in Gazebo and V-Rep simulators.

The base of the Robobo is available to purchase at their website, offering options to buy a single or in a pack with accessories. The unitary price is 399€, while the four robot pack is 1399€²².

²¹<https://theroboboproject.com/en/>

²²<https://theroboboproject.com/en/shop/>



Figure 2.8: The Robobo robot [3]

2.2.7 Robots Comparison and conclusions

To better evaluate and compare the existing robots, and to look at the advantages and disadvantages of each one, the table 2.1 was made. It presents the more relevant features of the robots, mainly the equipment and the price of each machine. For more information about each robot, check the respective robot section.

It is important to clarify that the values correspond to the price attributed on the official sites, or in stores indicated by them. If no robot purchase was found online, the price shown is the one presented in the conference papers and articles about these robots. In those cases, the price might be outdated. It should also be noted that the cost does not include taxes for transportation or any other additional payment required, and the listed price corresponds to the one presented by the websites and stores at the time of writing this document. Also, some of the prices were in other currencies and were converted to euros, meaning that the price can vary depending on the conversion rate.

When comparing the robots it is possible to check that most of them do not accomplish the requirements established. The problem is that some robots are too simple, being very limited in functionalities and processing capacity, or they are well equipped but the cost is too high. Also, some robots have a huge amount of sensors and actuators, which leads to higher complexity either in electrical and mechanical levels. Few robots are equipped with a camera, and most of those use a low quality one, mostly because of the limited processing power that is not able to handle high resolution cameras.

Vision systems are becoming more common, not only because the system's processing power is increasing to the point where it is possible to handle high quality images at high frame rates, but also due to the flexibility and capabilities a camera provides. A robot using a high quality camera is a good concept mostly because it is possible to create virtual sensors from the image captured, while at the same time it reduces the electrical and mechanical system complexity, making it

²³<https://www.generationrobots.com/fr/402241-robot-mobile-khepera-iv.html>

²⁴<https://theroboboproject.com/en/shop/>

Table 2.1: Mobile robots and their main features

Robot Model	Controller/Computer	Sensors	Actuators	Price (€)
BulbRobot (2.2.1)	Raspberry Pi 3 B	2 obstacle avoidance IR ; 5 line tracking IR ; 1 Ultrasonic; 1 IR receiver; 1 joystick; 1 camera;	2 DC motors; 4 RGB LEDs ; 1 servo-motor; 1 buzzer;	122 ^(a)
Mona (2.2.2)	ATmega 328	5 proximity IR ; 2 magnetic encoders;	2 DC motors;	118 ^(b)
Khepera IV (2.2.3)	Gumstix Overo FireSTORM COM; dsPIC33FJ64 GS608;	12 IR ; 1 IMU ; 2 microphones; 1 camera;	2 DC motors; 3 RGB LEDs ; 1 speaker;	3240 ^(b3)
Thymio (2.2.4)	Microchip PIC24F	7 proximity IR ; 2 line tracking IR ; 1 accelerometer; 1 thermistor; 1 microphone;	2 DC motors; 39 LEDs ; 1 speaker;	151.75 ^(c)
E-puck (2.2.5)	Microchip dsPIC30F6014A	8 proximity IR ; 1 accelerometer; 1 gyroscope(later versions); 3 microphones; 1 camera;	2 stepper motors; 1 speaker; 9 red LEDs ; Set of green LEDs ;	878 ^(b)
E-puck2 (2.2.5)	STM32F407	8 proximity IR ; 1 ToF ; 1 IMU with magnetometer; 4 microphones; 1 camera; 1 IR receiver; 1 rotary switch;	2 stepper motors; 1 speaker; 5 red LEDs ; 4 RGB LEDs ; Set of green LEDs ;	878 ^(b)
Robobo (2.2.6)	PIC32 based; Phone processor; ^(d)	4 motor encoders; 8 IR ; Phone sensors; ^(d)	2 movement motors; 2 phone support motors; 7 RGB LEDs ; Phone actuators; ^(d)	399 ²⁴

^a This price only includes the Alphabot2 and RPi3. The cost for the structure and other materials required were not specified.

^b Price does not include the extension boards.

^c Price shown is for the non wireless version.

^d Depends on the phone model.

minimalist. The BulbRobot (2.2.1) is a great example of a low-cost, minimalist, camera-based mobile robot.

This study and comparison only picked a very small amount of existing platforms. In the work done by A.Fernando Ribeiro and Gil Lopes[11] a more complete list of different robots was made. It is also possible to check some existing robotics competitions and events that occur around the

world.

2.3 Summary

This chapter presented some important characteristics and functionalities of **ROS**. It was explored how it works, how it is structured and some additional features this tool contains to program robots.

It also described and compared various robots to provide a better notion of the existing options.

Chapter 3

Questionnaire Results

During this dissertation, a questionnaire was sent to some Professors and researchers with some experience in robotics about the teaching of robotics and **ROS1/ROS2**. The form sent can be seen in appendix A. The summary of the results of this questionnaire is supported by all six responses gathered. Since not all questions were answered by everyone, for each one, the number of responses will be referred, if relevant.

The summary of the survey is divided into a series of subsections, where each one represents the questions asked, and contains a summary of the answers followed up by a small discussion of the results where appropriate.

Before starting to analyze and discuss the questions, it is important to note that there are no right or wrong answers. These answers come from the opinion and experiences of each respondent and those depend on what worked best for them. The discussion made also is an opinion of the gathered results and therefore, it should not be considered the absolute truth.

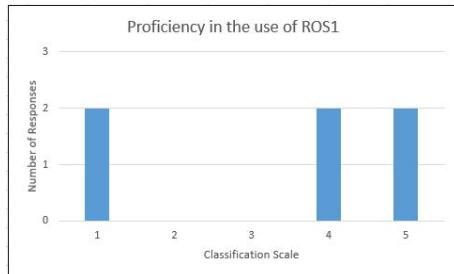
3.1 Classification of experience in **ROS** use and teaching

To better comprehend and analyze the level of capabilities and experience not only in the tools but also in teaching with those, it was made a question where was asked to classify how well they know both **ROS1** and 2 and how experienced they are in teaching these tools. These questions used a scale from 1 to 5, where 1 corresponds to no experience and 5 means very experienced.

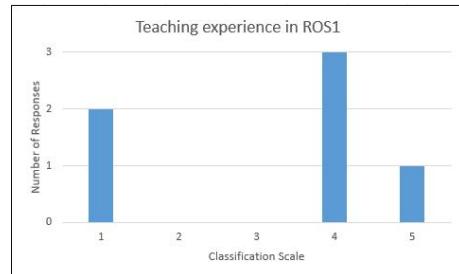
The obtained results are presented in figure 3.1 with the answers gathered by all six respondents represented by a graph, where 3.1a presents the proficiency level in **ROS1** and 3.1b the teaching experience on this tool, and the same can be seen in 3.1c and in 3.1d but for **ROS2**.

Analyzing the graph, it is ensured that there is always at least one person who has experience in a version of **ROS** and in teaching with the tool. Doing a further analysis of the answers given by each respondent, only one person does not have any skills both in teaching and using both versions of **ROS**, while the rest of the people are experienced in using and teaching at least one of these tools.

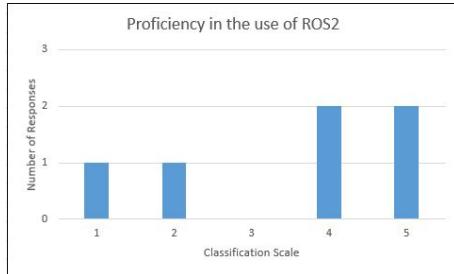
The results of these questions are very important to better analyze the following people's answers and opinions regarding the tools. The use and teaching of these tools require practice, and the better the experience in these fields, the stronger the opinion is about **ROS** and how it should be used and taught.



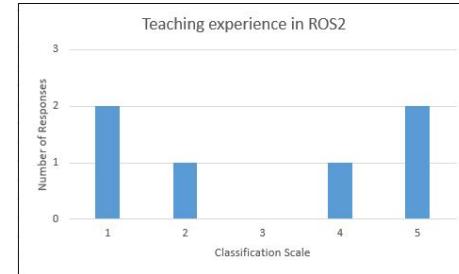
(a) Proficiency in the use of **ROS1**



(b) Teaching experience in **ROS1**



(c) Proficiency in the use of **ROS2**



(d) Teaching experience in **ROS2**

Figure 3.1: Graphs of the responses of personal classification on experience in using and teaching **ROS1** and 2, where 1 in the scale is no experience and 5 very experienced.

3.2 Robotics teaching course and academic level

It was also asked about the course and what academic level the respondents teach robotics.

Of the six people, five answered. Of those, four teach robotics in Electrical Engineering, one in Informatics Engineering, and one in Industrial Informatics and Robotics Master. As for the academic levels they teach, two of them teach in Bachelor's degree, four teach in Master's degree, four in a Doctoral degree, and one teaches robotics in Summer School. It is important to refer that there are more responses than people, meaning that some of them teach in more than one course and/or at more than one academic level.

3.3 Is teaching robotics with **ROS** important? Why?

For this question, five people said yes and only one said no.

The ones that said yes justified with it being the main industry standard, being free and open source, and having a good integration with other tools. It also is well documented, containing a big amount of learning material and simplifying the knowledge transmission. Finally, it provides better visualization of how the main robotics components interact with each other.

As for the no response, the justification is based on the teaching of a lot of robotic concepts that can be taught without the need of **ROS**.

Despite the disagreeing response not being invalid, it is a fact that **ROS** is a relevant tool in the industry¹, as verified by the overall responses. Having some knowledge of this type of robotics programming is relevant because it gives students a better formation if they learn what is most likely to use in the real world.

3.4 Is teaching ROS2 relevant? Why?

Just like the previous question, five people agreed and one disagreed.

The justifications given by the people who said yes consist of the fact that **ROS2** is becoming more relevant due to the **ROS1** End-of-Life (**EOL**) increasing the use of the later version, and eventually will be the official standard in most of the industry. Teaching obsolete technologies and standards does not prepare students for the real world. Also, it was also said that the **ROS2** is more reliable and stable than the previous version.

Who said no, justified with the response of the previous question, that is, **ROS** is not needed to teach robotic concepts.

It is a fact that **ROS1** is reaching his **EOL**. However, this version continues to have a strong community and market influence, so it is not obsolete at all. There is a lot of information available and applications developed that still use the first version **ROS**. Also, some concepts are transversal to both versions. So, despite the growing use of **ROS2**, having some knowledge of the previous version might be useful as well, at least while the last distribution of **ROS1** is still maintained.

3.5 Do you consider it important to work with real robots when teaching robotics? When teaching **ROS**? Why for robotics and why for **ROS**?

In this question, everyone agreed that it is important to use physical robots to teach robotics. Students can gain some mechanical and electronic skills, as well low-level and firmware programming skills, helping them to realize how important all of these skills really are for robotics development. It gives a better understanding of how each component of the system works and can even motivate students by observing the results physically instead of using just a simulation. It also better prepares students for the industry.

Although not everyone responded to the **ROS** questions, the ones they did agree on as well since it is important to deal with parts of the systems and better prepare students for the real-world industry.

Physical robots are important tools that assist the student's formation. Improves the teaching by approximating to real-world applications, since physical systems are used most of the time

¹<https://www.analyticsinsight.net/top-10-ros-based-robotics-companies-to-know-in-2022/>

in the industry, and provides the acquisition of mechanical and electronic skills, giving a better understanding of how the whole system works. It also motivates them to work because they want to see practical results.

3.6 Between C++ and Python, which one do you consider more important to teach ROS with? Why?

This question only got five answers.

Some responses suggested that there is no more relevant language for **ROS**, saying that C++ is better for certain applications, while Python is better for others. Python can be used for basic concept teaching and rapid prototyping, while C++ is considered more suitable for more advanced applications.

On the other hand, some respondents considered Python language better for **ROS** teaching, due to its simplicity, for being a more structured language, or for having more external libraries for robotics, such as Machine Learning or Computer Vision as examples.

Both languages are important. However, the choice of the language depends on the programming knowledge in one over the other, and the type of application to be developed. Python can be used for more simple programs and when execution performance is not important, while C++ is more relevant for more complex applications, mainly where performance is a big factor². Since robotic applications are usually real-time systems that require fast information processing and action execution, and they might contain heavy programs running in multiple threads, then C++ might be the best language for this type of work.

3.7 Do you consider any topic(s) of the book irrelevant (or less relevant) to a first introduction to ROS? Which one(s)?

This question is related to the topics discussed in the book "A Gentle Introduction to ROS" written by Jason M. O'Kane³.

This question got five responses, in which four people responded that all topics are important and relevant for robotics, with one also referring that the level of depth of each topic in the book might not be necessary for a first contact with **ROS**. One person responded that chapter 9, Recording and replaying messages, is not very important for robotics.

The low number of responses to this question demonstrates how well this book introduces **ROS**.

²<https://roboticsbackend.com/python-vs-cpp-with-ros/>

³<https://www.cse.sc.edu/~jokane/agitr/>

3.8 Classify how important you think the following topics of **ROS** are for teaching of **ROS** at an introductory level.

The purpose of this question was to gather some other topics that are only briefly talked about, or not talked at all by the book referenced by the previous question, and asked to classify the importance to teach these in an introductory level of **ROS**. Those were Integrated Development Environment (**IDE**) for **ROS**, utilization of GNU Debugger (**GDB**) (attach to process), actions servers/clients, **ROS** Transform (**TF**), Rviz, simulation 2D/3D and runtime parameters.

Relatively to the answers, there were six, but not all topics have six classifications, having some with only five responses. After a more in-depth analysis of answers, it was found that the respondent that did not classify all the topics is the one with no experience in both versions of **ROS** and, since this question is specific for **ROS**, it was decided to invalidate the classification attributed to the topics that he answered.

Figure 3.2 presents the graphs of the classifications given to each topic. Each topic was classified on a scale from 1 to 5, where 1 corresponds to saying that it is not important, and 5 it is a very important topic to learn in **ROS**.

To better analyze the classification attribution of each suggested topic, the average was calculated. Table 3.1 shows the topic and the resulting average, ordered from highest to lowest. With that, it can be considered that the highest ones are the topics considered more important, and the lowest ranking one the least important.

Analyzing the data, it is safe to say that in general, each topic was considered relatively important to teach, ones more than the others, but nothing had a bad classification.

Table 3.1: Average of the classifications gathered for each topic

Topic	Average
Rviz	5
Simulation 2D/3D	4,6
TF	4,4
IDE for ROS	4
Actions servers/clients	3,8
Runtime Parameters	3,8
GDB (attach to process)	3,4

In fact, all of these topics have their own importance when developing **ROS** programs. Not only expands the knowledge on **ROS** capabilities, allowing the creation of different applications, but also can speed up the development and testing process of the programs. Rviz and **TF** are really important topics to learn since they aid in the application development and are basic tools for robotics and **ROS**. The **IDE for ROS** is also important, especially for starters, so they can have a stable development platform with supporting tools for program development. An example tool is **GDB** to provide faster testing and error identification.

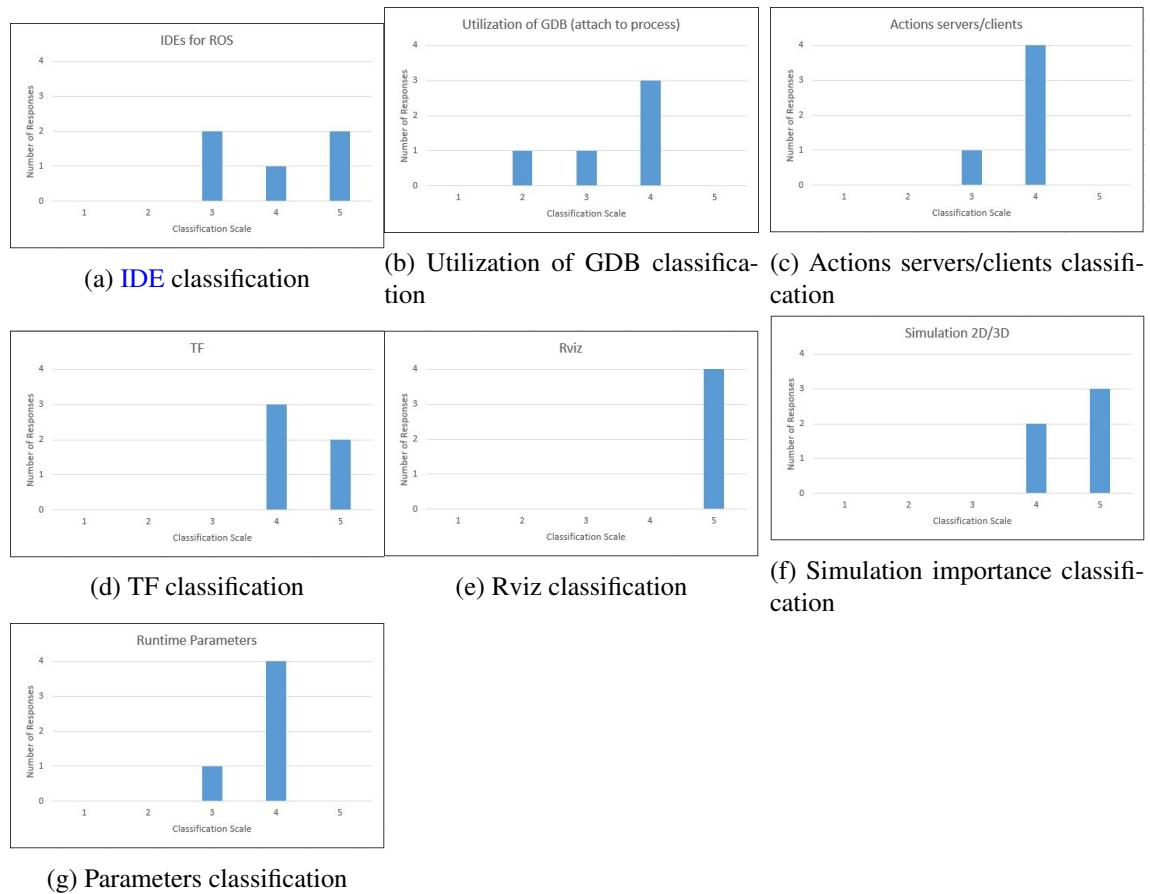


Figure 3.2: Graphs of the responses classifying the importance to learn some topics in ROS

3.9 Besides the topics of the previous question, which topics do you consider the book is lacking?

This question had the objective to check possible suggestions of the respondents on other important topics in ROS.

It got three answers, one being a simulation topic, but this one is already referred on the previous question, another answer was a chapter for important libraries that exists for ROS such as OpenCV, MoveIt, Point Cloud Library ([PCL](#)), among other, and the last one was the addition of some real case examples.

As for the others who did not answer, it was assumed that they did not consider any other topic important than the ones presented.

These suggestions are good ones since the given examples are usually simple and represent fictional cases with the only purpose to teach and show ROS concepts. Real case examples allow a better perception of how certain programs work and also provide a better understanding of when it does make sense to apply certain knowledge acquired in a certain situation. Having a topic dedicated to important, commonly used libraries is also useful, especially for a better understanding of how to use them.

3.10 Do you have any suggestions about books, guides or videos for ROS?

This question also had the purpose to collect suggestions of informative and educative material for ROS.

One person said that both ROS wiki and Answers should be enough in most cases, but it tends to not be updated as frequently as it should. He also mentioned that the Git Issues of the main packages are also a good place to find more information about specific packages.

The other answer recommended "A Concise Introduction to Robot Programming with ROS2" book, written by Francisco Martín Rico⁴.

3.11 Summary

The objective of this questionnaire was to check how important and relevant is considered robotics and ROS teaching from the perspective of those who have experience in these subjects and to understand what helps to have a more efficient and enriching learning experience.

The first few questions were important to provide a clearer picture of what formation areas and in what academic levels ROS and robotics teaching have greater influence. It was possible to understand that most people that teach robotics have knowledge in ROS and teach it to their students. This demonstrates the importance this tool has in student's formation in robotics.

Most people consider ROS2 relevant and even consider that this version is the logical evolution both of the tool and the industry.

Also, it is possible to conclude that everyone thinks that physical robots are a must in robotics teaching since it enriches the student's robotics formation with the acquisition of other skills besides robotics programming.

Regarding the programming language, there is a clear division in what is better for robotics teaching. However, this question also depends in what are the teaching objectives and what is the type of programs and applications pretended for the students to develop.

Lastly, on the book question, since most of the respondents did not eliminate anything regarding the book contents, it demonstrates how well made it is for ROS introduction, approaching the core concepts. However, with the follow-up question (classification of the importance of other ROS topics) it is possible to analyze that there are other topics considered important, and these are only briefly referred, or not referred at all, on the book's contents.

⁴<https://www.taylorfrancis.com/books/mono/10.1201/9781003289623/concise-introduction-robot-programming-ros2-francisco-mart%C3%ADn-rico>

Chapter 4

The Physical Robot

In this chapter, a detailed analysis of the robot used in this dissertation is provided. Firstly, the robot is briefly described, then its materials are listed and the hardware architecture of the machine is analyzed. Finally, the modifications done to the robot are explained.

The files created that are referred in this section are available on a Github repository¹.

4.1 The Robot and its Hardware

For this dissertation, a robot was not built from scratch, but rather utilized one that was already built, proceeding with the required modifications to meet the established requirements. The differential robot presented in the figure 4.1 was originally designed by Professor and researcher Paulo José Cerqueira Gomes da Costa² with the purpose of being a support tool in robotics teaching. It is currently used in Autonomous Systems³, a curricular unit in the second year of the Master's degree in Electrical and Computer Engineering at the Faculty of Engineering, University of Porto (FEUP).

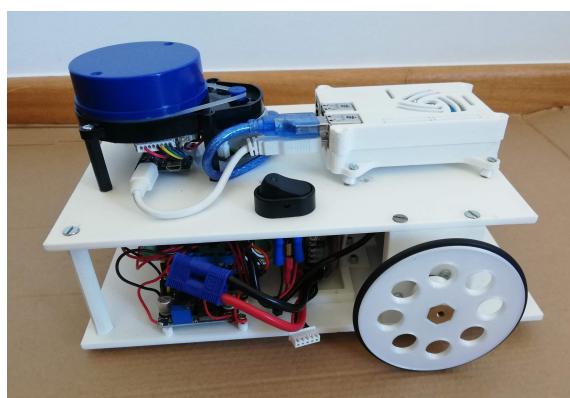


Figure 4.1: Physical robot in its original configuration

¹<https://github.com/AlexMiranda16/DissertationFiles>

²https://sigarra.up.pt/feup/pt/func_geral.formview?p_codigo=211795

³https://sigarra.up.pt/feup/pt/ucurr_geral.ficha_uc_view?pv_ocorrecnia_id=485759

This robot underwent some modifications, namely the attachment of a camera pointing down and the respective support, the upgrade of the Raspberry Pi, and the change of the layout to accommodate the added material. These changes will be explained and justified in more detail later in the chapter. The final robot is displayed in the figure 4.2.

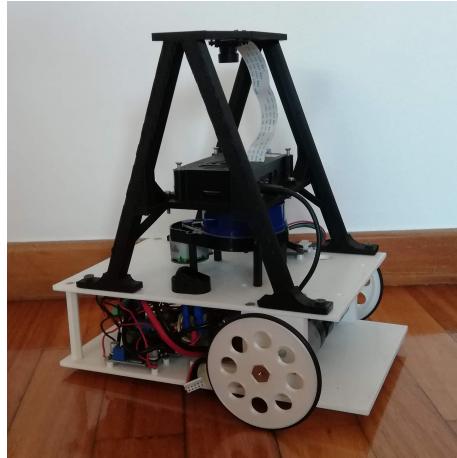


Figure 4.2: Final robot with all modifications applied

4.1.1 List of Material

All of the material listed in table 4.1 was already with the original robot received, with the exception of the Raspberry Pi4 4 Gigabytes (GB) version (which replaces the Raspberry Pi3 of the original robot), the [USB-C](#) cable and the camera. The reasons for adding these new materials will be explained later in this chapter. As for the rest of the material already provided in the original robot, it will not be discussed in detail, since it was already chosen and is out of the scope of this dissertation. In table 4.1, besides the material and its quantities, it also shows the price of each one. It should be noted that the cost does not include taxes for transportation or any other additional payment required, and the listed price corresponds to the one presented by the websites and stores at the time of writing this document. Also, some of the prices were in US\$ and were converted to euro, meaning that the price can vary depending on the conversion rate. The total cost of the robot is 428,59€.

The motors utilized are 12V (Volt) [DC](#) motors equipped with a metal gearbox and an encoder providing a resolution of 2797 counts per revolution (CPR) of the gearbox output shaft. The encoder works at 5V.

The Motor Driver Shield chosen is a board compatible with the Arduino board and can control two bidirectional [DC](#) Motors. This board operates at a range between 5V and 28V and can deliver 3Ampere (A) continuous per channel.

Table 4.1: List of Material of the robot

Material	Quantity	Product Name	Cost (€)
Motor with encoder	2	Gear Motor w/Encoder, No.GB37Y3530-12V 251R	25.37 (each) ⁵
Motor Driver Shield	1	Pololu Dual MC33926	74.30 ⁶
Micro Controller	1	Arduino Uno + USB-A to USB-B cable	17.10 ⁷ (a)
Arduino Screw Shield	1	Arduino Screw Shield V2	6.64 ⁸
Step-Down Converter	1	HCJ-IPM-V2	3.44 ⁹
Battery	1	Hacker TopFuel LiPo 20C ECO-X 2400mAh 4S MTAG	49.82 ¹⁰
Light Detection And Ranging (LiDAR)	1	YDLIDAR X4	79.30 ¹¹
Camera	1	200° , 5MP fisheye camera - JOY-IT	40.20 ¹²
Raspberry Pi	1	Raspberry Pi 4 Model B (4GB version)	63.90 ¹³
Memory Card	1	32GB Micro-Secure Digital card (SD)	5.60 ¹⁴
3D printing filament	1	1 kg Roll - Devil Design	20.20 ¹⁵ (b)
Other material	x	Wires, solder, screws, button, etc	15 ^(c)
USB-A to USB-Micro cable	1		1.10 ¹⁶
USB-C cable with no end connector	1		3.65 ¹⁷
Micro-USB cable with no end connector	1		3.20 ¹⁸
TOTAL			428.59

^a The selected Arduino already comes with an **USB** type A to **USB** type B cable.

^b Due to the fact that most of the robot structure is 3D printed, it is not possible to determine the total quantity of material required to be printed. However, 1 kg of filament should be enough to print all pieces. The printed pieces for this dissertation (camera support and **RPi4** case) utilized a total of around 200 grams of material. In this case, the pieces were printed with Polylactic Acid (**PLA**)+ filament, but other materials can be used.

^c The value assigned to this material is not exact. It is only an estimate with a margin of error.

The **Arduino Uno**⁴ is a famous micro-controller commonly used in small electronics and robotics projects. It is based on the ATmega328P with a clock speed of 16 MegaHertz (MHz) and has 14 digital pins and 6 analog pins.

The step-down converter is a **DC** to **DC** converter, with the output voltage adjustable from 1.25V to 35V, and a maximum output current of 3A.

The battery operates at 14.8Volts with a capacity of 2400mAh. It contains 4 cells and a voltage indicator, displaying the battery charge status with four **LED**.

The **LiDAR** rotates 360° being able to scan omnidirectionally. It has a range up to 10 meters and its angle resolution is between 0.43 to 0.86°.

The recommended size for the Micro-**SD** memory card is 32GB. This is due to the size of all software needed to install.

For the **RPi4** and the camera, as previously said, those will be detailed later in the chapter.

⁴<https://store.arduino.cc/products/arduino-uno-rev3>

4.1.2 Hardware Architecture

Figure 4.3 represents all the connections in the hardware, giving a good visualization of how the whole system is interconnected. For all components, the inputs are represented on the left side and the outputs on the right, with the exception of the **USB** since the communication can be bidirectional, being an input and output at the same time. A color scheme was added to differentiate the connections, grouping the traces by a specific voltage, or a type of cable. A figure legend was added to better identify the reasoning for each group.

Note that every connection is done through a cable or wire, except the analog and digital pins, because the Motor Driver shield board and the Arduino board are coupled together. As for the **USB** cables in the figure, the one connecting the Raspberry Pi and the Arduino Uno is a **USB** type A to **USB** type B cable, and the connection between the Raspberry Pi and the **LiDAR** is done using a **USB** type A to **USB**-Micro cable. To power the Raspberry Pi, a **USB** type C cable is used with the other end directly soldered to the output power of the step-down converter, and to power the **LiDAR** it is used a **USB**-Micro cable with the other end soldered on the same output as the **RPi** power cable is soldered.

This last cable was initially removed because it was thought that it was not needed. However, it was re-soldered so the **LiDAR** can be powered externally by the Step-Down converter. On the original robot, this sensor was powered only using the data port present on its board, being enough to correctly function. Unfortunately, when the equipment was tested with the new Raspberry Pi, it was not working properly due to the insufficient power provided by this newer computer. For this situation where there is a low supply of energy to the **LiDAR**, its board contains another **USB** port designed specifically to power the sensor using an external power source. Having this in mind,

⁵<https://www.dfrobot.com/product-634.html>

⁶<https://www.pololu.com/product/2503>

⁷<https://www.ptrobotics.com/plataforma-arduino-e-modelos-alternativos/4033-arduino-uno-r3-compativel-c-cabo-usb-da-funduino.html>

⁸<https://www.ptrobotics.com/arduino-proto-screw/8677-arduino-screw-shield-com-terminalis-de-parafusos.html>

⁹<https://www.ptrobotics.com/alimentacao/2647-modulo-conversor-dc-dc-step-down-lm2596.html>

¹⁰<https://www.ptrobotics.com/baterias-lipo/9496-bateria-lipo-148v-2400mah-xt60.html>

¹¹<https://www.amazon.com/youseetoo-Scanning-Obstacle-Avoidance-Navigation/dp/B07JGV5JTG?th=1>

¹²https://mauser.pt/catalog/product_info.php?cPath=1667_2620_2956&products_id=096-7651

¹³https://mauser.pt/catalog/product_info.php?products_id=096-7402

¹⁴https://mauser.pt/catalog/product_info.php?cPath=641_2548_1574&products_id=047-2895

¹⁵https://mauser.pt/catalog/product_info.php?cPath=2207_2402_2308&products_id=096-8418

¹⁶https://mauser.pt/catalog/product_info.php?cPath=1874_57_107_2260&products_id=047-2826

¹⁷https://mauser.pt/catalog/product_info.php?cPath=1667_1753&products_id=047-3229

¹⁸https://mauser.pt/catalog/product_info.php?cPath=1874_57_107_2260&products_id=047-3227

now the **LiDAR** board has its two **USB** ports being utilized, one for data communication, and another for power.

The figure 4.4 is a close-up picture of the real connections of the hardware, without the raspberry, **LiDAR** and camera components.

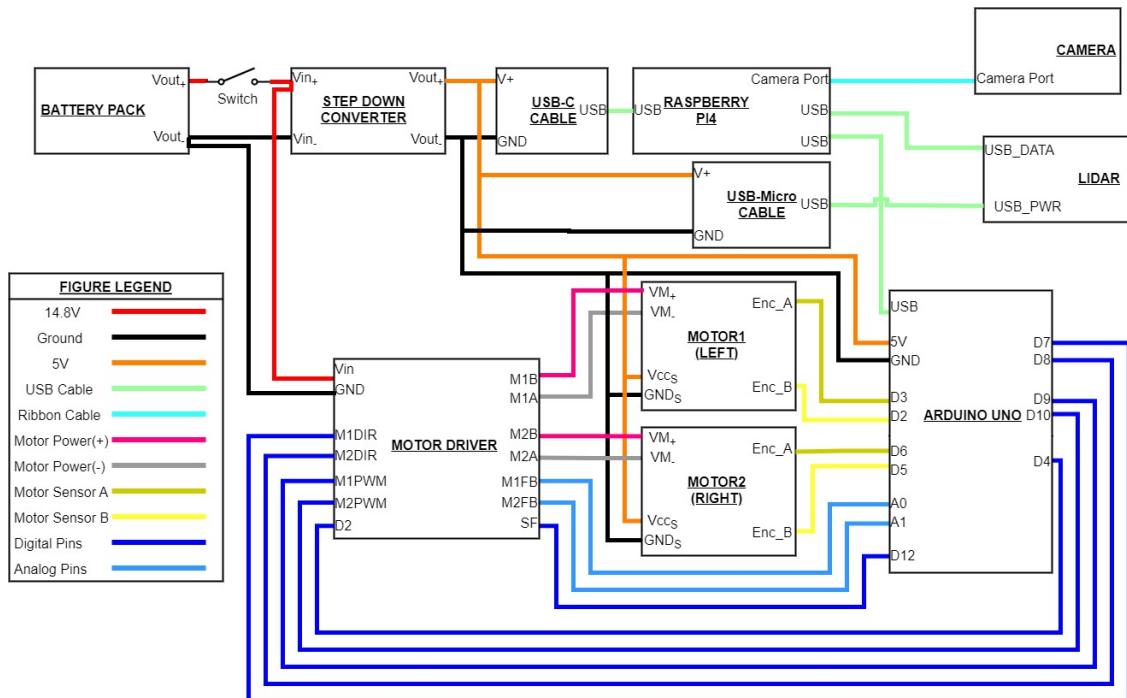


Figure 4.3: Schematic of the Hardware Layout

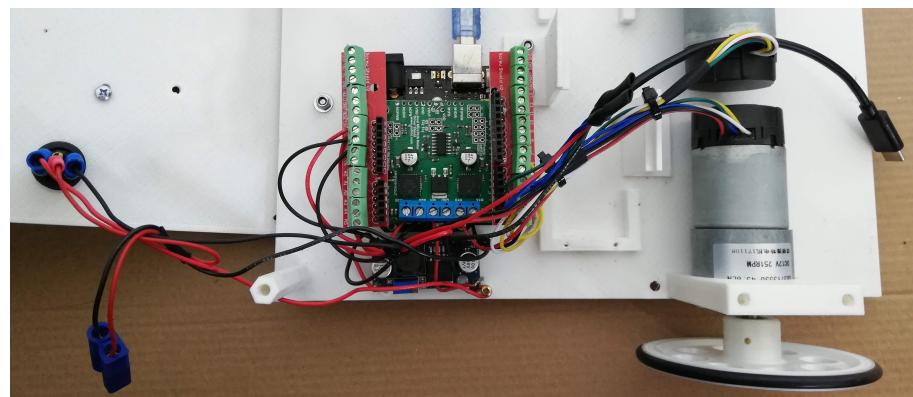


Figure 4.4: Hardware layout of the physical robot

4.2 Hardware and Design Modifications

To accomplish the requirements proposed in this dissertation, it was required to do some modifications to the robot. In the following sections, it will be detailed the modifications done, as well the procedures and justifications of the reasons to make them.

4.2.1 Camera support

Since the original robot has no support to install the camera in the desired configuration, it was required to create one. For this, Autodesk Fusion 360¹⁹ was used. The choice of this tool is essentially due to its ease of learning and use. With this software it is possible to design 3D models, being a viable tool to develop the camera support structure, with the ability to recognize more easily faults and problems associated with the design.

To better design and visualize the support, a construction of the physical robot was made in the software. Some measurements of the real model were made to more accurately design the virtual one in Fusion360, so the 3D model of the robot is approximately a simplified 1:1 scale of the real one. The figure 4.5 represents the robot made in the software. Some components were not drawn since they do not add value to the overall dimensions and, therefore, do not interfere with the implementation of the support.

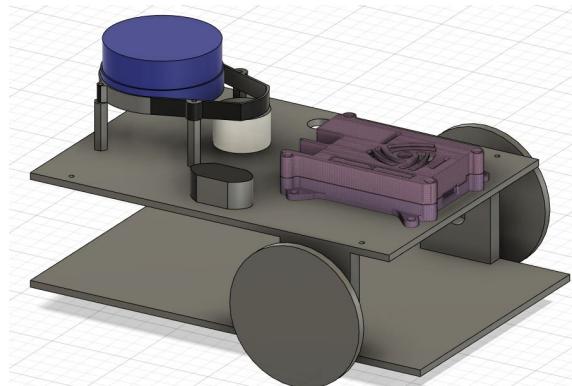


Figure 4.5: 3D model of the robot in its original configuration

Having the 3D model of the robot, it is easier to start doing some possible camera supports. The first supports were made at the front of the robot, trying to maintain the original configuration. However, although this configuration gives a really good vision at the front, and the sides are limited only by the support itself, the rear view is really limited, not only due to the long profile of the robot, but also due to the height of the [LiDAR](#), obstructing the rear view. The following figures 4.6a, 4.6b and 4.6c represents some of the first prototypes of the support.

Having the previous problems in mind, not only the camera support needed to be more centered but also the [LiDAR](#) needed to be moved. Moving only the camera support to the center is not

¹⁹<https://www.autodesk.pt/products/fusion-360/overview?term=1-YEAR&tab=subscription>

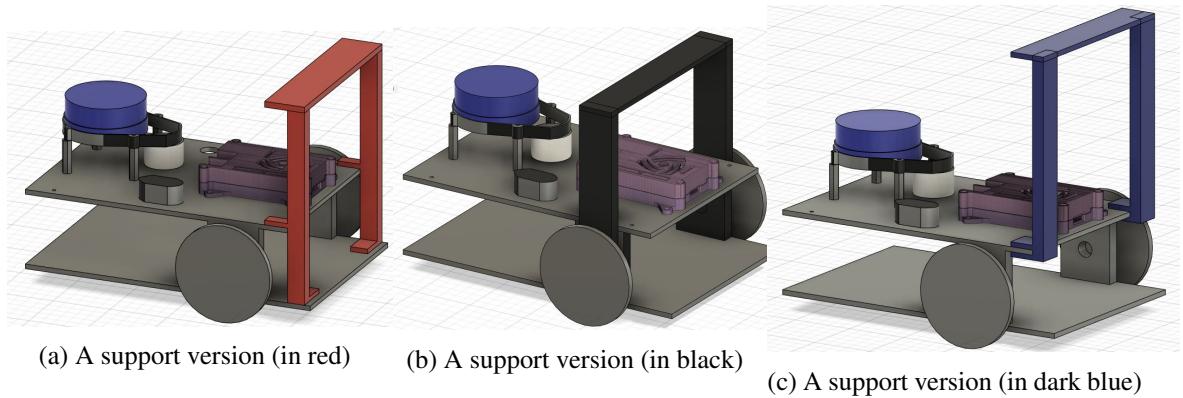


Figure 4.6: 3D model of the robot with different versions of a frontal camera support in red - (a); black - (b); and dark blue - (c)

enough because, despite improving the vision at the back, it is still obstructed significantly by the LiDAR height. So, both the LiDAR and the support need to move. Moving the LiDAR to the center of the robot will interfere with the current position of the Raspberry Pi, so the RPi also needs to move.

To comply with the required changes a solution thought was to place both the LiDAR and the Raspberry at the center, below the camera. The best layout is the one shown in the figure 4.7 corresponding to the final solution, where the LiDAR is at the bottom, the RPi is at the middle, and on top is the camera. This means that the camera support not only holds the camera itself but also the RPi. This layout was chosen not only to solve the problems already discussed but also to avoid some problems that would cause if a different arrangement of the components was used. More specifically, it avoids the need to purchase a new longer ribbon cable to connect the camera to the RPi, since they are near enough to use the one provided with the camera, not interfering too much with the vision. Also, the heat caused by the RPi is dissipated more easily, due to the fact of not being obstructed by another component. If the LiDAR was on top of the Raspberry, the heat generated would be transferred to the LiDAR, not being ideal. Finally, a lower position of the LiDAR is preferred. Objects are only detected by the sensor if they are at the same level as the LiDAR, not detecting the ones that are lower. So, to have a reliable detection of objects, the lower the LiDAR is in the robot the better.

Some changes were also made to the support to provide a platform for the Raspberry, to improve the overall stability of the structure, and to improve the camera vision at the lateral of the robot.

The figure 4.8b shows only the camera and RPi support made. Since this will be printed in a 3D printer, the model was divided by half without compromising the overall rigidity and stability of the piece, as exemplified in the figure 4.8a. This allows a lower printer time and avoids excessive use of printing support material, reducing the filament waste and lowering the cost per model printed. Despite both pieces appearing to be identical, the screwing holes for the Raspberry case are not in the same position. Therefore, it is required to print both pieces, and not only the

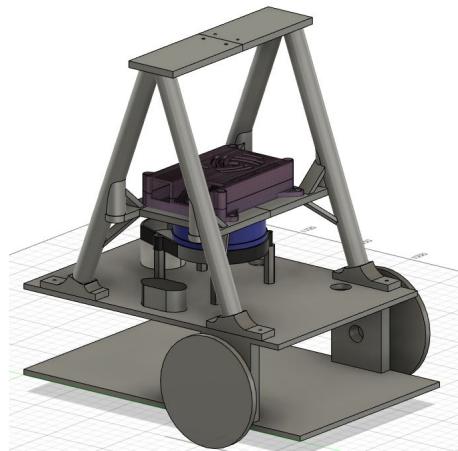


Figure 4.7: 3D model of the robot with the final version of the support

same one two times. Depending on the printer settings, it uses around 80 grams of material and about 8 hours to complete printing each piece.

The printed support, already with the Raspberry Pi and the camera place, is presented in the figure 4.8. To attach the modifications to the physical robot it was required to drill some holes in the upper base of the robot. Eight new holes were drilled to fix with screws both the [LiDAR](#) in its new position and the camera support. Due to the new position and orientation of the [RPi](#), and the shortness of the new [USB-C](#) cable acquired to power this computer, another hole was drilled near the [USB-C](#) port as a pass-through for the cable. Also, to avoid buying a new [USB](#) cable to connect the [LiDAR](#) to the Raspberry, the board was moved to the location shown in the figure 4.9 allowing to use the cable provided with the original robot. Four new holes were drilled to fix this board.

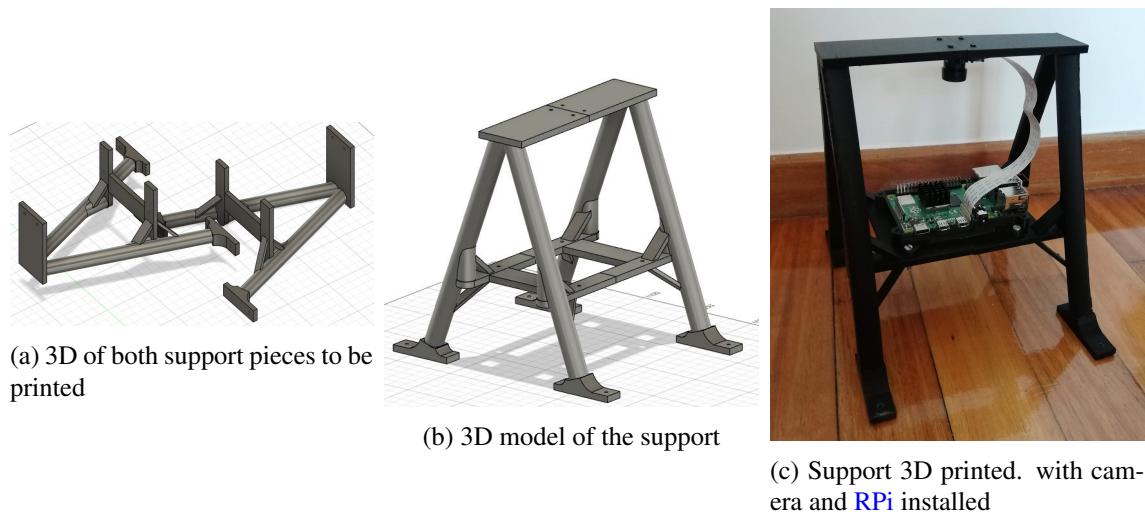


Figure 4.8: Final camera support, in a virtual 3D model - (a); Split in half version for 3D printing - (b); 3D printed version, with Raspberry and camera installed - (c)



Figure 4.9: New location of the LiDAR board

4.2.2 The Raspberry Pi upgrade

The original robot contains a Raspberry Pi 3B V1.2. Despite this Micro-Computer being enough to develop basic robotic applications, it was replaced with a Raspberry Pi 4 Model B 4GB of Random Access Memory (RAM) version. This upgrade was done because the RPi4 is significantly faster²⁰ than the RPi3 and the RAM was increased from just 1GB to 4GB. Another improvement was in the Input/Output (I/O), now equipped with 2 USB3.0, meaning faster transfer speeds over USB. All these upgrades, mainly the faster processor and the larger memory allow the development of more heavy and advanced robotics applications reducing the probability of suffering a bottleneck.

However, the upgrade to the Raspberry Pi 4 required some modifications to the robot. The Micro USB cable used in the robot to power the RPi3 was replaced with a USB-C one. Due to the shortness of the cable acquired and the new position and orientation of the RPi, a new hole was drilled to pass the cable.

The already printed case for the RPi3 is not compatible with the RPi4 due to the changes in the I/O. Fortunately, a similar case but for the Raspberry Pi 4 was already designed, and available at the Thingiverse website²¹ so, there is no need to redesign any structure created. Both the top and the lower case, depending on the 3D printer settings, take about 2h30min and 20 grams of material each.

The Raspberry Pi chip gets really hot²⁰, so a heat sink and thermal paste were added for better heat dissipation. The 3D printed top case touches this heat sink enough to not allow the case to be totally closed so, the four screws used to close the case were screwed enough to fix it but not all the way down to avoid too much tension.

²⁰<https://medium.com/@ghalfacree/benchmarking-the-raspberry-pi-4-73e5afbcd54b>

²¹<https://www.thingiverse.com/thing:3714695>

4.2.3 The Raspberry Pi camera

Another important component was the camera to be used in the robot. There are some key features for a camera to be a good candidate for the robot. One of them is the camera resolution. The better resolution, the crispier the image is, enhancing the definition of the objects especially the ones further away from the robot. This improves the accuracy and the range of the detection of the objects by the software. However, an image with too many pixels means more processing time, lowering the number of processed images per second. Having a too low rate of processed images is particularly bad because the information can become outdated, obligating the robot to lower the speed of its movement or, in a worst-case scenario, to collide with an object due to the real location of the object not being processed in time. Another important feature is the angle of view. Since the camera is on the top of the support pointing downwards, with the purpose of achieving a 360° view around the robot, the camera itself must have a big angle view. A too narrow viewing angle limits the range of viewing around the robot. Lastly, the price is an important parameter. To keep the cost low, a good quality camera with a low cost is important.

The selected camera complies with these key factors. It is equipped with a 5 Megapixels(MP) sensor, and capable of recording in 1080p at 30 frames per second, which is good enough to give an image with a decent resolution, and not big enough to give major bottlenecks when processing the image. The lens used is a fish-eye one, which gives an angle view of 200°. This allows viewing all around the robot, limited only by the distance from the robot to the object (due to the resolution). This camera connects to the Raspberry in the camera module port using a ribbon cable (which came included with the camera).

4.3 Summary

This chapter focused on the hardware of the robot, listing all the materials used and explaining how these are interconnected. It also explained all the modifications required to do on the original robot to accomplish the pretended system. It explained all the procedures followed for the camera support creation, the reasoning for the Raspberry Pi upgrade and it was listed the major characteristics needed for a camera that leads to the one that it is using.

Chapter 5

The Software

During the dissertation, multiple code files were created, imported, or modified and various software tools were used. This chapter focus on explaining everything done during this dissertation related to software. It will list the software tools used, the work involved in environment configuration, and the development of code/programs.

Figure 5.1 depicts the structure of the whole robot system, software and hardware. It represents where each developed program runs and its interaction with other programs, and the interconnection between all robot's hardware.

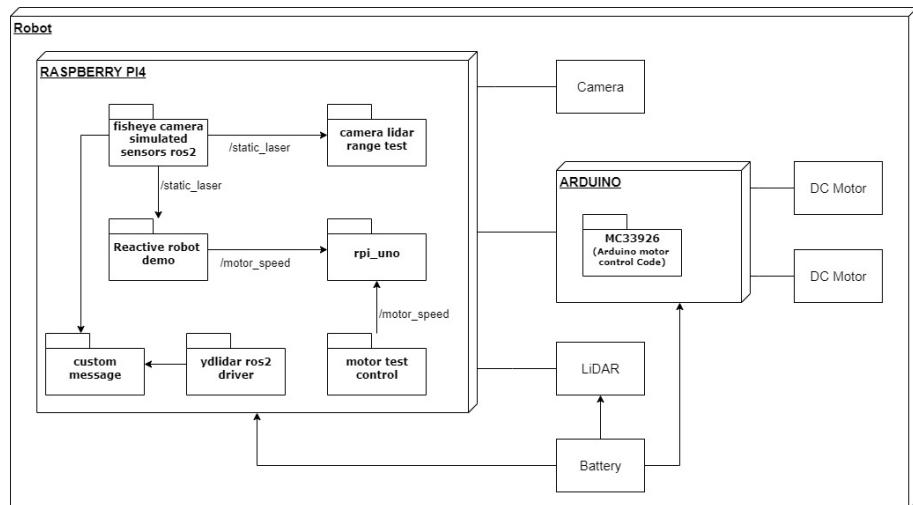


Figure 5.1: Software and Hardware structure

5.1 Requirements

This section has the purpose to list and introduce all the pieces of software and other tools used during this dissertation. More information about it, and about how to properly install and configure the environment is presented in appendix B.

The Raspberry Pi requires an Operating System ([OS](#)) so it can be properly used. The installation was the official one, called Raspberry Pi [OS](#), and the version chosen was the Debian Buster. This version is not the most recent at the time of this dissertation, but the guides followed to install other software used it, and to avoid installation problems related to [OS](#) versions, it was chosen to install the same one as the guides.

Since one of the objectives of this dissertation is to use the robot for [ROS](#) teaching, this software also needs to be installed. Both versions of [ROS1](#) and [ROS2](#) were installed, with the distributions being Noetic Ninjemys and Foxy Fitzroy respectively.

The physical robot contains an Arduino Uno and to more easily program and load code to this micro-controller, the Arduino [IDE](#) was installed. Besides this [IDE](#), the Visual Studio Code ([VSCode](#)) was also installed for the creation and development of [ROS](#) code.

Appendix [B](#) also has information on how to fix device ports assigned by the Operating System. The lock of these ports is important because some programs use them to connect the Raspberry Pi to other devices for communication. That is the case of the Arduino Uno and the [LiDAR](#). Appendix [B](#) also explains how to configure a new fixed port name.

Lastly, it was installed the Virtual Network Computing ([VNC](#)) Server. This is a piece of software that provides remote access to the [RPi OS](#) from another computer. It is easier to work and debug while the robot is being tested with certain applications.

5.2 [ROS2](#) and Applications Development

This section briefly explains the developed and modified [ROS](#) packages for the robot. All the files referred here are available in a GitHub Repository^{[1](#)}. This GitHub also has an explanation about the prerequisites and the necessary steps to run the practical applications developed. Appendix [D](#) is a copy of this explanation guide.

5.2.1 [ROS](#) code Migration

Some source code designed for [ROS1](#) belongs to other authors. However, these original files were modified to be supported in [ROS2](#). The procedures to migrate packages originally made in [ROS1](#) to the later version, [ROS2](#), are explained in the appendix [C](#). Briefly, certain parts of the files and code suffer modifications in a way to correspond to the structure and naming of [ROS2](#). These changes will depend on the complexity of the packages to migrate.

The migration process was performed on the created package for the camera, provided by the master's student in computer engineering Vítor Ventuzelos^[12], and the code was developed using the [ROS](#) Noetic distribution. Another package migrated was the one responsible for the [LiDAR](#) developed in [ROS](#) Melodic distribution. This package is available on the official [LiDAR](#) website^{[2](#)}.

¹https://github.com/AlexMiranda16/Physical_Robot_Introduction_ROS2

²https://www.ydlidar.com/service_support/download.html?gid=5

5.2.2 ROS2 packages explanation

In this subsection, a brief description of created/modified packages is given, explaining their purposes and how they work.

However, not all packages developed are in this subsection, because the remaining ones have the intent to create some practical function for the robot. For that reason, they are explained in section 5.2.3.

5.2.2.1 RPi-Uno package

This developed package works as a bridge between the Raspberry Pi, running ROS2 programs, and the Arduino Uno. It is responsible for the communication between the ROS2 running programs, and the program executing on the Uno board. This communication is serial via USB cable connected to both RPi and Arduino USB ports.

The node on this package (called serial_com) sends/receives messages to/from the Arduino board. More precisely, it contains one subscriber that receives from another ROS2 node the speed commands to apply to the robot motors. These are then sent to the Arduino program. It also has 2 publishers responsible to publish the current and odometry values of each motor. The information of these sensors is received from the Arduino program. Chapter 5.3 contains more information about the Arduino program.

5.2.2.2 Fisheye camera package

As previously mentioned, this package is authored by Vítor Ventuzelos and it was modified so it can be compatible with ROS2 Foxy. The objective of the package is to apply an image processing algorithm to the image captured by the camera, with the intention to create virtual sensors, that is, to imitate the behavior of different physical sensors.

It contains various nodes with different functionalities but the one used by the programs or applications that will be referred later in this chapter only makes use of the "simulated_lidar" node. This node, as the name says, with the image processing simulates a LiDAR sensor. It simulates various laser beams all around the robot and estimates the distances of the objects near it. This information is then published on a topic to be available to other nodes.

The other nodes in this package are "object_detection", which detects objects around the robot (in four positions: at the front, back, left and right of the machine) and classifies them on a scale from zero to three, where three is the closest to the robot. These levels are associated with the danger level. There is also the "proximity_sensor" node with the function to make approximated measurements in five directions in front of the robot.

This package requires other packages to properly work. Those are "cv_bridge", "OpenCV" and "image_transport". It also needs an Application Programming Interface (API) to use the Raspberry Pi camera with OpenCV, called RaspiCam. At the end of the section B.3 (ROS2 Foxy Fitzroy installation) on the appendix B it is explained how to get and configure them on the system.

5.2.2.3 LiDAR package

As for the physical LiDAR package, its program is responsible to control and collecting the data gathered by this sensor. This node (ydlidar_ros2_driver) is responsible to configure the communication since it communicates with the RPi through a serial port (uses USB), and has the function to adjust and setup the settings of the LiDAR mode of operation. The laser information with the distances measured is published on a topic to be available to other nodes.

Besides the package conversion from ROS1 to ROS2, some more modifications to the code were made, mainly the change of the baud rate value to 128000, the typical value according to its datasheet².

On the official LiDAR website², there is the Software Development Kit (SDK) that is required to enable use of the sensor. It is also available documentation for further information about the specifications or how to properly configure and use it.

5.2.3 Applications created

To test parts of robot hardware and to test some of the system potentials, it was developed three small and simple applications. Each demo corresponds to a ROS2 package and requires some of the packages mentioned in the previous section.

The first demo is the "motor_test_control" package which has the objective to test the motors, the second one is the "camera_lidar_range_test" package that measures the distance of near objects at the front and back of the robot, and the last demo is a simple obstacle avoidance application.

These demos are available on the GitHub Repository¹ along with an explanation on how to setup and execute them. Appendix D is a copy of this guide, but it is recommended to consult the actual web page to visualize video examples that are not possible to be seen in the appendix.

5.2.3.1 Motor_test_control package

This package was the first demo developed with the purpose to test motor movement, while at the same time have a useful and practical application for a user-controlled mobile robot.

In this program, the robot movement is controlled using a keyboard with the following keys corresponding to these respective movements:

- w -> forward
- a -> left rotation
- s -> backward
- d -> right rotation
- any other key -> stop the robot

The node (called demo) reads the key pressed by the user on the terminal and publishes the corresponding motor control commands that the RPi-Ubo subscribes.

5.2.3.2 Camera_lidar_range_test package

This package corresponds to the second demo created and it has the objective to test the camera and to use image processing nodes. It uses a camera-based virtual LiDAR to measure the distance of near objects at the front and back of the robot.

The virtual LiDAR is generated by an image processing node (simulated_lidar) executed on the fisheye camera package (fisheye_camera_simulated_sensors_ros2). This node publishes the beams of the simulated LiDAR containing the distance measured by each one. The demo subscribes to this node and prints on a terminal the distances measured by the frontal and rear laser beam so it can be visible to the user.

5.2.3.3 Reactive_robot_demo package

The last demo is a simple autonomous application of a reactive robot³. The application is a simple obstacle avoidance example.

The robot moves forward and when an object is near him and there is a collision risk, the robot rotates to the left until the object is not interfering anymore. Having a free path, the robot resumes its rectilinear motion.

This demo uses the same virtual LiDAR used by the previous demo, but now it is only using the frontal laser measurements to detect objects. It checks the distance measured identifying whether there is an object near in front of the robot or not. Evaluating the result, it is decided what command should be sent to the motors. The command is then published. This demo also uses the RPi-Uno package node for communication and motor control.

5.3 Arduino Code

The code for the Arduino Uno was supplied by Professor Paulo Costa, the one also responsible for the development of the original physical robot. This code can be found on the GitHub Repository¹ along with the developed ROS2 packages. This code has essentially two functions:

- Low-level control of the motors. It reads the encoder and current of each motor, while at the same time it has logic to better control the requested motor speed.
- Serial communication. The Uno communicates with the Raspberry Pi via USB exchanging information with each other.

The Arduino program executing communicates with the previously mentioned "RPi-Uno" ROS2 package. The current information gathered by the motor driver shield board and the odometry values are sent to the RPi, while the motor speed commands are received by the Arduino. These messages are sent via serial communication through USB port.

This communication is done using specific channels. Each channel identifies what the message information is for. Table 5.1 shows the association between the channel letter and their meaning.

³<https://www.igi-global.com/dictionary/swarm-robotics/24606>

Note that the channel letters are case-sensitive. Another particularity is that all data values are sent/received in hexadecimal so, it is required to convert the data of the message before it is ready to be used. This is due to the channel form of communication, to avoid having actual numbers being read as letters. All messages have the same structure of 1 byte for the channel followed by 4 bytes corresponding to the data value.

Table 5.1: Channels and their respective function

Channel	Purpose
L	Command left motor speed
R	Command right motor speed
i	Current of the left motor
j	Current of the right motor
l	Encoder value of the left motor
r	Encoder value of the right motor

Regarding the odometry of each motor, what is actually sent by the Arduino program is the number of encoder triggers in a certain direction. A full motor wheel rotation corresponds to 2797 counts⁴. Also, this value increments or decrements according to the motor rotation direction.

5.4 Summary

This chapter focused on the software part of the system, listing all the necessary tools and explaining how the developed code and programs are organized. It explains ROS developments, mainly on the ROS1 code migration to ROS2 and on the ROS2 packages and applications explanation.

Lastly, it is explained the functioning of the Arduino program and how it communicates with the Raspberry Pi.

⁴https://wiki.dfrobot.com/12V_DC_Motor_251rpm_w_Encoder_SKU__FIT0186_

Chapter 6

Results and Discussions

6.1 Physical robot discussion

The 3D printed camera and RPi support is physically stable, but the mounting was not perfect, due to the handmade screw holes not being perfectly aligned. The same can be said for the LiDAR and its data and power board. However, these imperfections do not directly interfere with the robot's functionality. The worst-case scenario is a slight inclination in the camera/LiDAR, leading to small measurement errors, but since the purpose of the robot is to be a learning platform and not a high precision system, it is not relevant.

An important aspect to be discussed is the price. In fact, the total robot cost is higher than expected, but this is a consequence of not choosing most of the components of the system. However, the LiDAR is meant to be removed after being used for the camera calibration, with the objective to reduce the overall cost of the machine and achieve the minimalist setup pretended. Despite that, its cost must be presented. Also, the specifications of some pieces utilized are above what would be necessary for a low-cost machine for a beginner and intermediate level of robotics and ROS teaching. For example, the capacity of the battery could be smaller, leading to cheaper alternatives. Also, the motors could be less powerful, with the possibility to use motors driver shields with lower specifications and therefore, cheaper.

With the new RPi4 and with the unexpected and late connection of the LiDAR to the power output of the step-down converter, this component might exceed its power limits. According to the converter information¹, the maximum power output is 15Watts(W), and when the power is above 10W it is recommended to attach a heat sink. Looking at the RPi4 benchmarks², this computer on load consumes about 8W of power, but it can be higher depending on what it is doing and what it is connected through its I/O. Lastly, checking the LiDAR datasheet³ the minimum start-up peak current is at least 1A or 5W of power, and that is more power than the Raspberry Pi USB port allows to draw. Adding all this up, there might be some times when the power draw is superior

¹<https://www.ptrobotics.com/alimentacao/2647-modulo-conversor-dc-dc-step-down-lm2596.html>

²<https://medium.com/@ghalfacree/benchmarking-the-raspberry-pi-4-73e5afbcd54b>

³https://www.ydlidar.com/service_support/download.html?gid=5

to 10W, and might go even closer to the 15W if an unexpected power peak happens. Using this current configuration for long periods of time might shorten the lifespan of the converter.

6.2 Software and applications development

This section is to discuss some results and problems associated with software and with the demo applications created for the robot. Note that the GitHub page where the developed application files are located contains photos and videos on the "ReadMe" file about the demos. Appendix D is a copy of this tutorial page, but to better visualize the robot working and doing the desired tasks, it is suggested to consult the repository⁴. It has videos of all three demos developed and those can not be visualized in the appendix.

6.2.1 User controlled robot demo

This demo along with the GitHub tutorial introduces some basic ROS2 concepts. It teaches how to run nodes with the ROS2 "run" command, explains how to visualize the running nodes and topics, and shows how to visualize topic messages by using the "topic echo" ROS2 command. These are simple concepts but important to know to better understand how the software system is built and what are some interactions. These can also be seen as simple debugger tools since they show if everything is running properly, removing the barrier between bugs on the code itself or something that is missing to run, for example. This demo also allows for a better understanding of how the robot works and how ROS2 interacts with it.

Regarding the execution of the application itself, it is possible to visualize that the robot is responsive to the commands introduced. To see a video example of this demo, the GitHub repository has one in the "ReadMe" file⁴.

Since this demo communicates with the Arduino Uno to control the motors, it is possible to verify if the messages are being well received and published by the node responsible for that (the "serial_com" node of the "RPi-Unc package"). Figure 6.1 shows two terminals printing the messages published by this package node. Figure 6.1a corresponds to the current draw by each motor, and 6.1b is the Odometry value of each motor sent by the Arduino (Note that this value is only precise if the wheels do not spin with the robot on the same place).

6.2.2 Distance measure demo

Regarding the second demo tutorial, this introduces a new command, the ROS2 "launch". This in comparison to the "run" ones, demonstrates that it is possible to simplify and reduce the number of commands and terminals open required to execute a full application. Now, to fully start a demo it is only needed one terminal and one command.

Evaluating the second demo (the one that prints the distances of objects at the front and back of the robot), the distances measured are not very precise and they do not translate to the actual

⁴https://github.com/AlexMiranda16/Physical_Robot_Introduction_ROS2

```
data: i5 j4
...
data: i4 j4
...
data: i4 j3
...
data: i3 j4
...
data: i4 j3
...
data: i3 j3
...
data: i4 j1
...
data: i4 j3
...
```

(a) Current draw from each motor

```
data: l62940 r2601
...
data: l62857 r2684
...
data: l62778 r2756
...
data: l62699 r2831
...
data: l62620 r2912
...
data: l62540 r2995
...
data: l62457 r3076
...
data: l62375 r3156
```

(b) Odometry of each motor

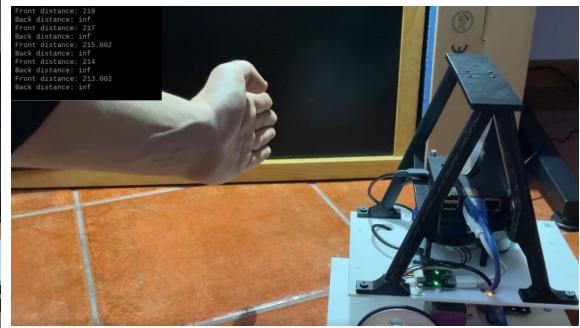
Figure 6.1: Messages received from the Arduino and published by the [RPi-Uno](#) communication package

distance of the objects. Since the distances measured are based on the object pixel location in the image, objects at the same relative distance to the robot but with different heights might retrieve different measures.

Figure 6.2 makes this problem clear, where 6.2a represents the detection and measurement of the relatively low position of the hand, retrieving the value of around 148, and 6.2b shows the detection of the hand at approximately the same distance to the robot but raised in comparison to the first example, calculating measurements of about 215.



(a) Distance measured with hand low



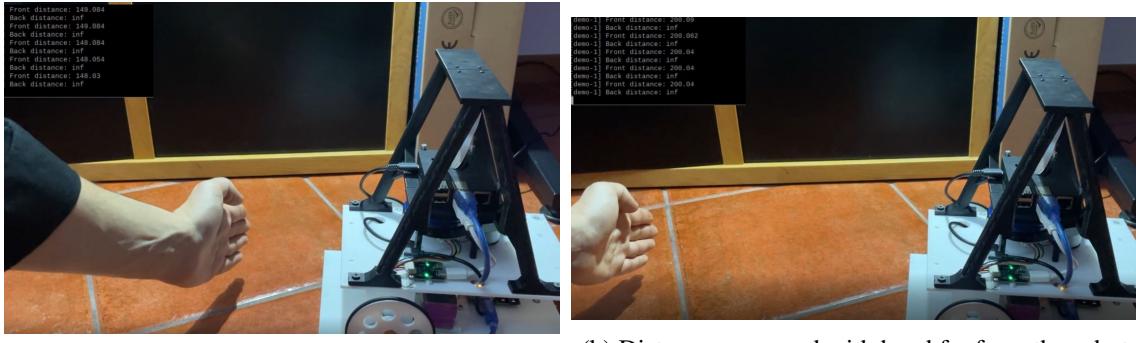
(b) Distance measured with hand high

Figure 6.2: Distance to the hand measured by the robot when it is at the same distance but with the hand in lower position - (a); and upper position - (b)

Also, the camera was not calibrated, and that leads to imprecise measurements too.

However, this lack of precision does not affect the functionality of the robot by much. It is still possible to have a notion of distances if the robot is used in a more controlled environment. Instead of looking at the values as absolute ones, they should be looked as relative, meaning that the higher the number, the further the objects are. Figure 6.3 shows an example of two measurement situations, where the closer hand retrieves a lower number (6.3a), while the further hand gives an higher one (6.3b).

Since this robot is more oriented to beginners in [ROS](#) and/or robotics, an high precision system is not mandatory. As the next demo proves (the object avoidance application), it is still possible to create and develop programs without a very precise machine.



(a) Distance measured with hand close to the robot

(b) Distance measured with hand far from the robot

Figure 6.3: Distance to the hand measured by the robot when the hand is close - (a); and when it is far - (b)

6.2.3 Reactive robot demo

Just as discussed in the previous demo, this application (reactive robot avoiding obstacles) uses the same distance indicative values measured from the camera virtual sensor.

Also, due to the simplicity of this program, because it is only using a single laser beam from the virtual **LiDAR** created by the camera, objects that are not directly in front of the robot, but are in its trajectory, might not be detected. The same can happen to very small, or very big objects, especially if they are very uniform in shape and/or color. That is due to the implemented image processing algorithm used (and the demo only using a single beam). This algorithm is based on the identification of edges of the objects, and if no edge is identified on the **LiDAR** laser direction, for the demo there is no obstacle identified.

However, this did not stop from creating a functional practical application. By using objects that the robot has little trouble detecting (with colors that do not blend in with the surrounding environment, and not being bigger than the robot to avoid vision obstruction), and by using the robot in a controlled environment where it will not appear unexpected objects, it is possible to create functional programs. To see a video example of this demo working, the GitHub repository has one in the "ReadMe" file⁴.

6.2.4 **LiDAR**

Unfortunately, the code for this device is not working, and therefore, it is not used in any of the developed applications. The code possibly has some bugs because, despite passing all device verification steps returning that it is running fine, and saying that the scan is about to start, this scanning process does not start and returns an "Operation time out" error, not being able to get the **LiDAR** data. The messages printed by the **LiDAR** on the terminal can be seen on the figure 6.4.

However, despite this device not being operational due to code problems, this does not have major consequences on the possibility of creating and developing programs for the robot. The initial objective of the **LiDAR** was to calibrate the camera, increasing its precision. So, the biggest disadvantage of not having this device is not having a system with higher precision. It is obvious

that with that it is not possible to create applications using high precision, but that did not stopped the development of vision system base functions with the robot, as previously seen with the demos created.

```
pi@raspberrypi:~/dev_ws/src/DissertationRobot/ydlidar_ros2_driver/launch $ ros2 launch X4.launch.xml
[INFO] [launch]: All log files can be found below /home/pi/.ros/log/2022-09-17-14-58-22-586001-raspberrypi-6143
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [ydlidar_ros2_driver-1]: process started with pid [6148]
[ydlidar_ros2_driver-1] [INFO] [1663423105.324190777] [ydlidar_lidar_publisher]: YDLIDAR ROS Driver Version: 1.0.2
[ydlidar_ros2_driver-1] [ERROR] [1663423105.333091542] [ydlidar_lidar_publisher]:
[ydlidar_ros2_driver-1]
[ydlidar_ros2_driver-1] [ERROR] [1663423105.333234687] [ydlidar_lidar_publisher]:
[ydlidar_ros2_driver-1]
[ydlidar_ros2_driver-1] YDLidar SDK initializing
[ydlidar_ros2_driver-1] YDLidar SDK has been initialized
[ydlidar_ros2_driver-1] [YDLIDAR]:SDK Version: 1.1.1
[ydlidar_ros2_driver-1] LiDAR successfully connected
[ydlidar_ros2_driver-1] [YDLIDAR]:Lidar running correctly ! The health status: good
[ydlidar_ros2_driver-1] [YDLIDAR] Connection established in [/dev/ttyYDLidar][128000]:
[ydlidar_ros2_driver-1] Firmware version: 1.4
[ydlidar_ros2_driver-1] Hardware version: 1
[ydlidar_ros2_driver-1] Model: X4
[ydlidar_ros2_driver-1] Serial: 2018052200000055
[ydlidar_ros2_driver-1] LiDAR init success, Elapsed time 674 ms
[ydlidar_ros2_driver-1] [CYdLidar] Successed to start scan mode, Elapsed time 1066 ms
[ydlidar_ros2_driver-1] [YDLIDAR] Fixed Size: 505
[ydlidar_ros2_driver-1] [YDLIDAR] Sample Rate: 5K
[ydlidar_ros2_driver-1] [YDLIDAR INFO1] Current Sampling Rate : 5K
[ydlidar_ros2_driver-1] [YDLIDAR INFO] Now YDLIDAR is scanning .....
[ydlidar_ros2_driver-1] timeout count: 1
[ydlidar_ros2_driver-1] [YDLIDAR ERROR]: Operation timed out
[ydlidar_ros2_driver-1] [ERROR] [1663423109.075913197] [ydlidar_ros2_driver]: Failed to get Lidar Data
[ydlidar_ros2_driver-1] timeout count: 2
[ydlidar_ros2_driver-1] [YDLIDAR ERROR]: Device Failed
[ydlidar_ros2_driver-1] [ERROR] [1663423111.076551495] [ydlidar_ros2_driver]: Failed to get Lidar Data
[ydlidar_ros2_driver-1] timeout count: 1
```

Figure 6.4: Terminal printed logs of the [LiDAR](#) program

6.3 Summary

Chapter 6 discussed some observations regarding the developed work on the physical robot.

It also discussed the software developments, mainly the demos created by analyzing some of its characteristics. It also explained what are the learning outcomes of the tutorial created for the robot.

Chapter 7

Conclusions and Future Work

7.1 Main contributions

With the work on this dissertation, it was possible to accomplish the initial objectives defined. However, these can always be improved and, in the following section, some suggestions are given.

The work of this dissertation had three main contributions: The questionnaire results, the physical robot, and the robot tutorial.

The questionnaire results contributed to a clearer notion in robotics and **ROS** teaching. Even though this form had a small number of responses, it allowed for a clearer picture of what people with experience in robotics or in both versions of **ROS** find more important and valuable in teaching these subjects. These gathered answers provide some guidelines to help give an overall better learning experience to students.

Another contribution was the adaptation of one robot so it can be a more capable **ROS**/robotics teaching platform. The developed demos not only demonstrate some potentialities of the created machine but also serves as validation that the robot is actually capable to be used in **ROS** teaching with the ability to perform different tasks.

Lastly, the GitHub tutorial creation contributed to a formal introduction and explanation on how to use the developed robot, with a step-by-step guide on how to start all the demos created, while at the same time, it teaches some **ROS2** basic concepts and commands.

7.2 Future work

7.2.1 Physical platform

Most of the recommendations presented here are not mandatory but would improve slightly the robot's design and functionality. These are suggestions that were noted after the build was done and resulted from the discussion previously made in chapter 6.

A new 3D printable design of the robot's upper base can be made with the addition of screw holes for the camera support and to accommodate the new layout of the components. The holes for the cables can also be in the design.

Although the camera support does not appear to have any problem, at least from what was used during this dissertation, the design can be printed with thinner cylindrical legs to decrease the [LiDAR](#) vision obstruction. The cables can also be routed around the camera support legs for the same reasons.

Since the primary objective was to get a low-cost mobile robot, a new evaluation of all components can be made, with the possibility of finding cheaper alternatives, with specifications sufficient to meet the necessary requirements. Of course, this new material list would probably require a redesign of the robot.

Regarding components, the step-down converter problems referred on the physical robot discussions in chapter 6 can be solved by replacing it with one with higher power output capability, or by adding a second one equal to the step-down already used, and distributing the components among the converters. One of these options is preferred over adding the heat sink, to ensure that the output power draw is always below the maximum allowed. Note that if the 2 converters solution is used, using a heat sink is still recommended.

7.2.2 Software and program development

Regarding the code, this can always be improved. There is always the possibility to add new features and create entirely new applications or improve some already existing either by performance or precision. For example, in the reactive robot demo to avoid objects, it is possible to make it use more information sent by the simulated [LiDAR](#) and make it aware of objects in more directions.

Despite that, there are still some suggestions to be made.

An important step to be done is the camera calibration. This calibration allows reducing precision errors in object detection and distance measurement that are associated with image distortion. There is also interest in making the [LiDAR](#) work by fixing the bugs presented in its current code. With that, it is possible to improve the camera precision by using the [LiDAR](#) distance information. Making the [LiDAR](#) functional also allows the creation of other programs and applications.

Although [ROS1](#) is reaching its End of Life, there is still interest in teaching and learning this older version, so it still makes sense to convert the codes created in [ROS2](#) to [ROS1](#), and reuse the codes that were originally made for [ROS1](#).

7.2.3 Simulator

A useful tool for robotics teaching is the simulator. This simulator would not replace the physical robot but rather act as a support tool for application development. A simulator allows to work and test programs being developed without the risk of damaging the real robot. It also eases and speeds the work process, not being dependent on the robot's availability. Another advantage is the easy access to a simulator. It is a cheaper solution than building a robot for every single person or student that will use it. So, everyone can develop their own applications without too many restrictions, and when implemented and tested in the simulator, they can pass to the real robot testing.

Appendix A

Questionnaire Form

Teaching of robotics and ROS1/ROS2

Faculty of Engineering of the University of Porto

This questionnaire has the purpose to gather information about robotics teaching and ROS1/ROS2 from the community with some experience in these fields.

There are no correct or incorrect answers.

The data collected in this questionnaire will be processed anonymously, and no personal data will be published or shared.

The data received from these questions will be used for statistical purposes only, being presented anonymously in the reports that will feature them.

All of the collected data will not be used outside of the scope of this study.

Thank you for your cooperation :)



*Obrigatório

In which faculties/universities do you teach robotics?

- Universidade do Porto (UP)
- Instituto Politécnico do Porto (IPP)
- Universidade de Aveiro (UA)
- Universidade de Coimbra (UC)
- Universidade do Minho (UM)
- Universidade de Coruña (UDC)
- Outra:



In which course(s) do you teach robotics?

- Electrical Engineering
- Informatics Engineering
- Mechanical Engineering
- Artificial Intelligence
- Outra:

At what level do you teach robotics?

- High School
- Bsc.
- Msc.
- Phd.
- Outra:



Please classify the level of agreement with each statement: *

	1- Totally disagree	2- Partially disagree	3- Neither disagree nor agree	4- Partially agree	5- Totally agree
I am proficient in the use of ROS1	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I am proficient in the use of ROS2	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I have teaching experience in ROS1	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I have teaching experience in ROS2	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Is teaching robotics with ROS important? Why?

A sua resposta

Is teaching ROS2 relevant? Why?

A sua resposta

Do you consider important to work with real robots when teaching robotics? When teaching ROS? Why for robotics and why for ROS?

A sua resposta



Between C++ and Python, which one do you consider more important to teach ROS with? Why?

A sua resposta

Página 1 de 3

Seguinte

Limpar formulário

Nunca envie palavras-passe através dos Google Forms.

Este formulário foi criado dentro de Universidade do Porto. [Denunciar abuso](#)

Google Formulários



Teaching of robotics and ROS1/ROS2



Robotics/ROS course contents

The book "A gentle Introduction to ROS", by Jason M. O'Kane, is often used as introductory material. Here (in 2 images for a better visualization) is presented the Contents in Brief of the book (if you want to access the book, it is available through this link: <https://www.cse.sc.edu/~jokane/agitr/>)

Contents in Brief

Contents in Brief	iii
Contents	v
1 Introduction	1
<i>In which we introduce ROS, describe how it can be useful, and preview the remainder of the book.</i>	
2 Getting started	11
<i>In which we install ROS, introduce some basic ROS concepts, and interact with a working ROS system.</i>	
3 Writing ROS programs	39
<i>In which we write ROS programs to publish and subscribe to messages.</i>	
4 Log messages	61
<i>In which we generate and view log messages.</i>	
5 Graph resource names	77
<i>In which we learn how ROS resolves the names of nodes, topics, parameters, and services.</i>	

iii



6 Launch files	83
<i>In which we configure and run many nodes at once using launch files.</i>	
7 Parameters	105
<i>In which we configure nodes using parameters.</i>	
8 Services	117
<i>In which we call services and respond to service requests.</i>	
9 Recording and replaying messages	133
<i>In which we use bag files to record and replay messages.</i>	
10 Conclusion	141
<i>In which we preview some additional topics.</i>	
Index	145

Do you consider any topic(s) of the book irrelevant (or less relevant) to a first introduction to ROS? Which one(s)?

A sua resposta



Classify how important you think the following topics of ROS are for teaching of ROS at an introductory level.

	1- Not important at all	2	3	4	5- Very Important
IDEs for ROS (ex: VSCode)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Utilization of GDB (attach to process)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Action servers/clients	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
TF	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Rviz	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Simulation 2D/3D	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Runtime parameters	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Besides the topics of the previous question, which topics do you consider the book is lacking?

A sua resposta

Do you have any suggestions about books, guides or videos for ROS?

A sua resposta



Teaching of robotics and ROS1/ROS2



Personal info (optional)

Optionally insert your name here:

A sua resposta

Optionally insert your e-mail here, if you wish to receive a summary of this survey

A sua resposta

Página 3 de 3

Anterior

Enviar

Limpar formulário

Nunca envie palavras-passe através dos Google Forms.

Este formulário foi criado dentro de Universidade do Porto. [Denunciar abuso](#)

Google Formulários



Appendix B

Software Installation and Environment Configuration

To fully prepare the system to run all the required software and to execute the desired functions, a series of installation and configuration steps were made. This guide has the purpose to detail the procedures to replicate the Raspberry Pi configuration done in this dissertation.

Before starting the explanation of the steps for each installed tool, it is important to know how to create disk backups. Backing up is particularly useful because the installation process takes some time and, in case of a need to restore the Operating System or if it is pretended to replicate multiple robots, restoring the backup created is much less time-consuming.

This online guide¹ explains very well how to do both the backup and the restore. The created backup should be for the whole disk, as the guide implies, and not only for a partition. There is a risk of the restoration not working if the image is only a partition.

B.1 Operating system installation

The official Operating System supported for the RPi is the [Raspberry Pi OS²](#). To install this Operating System the official website provides the Raspberry Pi Imager, an easy-to-use software to install the OS in just a few steps. It just needs to select the Operating System version to install, select the target drive where to install, and click on write. Note that with the imager, all the existing data will be overwritten, and therefore, lost.

The OS version used was the Debian Buster since it was the version used by the online guide followed to install ROS2 (more on this guide on the ROS2 Foxy Fitzroy installation section). Earlier versions are not supported by the RPi⁴ so, if an older version is installed, which was the case of the original robot (with Debian Stretch), it will not boot. It is possible to upgrade the installed version to a more recent OS, however, a clean installation is preferable because of its simplicity, speed and stability.

¹<https://blog.agungpratama.id/post/2020-02-05-how-to-backup-and-clone-your-raspberry-pi-image/>

²<https://www.raspberrypi.com/software/>

As previously referred, it is recommended to use a 32GB Micro-SD card since the software used occupies more than 16GB.

B.2 ROS1 Noetic Ninjemys Installation

To install this distribution of **ROS**³, [this guide](#) explains all the steps required to successfully do it on the [Raspberry OS](#).

From the authors of the guide, there are multiple variants of the noetic version. The one installed was ros_comm because of the unlikely success in installing the others, as the guide explains.

Despite recommending expanding the swap memory size, it was not done for this dissertation. The [4GB of RAM](#) available on the [RPi4](#) should be enough for the kind of applications used in the robot. If memory problems start to appear, it is suggested to do these steps.

B.3 ROS2 Foxy Fitzroy installation

Installing **ROS2** in a Raspberry Pi with [Raspberry Pi OS](#) is not as straightforward as in an Ubuntu machine. Some packages are not supported and therefore, will not build properly. So, an installation guide⁴ was followed to perform a correct installation of **ROS2**. This guide works as a complement to the official installation guide⁵. The installed distribution was the same as the guide, [ROS2 Foxy Fitzroy](#).

However, despite the tutorial, problems still surged when installing **ROS2**. The following is a list of recommendations and solutions to problems that arose during the installation for this dissertation:

- On the official installation guide, when doing the "Install development tools and ROS tools" step, if the following error appears "Malformed entry 1 in list file /etc/apt/sources.list.d/ros2.list (Component)", it is needed to run “sudo rm /etc/apt/sources.list.d/ros2.list”⁶ command to fix the problem.
- When running the commands on "Install developing tools and ROS tools" from the official installation guide, it might result in an "unable to locate package python3-colcon-common-extensions" and “unable to locate package python3-vcstool” error. The resolution to this problem consists in installing previously the python colcon extension. This procedure can be consulted [here](#)⁷.

³<https://varhowto.com/install-ros-noetic-raspberry-pi-4/>

⁴<https://medium.com/swlh/raspberry-pi-ros-2-camera-eef8f8b94304>

⁵<https://docs.ros.org/en/foxy/Installation/Alternatives/Ubuntu-Development-Setup.html>

⁶<https://askubuntu.com/questions/1168520/how-i-can-fix-the-error-malformed-entry-1-in-list-file-etc>

⁷<https://colcon.readthedocs.io/en/released/user/installation.html>

- At the web guide followed to install ROS2 there is a reference to a build error on the "mimick_vendor" package. This error did not happen during the installation so the procedures to solve it were ignored.
- Every time a new terminal is opened, the ROS2 setup files have to be sourced. To avoid this, the sourcing can be added to the shell startup script⁸. Running this command “echo “source ~/ros2_foxy/install/local_setup.bash” >> ~/.bashrc” in a terminal will add the sourcing and automatically source the setup files when opening a terminal.

To properly use the camera and process the image gathered by it, additional packages need to be installed. The guide followed to install ROS2 can be used to install some of those packages as well since it shows how to get "vision_opencv", "cv_bridge", "image_transport", among others. However, the branches of some repositories are not updated so, to get the correct packages, checking GitHub is important. For example, the branch name on the guide for the "vision_opencv" package is incorrect and should be replaced to clone a branch called foxy.

The last step is to configure the RaspiCam, an API for using the RPi camera with OpenCV. The GitHub repository can be found [here](#)⁹ with the compiling information on the ReadMe file.

B.4 Arduino IDE installation

Installing the Arduino IDE does not require too many steps. [This guide](#)¹⁰ explains everything needed to do to successfully install the IDE.

This software was used for the original robot, and code for the Arduino UNO board was developed. This code was reused for this dissertation because the UNO continues to communicate with the Raspberry Pi and still has the same function as before, that is, to control the motors.

To use the available code, other libraries need to be included in Arduino IDE fresh install. From the downloaded folder, all the content inside the "libraries" folder goes to the Arduino IDE libraries folder (usually it is inside of "arduino-ide_version" folder). The downloaded "MC33926" folder contains the code used to control the robot.

B.5 Visual Studio Code installation

The installation and configuration of the VSCode was supported by a video tutorial¹¹.

In the official VSCode site, it is possible to download an arm version for the Debian distribution. After downloading and installing the IDE it is time to install extensions to better support ROS packages and files. The extensions installed are "C/C++ IntelliSense (ms-vscode.cpptools)", "Python (ms-python.python)" and "CMake(twxs.cmake)". These extensions will auto-install some

⁸<https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Configuring-ROS2-Environment.html>

⁹<https://github.com/cedricve/raspicam>

¹⁰<https://www.raspberrypi-spy.co.uk/2020/12/install-arduino-ide-on-raspberry-pi/>

¹¹https://www.youtube.com/watch?v=RXYFSnjMd7M&ab_channel=LucasSchulze

more. Finally, it is needed to change the configuration.json and add the path in the "includepath" section to source the packages both from the source workspace, and the working workspace.

B.6 Define fixed tty devices

When a device connects to the Raspberry Pi, the Operating System attributes a device identification port that can be used to connect to it. This type of attribution is dependent on what type of device is connected. For example, some **USB** devices are identified by a "ttyUSBX" where the "X" is a number that is not fixed. That means that the same device might not have the same number identifier every time it is reconnected to the **RPi**.

In code and programs where the connection is made using the "tty", it is required to successfully connect to it independently of the port number to avoid the need to always check which number is the identifier port for the device, and to change on the code the number.

B.6.1 **LiDAR** device identifier

The **LiDAR** is identified with "ttyUSBX" (X being a number) and to lock the port identifier, a tutorial¹² was followed. Doing the first command presented in the guide, the device is easily identified if no other devices are connected (if there are more devices, a trick to better get the correct one is to run the command before and after plugging the device). For the **LiDAR** it should be something similar to this: "Bus 001 Device 004: ID 10c4:ea60 Cygnal Integrated Products, Inc. CP2102/CP2109 UART Bridge Controller [CP210x family]".

With that, it is already possible to obtain the vendor and product identifier (**ID**). Those are 10c4 and ea60 respectively. However, it is required a deeper research to collect the serial number attribute. That is accomplished using part of the "udevadm" command, more precisely "udevadm info -a -n /dev/ttyUSBX". If the value of the "tty" port is unknown, running in a terminal the "ls /dev/" command displays various "tty", including the **USB** one with the number pretended (if there are multiple "ttyUSB" the trick previously mentioned allows to identify the correct number). This command prints multiple parent devices. The serial number can be obtained by looking for the parent device with the vendor and product **ID**. Figure B.1 shows the correct parent device to collect the information required. For easier identification of the valuable data, it was marked with a red circle.

Finally, the rule must be added to the file indicated on the guide, where "SYMLINK" corresponds to the name wished to attribute to the **LiDAR**. In this case, the given name was "ttyYDLidar".

¹²https://community.silabs.com/s/article/fixed-tty-device-assignments-in-linux-using-udev?language=en_US

```

ATTRS{bDeviceProtocol}=="0100"
ATTRS{ltm_capable}=="no"
ATTRS{product}=="CP2102 USB to UART Bridge Controller"
ATTRS{bDeviceProtocol}=="00"
ATTRS{bmAttributes}=="80"
ATTRS{bNumInterfaces}==" 1"
ATTRS{tx_lanes}=="1"
ATTRS{speed}=="12"
ATTRS{bNumConfigurations}=="1"
ATTRS{busnum}=="1"
ATTRS{bDeviceClass}=="00"
ATTRS{bConfigurationValue}=="1"
ATTRS{serial}=="0001"
ATTRS{configuration}==""
ATTRS{rx_lanes}=="1"
ATTRS{urbnum}=="16"
ATTRS{version}==" 1.10"
ATTRS{idProduct}=="ea60"
ATTRS{idVendor}=="10c4"

```

Figure B.1: Parent device with the vendor and product ID and serial number for the [LiDAR](#)

B.6.2 Arduino device Identifier

The Arduino Uno is another device that is connected to the [RPi](#) via [USB](#) cable. However, the Operating System identifies the Uno with "ttyACMX" (X is a number given by the [OS](#)). The steps for this device are very similar to the previous one and the same guide^{[12](#)} can be followed.

With the first command, it is expected to return a message similar to this one "Bus 001 Device 006: ID 2341:0043 Arduino SA Uno R3 (CDC ACM)". For this case, the vendor [ID](#) is 2341 and the product is 0043. Doing the same search for the serial number done for the [LiDAR](#), but now for "ttyACMX", will print a parent device with all the information displayed. The expected result is presented in figure B.2.

The last step involves adding the same rule on the same file mentioned in the guide. For the Arduino Uno, the name given was "ttyUNO".

```

looking at parent device '/devices/platform/scb/fd500000.pcie/pci0000:00/0
000:00:00.0:0000:01:00.0:usb1/1-1/1-1.1':
KERNELS=="1-1.1"
SUBSYSTEMS=="usb"
DRIVERS=="usb"
ATTRS{bMaxPacketSize0}=="8"
ATTRS{serial}=="55737313631351F03292"
ATTRS{devspec}==(null)
ATTRS{bNumConfigurations}=="1"
ATTRS{busnum}=="1"
ATTRS{quirks}=="0x0"
ATTRS{bmAttributes}=="c0"
ATTRS{bConfigurationValue}=="1"
ATTRS{maxchild}=="0"
ATTRS{bDeviceSubClass}=="00"
ATTRS{removable}=="unknown"
ATTRS{configuration}==""
ATTRS{idProduct}=="0043"
ATTRS{rx_lanes}=="1"
ATTRS{bDeviceProtocol}=="00"
ATTRS{bcdDevice}=="0001"
ATTRS{devpath}=="1-1"
ATTRS{idVendor}=="2341"
ATTRS{authorized}=="1"
ATTRS{manufacturer}=="Arduino (www.arduino.cc)"

```

Figure B.2: Parent device with the vendor and product ID and serial number for the Arduino

To check if the devices are connected, and the names assigned, running the command "ls /dev/" on a terminal will return the ports and they should appear just like figure B.3.

```
tty30  ttyACM0      vhci
tty31  ttyAMA0      video0
tty32  ttyprintk    video10
mem   tty33  ttyS0   video11
mem   tty34  ttyUNO  video12
mem   tty35  ttyUSB0  video13
em    tty36  ttyYDLidar video14
tty37  uhid        video15
tty38  uinput      video16
```

Figure B.3: New names assigned for the devices, written in blue

Appendix C

ROS Migration Guide

Some work involved in this dissertation was the migration of **ROS1** packages to **ROS2**. This guide details the procedures done to the camera and **LiDAR** packages that was initially written in **ROS1**.

To fully migrate a package, the official documentation¹ for **ROS2** has a guide explaining how to do it. However, some instructions are not very clear and information is spread around. Nevertheless, the reading of the guide is recommended, especially the example because it is more organized and clearer of what to modify.

The easiest way to migrate **ROS1** to **ROS2** is to first create a new **ROS2** package and then transfer the files one by one, or moving and changing small pieces of the files. The advantage of creating a new package is that some files, in particular, the "package.xml" and "CMakeLists.txt" require transformations that are already done in the **ROS2** standard format of a package, not being necessary to do so many alterations to these files.

Besides the creation of a new package to move the content pretended to transform, another new package should be created specifically for custom messages and services if the package to migrate contains these types of files. It is not mandatory to do that, but it is better for the organizational structure of multiple packages systems because if multiple packages use the same type of custom message they all depend on the same package with just the message, instead depending on a package with other purposes that also contains the message².

C.1 Migration of Custom Messages and Services

The official documentation³ for **ROS2** Foxy Fitzroy has a tutorial to learn how to create a custom message or service. However, this guide does not clarify some important aspects that can lead to errors if not properly handled.

When the custom message has a data type of another package, for example, geometry_msgs/msg/Point this package must be installed and its dependency must be added both on "package.xml"

¹<https://docs.ros.org/en/foxy/The-ROS2-Project/Contributing/Migration-Guide.html>

²<https://roboticsbackend.com/ros2-create-custom-message/>

³<https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Custom-ROS2-Interfaces.html>

and "CMakeLists.txt" of the message package. At "package.xml" the "<depend>package_name </depend>" must be added and in "CMakeLists.txt" the "find_package(package_name REQUIRED)" must be as well². Also, "DEPENDENCIES builtin_interfaces package_name" need to be added in "rosidl_generate_interfaces" from the CMakeLists.txt file⁴. If when building the custom message package an error occurs about a specific type of data on the message or service structure, it is most likely to be missing one of the dependencies just referred.

With the message created, the last step is to add to the packages where it is going to be used. This step will be referred later in this guide when explaining the modifications required to make to the other files.

Both packages migrated had a custom message structure created and no services, so, this guide only focused on the transformation of message files. These files went to the same package created specifically for messages called "custom_message". The recommendations previously referred were important to the [LiDAR](#) message, because the data structure contains a data type included in the "std_msgs" package.

C.2 Migration of Package.xml files

Since a new package was created, some of the alterations needed to be done in this file are already accomplished. These are the ones involving the substitution of "catkin" to "ament_cmake" dependencies. Note that most of the official migration¹ guide instructions for the custom messages and services are already done with the specific package created for that so, there is no need in adding the dependencies for "rosidl_(...)" that was already added in the custom_message package.

Therefore, for these files, all there is to do is adding the depend of the packages used by the [ROS1](#) version and the package used for the custom messages. However, some of the packages have different names due to the [ROS](#) version. That is the case of "roscpp" and "rospy" that in [ROS2](#) are called "rclcpp" and "rclpy" respectively. Also, the dependencies "message_runtime" and "message_generation" are not to be included since these are for custom messages and services for [ROS1](#).

Despite the alterations of the modified files, more dependencies were added that were missing from the original ones. It was the case of the "std_srvs" package dependency on the [LiDAR XML](#) file, and "cv_bridge", "image_transport" and "OpenCV" on the camera package [XML](#) file. In both packages it was also added the dependency of the custom message package, that is "custom_message".

C.3 Migration of CMakeLists.txt files

Again, with the creation of a new package some modifications on the guide¹ are already done, only needing to port some parts of the [ROS1](#) CMakeLists file to [ROS2](#). Also, most of the custom messages and services porting can be ignored again, since they are already done in the package

⁴<https://answers.ros.org/question/314724/ros2-embedding-a-msg-as-a-field-in-a-custom-msg/>

created with that purpose, only being required to include the dependency of the created custom message package.

The first step to do is adding the "find_package" command, separating each package and adding a "REQUIRED" in each one (`find_package(package_name REQUIRED)`). The "catkin" and "message_generation" packages are not to be included in **ROS2** since these are for the previous version of **ROS**.

The existing "include_directories" command for **ROS1** is changed to "target_include_directories", which already exists when creating a new package, so there is no need to add it. However, some arguments might be missing. The ones belonging to catkin are not meant to be added.

The "target_link_libraries" remains the same for **ROS2** so it can be copied directly from the original version, with the exception of the catkin libraries that are not used.

Finally, the "ament_target_dependencies" includes all other packages used by the source files.

For the **LiDAR** package, besides the packages already included in the **ROS1** version, the package "std_srvs" and "custom_message" were added. On the "target_link_libraries" and "target_include_directories" the **LiDAR SDK** arguments existing on the corresponding commands on the original files were included. Finally, the packages "rclcpp", "rclpy", "sensor_msgs" "std_srvs" and "custom_message" package were added on "ament_target_dependencies".

As for the camera package, the "PkgConfig" package on the original version was not included. Also, for this case, it was not required to set the OpenCV directory. On the other hand, for the raspicam, it was set (`set(raspicam_DIR "/usr/local/lib/cmake")`). The "custom_message" package was added as well.

Since the camera package contains multiple source files, it is required to add an executable for each one. Also, each one needs an "ament_target_dependencies", a "target_link_libraries" and a "target_include_directories", and these executable need to be included on the "install" command. The "target_link_libraries" were copied from the original file with the catkin libraries removed. For the "target_include_directories", the arguments existing on the corresponding commands on the original files were included. Lastly in each source file "ament_target_dependencies" it was included the packages they depended on.

C.4 Migration of source files

With a well configured **IDE** for the used **ROS2** distribution and with the previous modifications well done, most of the required changes to do in the source files will be marked as errors. Here it won't be explained every single error that occurred in the migration process due to the simplicity of most errors and the easy clarification given both by the example on the official migration guide¹ previously mentioned and by the basic guide to create a simple publisher and subscriber⁵. These guides were enough to migrate most of the code to **ROS2**.

⁵ <https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html>

The first thing to change is the included libraries on the header files, or on the source ones if the package does not contain them. Here, "ros/ros.h" is changed to "rclcpp/rclcpp.hpp", the msgs libraries types need a "msg" in the include (e.g.: sensor_msgs/msg/....hpp), the srvs libraries are equal as the msgs ones, expect with a "srv" (e.g.: std_srvs/srv/....hpp) and the custom message needs to be added where it is going to be used. The structure of the custom message include is important, "custom_message/msg/file_name.hpp"⁶.

On the code itself, most of the changes are related to the new structure or naming of the included packages. Some functions of these packages are different either by their arguments or by their syntax to achieve the same purposes as the original code intends. It is the case of the "ros" class that was replaced with "rclcpp" in ROS2 so, the commands on the original files with "ros" need to be replaced. The ones related to the creation of a node and a publisher/subscriber are easy to modify following the tutorials mentioned above^{1/5}. To get the current time originally obtained by the "ros::Time::now()" command, ROS2 uses the node itself for that (node_name->now())⁷.

It is important to refer that the object declaration type of the packages included, including the objects created for the publishers and services, changes slightly in comparison to the original code due to the different naming of the packages. In ROS2, the objects created from the package types included need to have the "msg" or "srv" just like in the header files.

The ROS2 parameters tutorial⁸ supported the migration of the parameters on the code. These were replaced with "node_name->get_parameter", with the arguments of the function being the same as the original command replaced. As for the parameters of the launch files, the next section will specify the alterations in more detail.

C.5 Migration of launch files

The official ROS2 Foxy documentation has a separate tutorial from the Migration guide specifically to explain how to migrate launch files⁹ from ROS1 to ROS2. The launch files on the packages to migrate are not too complex, so there is not much to change. On the "node" the "type" attribute is changed to "exec", on the "param" if there is a "type" attribute it is eliminated. Some files used the "arg" and despite existing in ROS2 as well, it was opted to not use and just use the "param" with the default values on the "arg" one.

⁶<https://stackoverflow.com/questions/68052476/ros2-colcon-build-could-not-generate-hpp-files>

⁷<https://answers.ros.org/question/287946/ros-2-time-handling/>

⁸<https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Using-Parameters-In-A-Class-CPP.html>

⁹<https://docs.ros.org/en/foxy/How-To-Guides/Launch-files-migration-guide.html>

Appendix D

Physical Robot Introduction using ROS2 (GitHub ReadMe)

USB main ▾

...

[Physical_Robot_Introduction_ROS2 / README.md](#)

AlexMiranda16 Update README.md

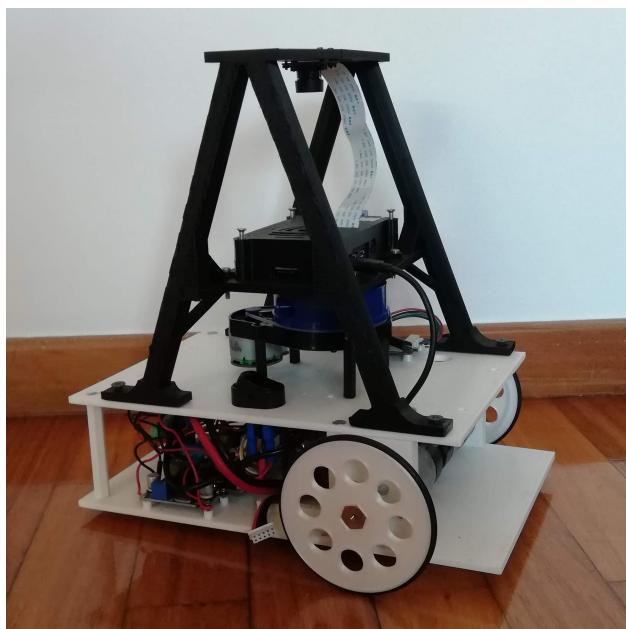


2 contributors



Physical_Robot_Introduction_ROS2

This repository contains three ROS2 demos to teach the user how to work with ROS2 to develop programs for physical robots. This tutorial will explain how to setup and run each demo for the robot depicted below. The objective of this guide is to give a first contact to the robot by running and testing these demos, while at the same time get more used to ROS2 environment. These programs were specifically made using this robot so, it is assumed that you are using one equal to the one shown on the image below. It is not guaranteed that these demos will work on a different robot.



The first demo tests the motors, controlling the robot using a keyboard, and checking if the motor current and encoders are published. The second demo tests if the camera and the image processing algorithm is properly working, by reading the distances measured at the front and back of the robot. The third and last demo is a simple reactive robot demo, where the robot uses the camera to check for objects at his front and avoids them.

These demos are independent from each other and, if you want to run one demo when another is running, you should stop the running one first before starting the demo wanted.

This repository was developed during Alexandre Miranda's master dissertation, at the Faculty of Engineering of the University of Porto (FEUP).

1. Prerequisites

One obvious prerequisite is to have at least one robot equal to the one used to develop these demos.

Regarding software, you need to install some tools, mainly:

- Raspbian OS (Debian Buster version)
- ROS2 Foxy Fitzroy (and some packages described below)
- Arduino IDE
- Visual Studio Code (optional)
- VNC Server (optional but recommended for remote access to the robot's Operating System, both for developing, running and debugging programs)

You will also need to configure device identification ports attributed by the OS.

For you to know how to install all the software and how to create fixed tty device ports, you can download the following PDF file. This guide provides additional information to possible installation and configuration problems.

[Software_Installation_Env_Config.pdf](#)

Besides the software, you need to create a new ROS2 workspace to place all the robot packages. In this example, the ROS2 workspace is "dev_ws" but you can give it another name.

Open a terminal and create a new directory. Then, go to the "src" directory and clone the repository. Check the dependencies of the workspace and then build it using the colcon command.

```
mkdir -p ~/dev_ws/src  
cd ~/dev_ws/src  
git clone https://github.com/AlexMiranda16/Physical_Robot_Introduction_ROS2  
cd ..  
rosdep install -i --from-path src --rosdistro foxy -y  
colcon build --cmake-args -DCMAKE_BUILD_TYPE=Release
```

As the instalation and configuration guide refers, you need to add additional packages to your workspace.

Download the following packages to the src folder of your workspace directory. When downloaded, compile the workspace again.

```
cd ~/dev_ws/src  
git clone --branch foxy https://github.com/ros-perception/vision_opencv.git  
git clone --branch foxy https://github.com/ros-perception/image_common.git  
git clone --branch ros2 https://github.com/ros-  
perception/image_transport_plugins.git  
cd ..  
rosdep install --from-paths src -r -y  
colcon build --cmake-args -DCMAKE_BUILD_TYPE=Release
```

The next step is to install the RaspiCam, an API for using the Raspberry Pi camera with OpenCV. The compilation process guide is explained on the ReadMe file of the RaspiCam repository, located here: <https://github.com/cedricve/raspicam>.

Do not forget that in every new terminal opened, you need to source the setup script of this workspace. Alternatively, you can add the source command to the .bashrc file in a similar way to that shown in section A.3 of the intallation guide.

```
source install/setup.bash
```

For more information about the ROS2 workspace creation, please visit "[Creating a workspace](#)".

2. Demo 1 - Keyboard robot motor control

Objective: Test the motors' operation.

The program controls the robot movement using the "wasd" keyboard keys:

- w - foward;
- a - left rotation;
- s - backward;

- d - right rotation;
- To stop the motors, any other key can be pressed.

If not done yet, use the Arduino IDE to upload the code to the Arduino board you are using (must be an ATMega328 microcontroller board).

Assuming the projects are already compiled (if not, use the colcon build command previously mentioned in your workspace directory), open two terminal windows and go to your workspace directory and source the script file in both of them.

```
cd ~/dev_ws/
source install/setup.bash
```

Now, in one terminal start the motor control program with the following command. It is in this terminal that you will insert the keyboard commands to move the robot.

```
ros2 run motor_test_control demo
```

On the other terminal, start the "serial_com" node from the "rpi_arduino" package. This package is responsible for the communication between the Raspberry Pi and the Arduino. This program subscribes to the topic that sends the movement command for each motor, and send it to the Arduino. This node also receives information from the Arduino, namely the motor current and the odometry, and publish it.

```
ros2 run rpi_arduino serial_com
```

The robot now should be ready to go!

To check if everything is running properly, with both nodes running, run the `ros2 node list` and `ros2 topic list` commands on one new terminal. You should see the following nodes and topics running.

The image shows two terminal windows side-by-side. The left terminal window has the title 'pi@raspberrypi: ~/dev_ws' and displays the command 'ros2 node list'. The output shows two nodes: '/Serial_com' and '/keyboard_robot_control'. The right terminal window also has the title 'pi@raspberrypi: ~/dev_ws' and displays the command 'ros2 topic list'. The output shows five topics: '/motor_current', '/motor_odometry', '/motor_speed', '/parameter_events', and '/rosout'.

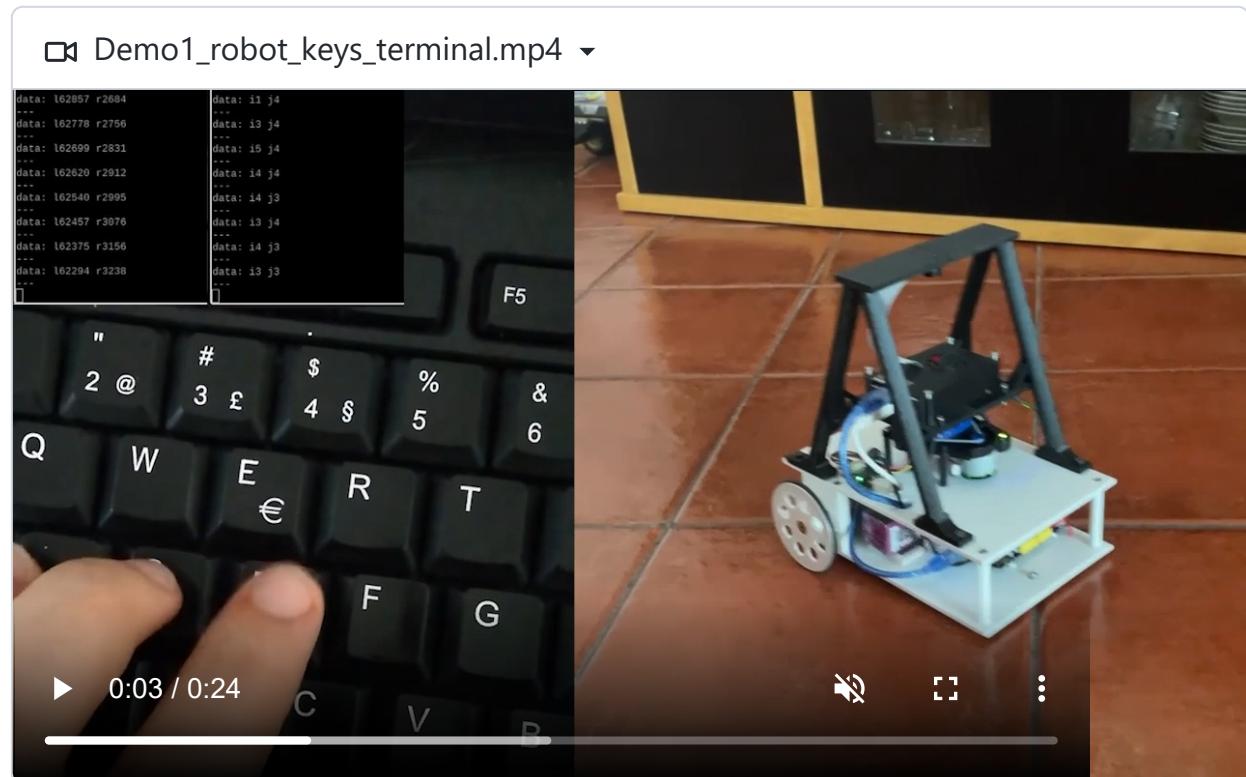
```
pi@raspberrypi:~/dev_ws $ ros2 node list
/Serial_com
/keyboard_robot_control
pi@raspberrypi:~/dev_ws $ 

pi@raspberrypi:~/dev_ws $ ros2 topic list
/motor_current
/motor_odometry
/motor_speed
/parameter_events
/rosout
pi@raspberrypi:~/dev_ws $
```

If you want to check if the "serial_com" node is publishing the current and odometry from both motors, you can use the `ros2 topic echo` command. In two new separate terminals, run:

```
ros2 topic echo /motor_current  
ros2 topic echo /motor_odometry
```

The following video shows an example on how the application should work. You can see on the top left the two terminals. The left one prints the odometry values received by the arduino, and the right one prints the current. You can also see how the robot should behave by pressing different letters. If you haven't done yet, give it a try!



Additional Notes:

All messages have a letter that works as an identification and a number that corresponds to the value we want to send or read. In this case, each letter represents a motor and the type of data we are sending/receiving, and the number translates its value. All messages have the same order: `left_motor&value right_motor&value` (first is the letter and value of the left motor, followed by a space and then the letter and value of the right motor). This next table clarifies the different letters meaning.

Channel	Meaning
L	Command left motor speed
R	Command right motor speed
i	Current of the left motor

Channel	Meaning
j	Current of the right motor
l	Encoder value of the left motor
r	Encoder value of the right motor

The odometry value received is actually the number of encoder triggers in a certain direction, giving the position of the motor. A full 360° rotation corresponds to 2797 counts. Also, the value increases or decreases according to the motor rotation direction.

For more information about the motor, you can visit its wiki page:

https://wiki.dfrobot.com/12V_DC_Motor_251rpm_w_Encoder_SKU_FIT0186_

3. Demo 2 - Camera based virtual LiDAR test

Objective: test the camera and the applied image processing algorithm.

This demo prints on the terminal the distance of the nearest object both at the front and at the rear of the robot.

If not done yet, use the Arduino IDE to upload the code to the Arduino board you are using (must be an ATMega328 microcontroller board).

Assuming the projects are already compiled (if not, use the colcon build command previously mentioned in your workspace directory), open two terminal windows and go to your workspace directory and source the script file in both of them.

```
cd ~/dev_ws/
source install/setup.bash
```

Now, in one terminal start the camera range test program with following command. This terminal will display the frontal and rear measurements.

```
ros2 run camera_lidar_range_test demo
```

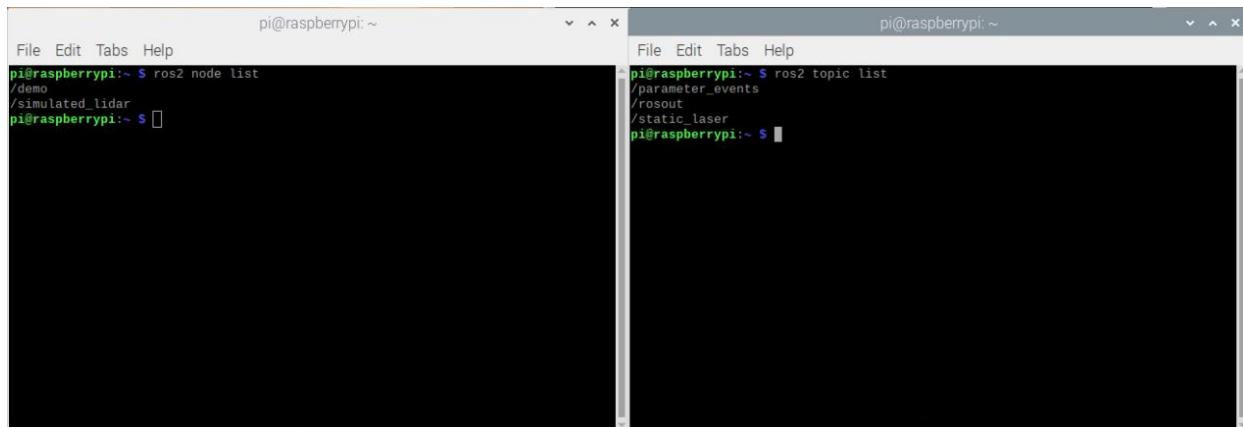
On the other terminal, run the camera simulated LiDAR. This node is the one responsible for gathering the camera image, do some processing on it and with that it simulates a LiDAR sensor, and publish the distances calculated around the robot.

```
ros2 run fisheye_camera_simulated_sensors_ros2 simulated_lidar
```

Instead of running both `ros2 run` to execute this application, it can be used the launch file created. Stop both nodes if they are still running and in one terminal run the next `ros2 launch` command:

```
ros2 launch camera_lidar_range_test range_demo.launch.xml
```

Again, to check if everything is running properly, with both nodes running, run the `ros2 node list` and `ros2 topic list` commands on one new terminal. You should see the following nodes and topics running.



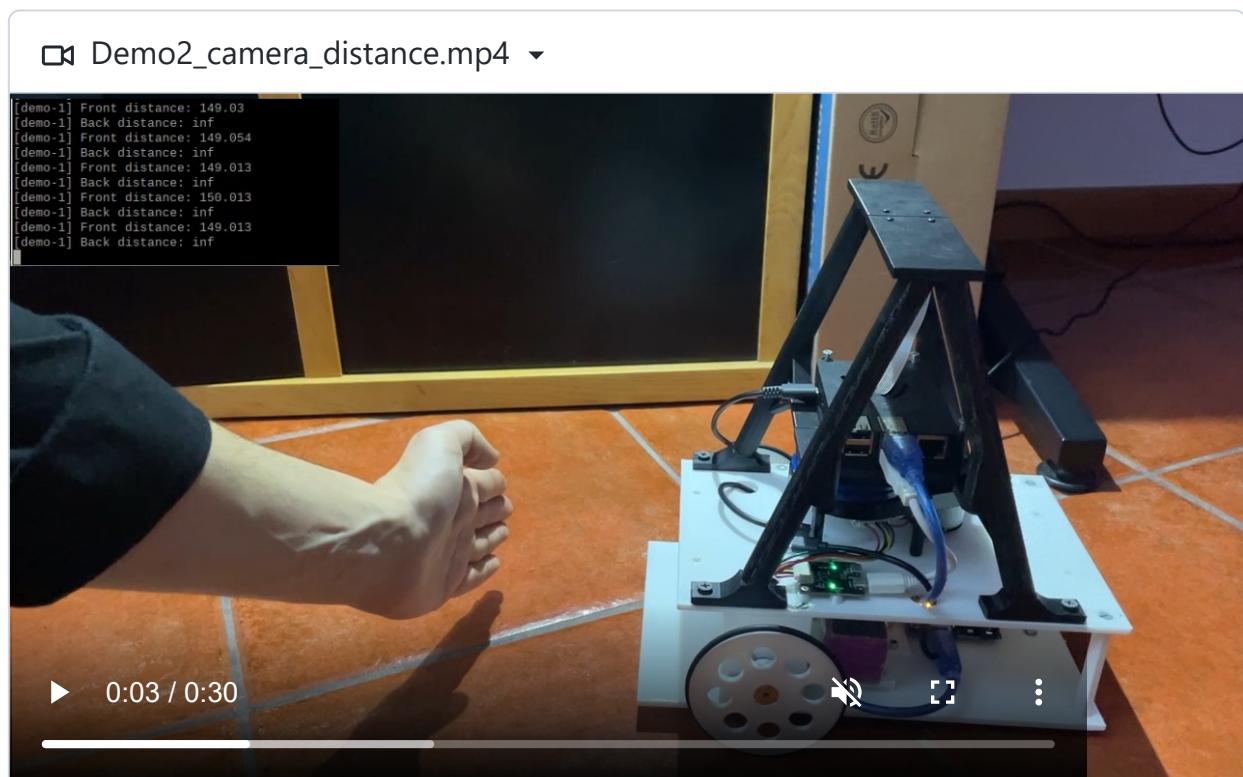
The image shows two side-by-side terminal windows on a Raspberry Pi. The left terminal window displays the output of the command `ros2 node list`, which shows two nodes: `/demo` and `/simulated_lidar`. The right terminal window displays the output of the command `ros2 topic list`, which shows three topics: `/parameter_events`, `/rosout`, and `/static_laser`.

If everything is working as pretended, the demo should behave similarly to the video shown. On the top left of the video you can see the distance measured both by the

224 lines (141 sloc) | 12.3 KB

...

the number, the further the objects are.



Additional Notes

Tall objects might give wrong measures. In the video you can see that the hand down and up at a similar distance give different measurements.

4. Demo 3 - Reactive robot demo

Objective: Run a simple reactive robot that avoids collision with objects.

The robot moves forward until a nearby object is identified in front of it. The robot then rotates to the left until the object is no longer in its front and proceeds to move forward in the new direction.

If not done yet, use the Arduino IDE to upload the code to the Arduino board you are using (must be an ATMega328 microcontroller board).

Assuming the projects are already compiled (if not, use the colcon build command previously mentioned in your workspace directory), open one terminal window and go to your workspace directory and source the script file.

```
cd ~/dev_ws/  
source install/setup.bash
```

Now, start the reactive robot program with the following `ros2 launch` command. This launch file will start three nodes:

- The reactive robot demo node. Evaluates the camera information and commands the movement for the robot to make.
- The simulated LiDAR node. As previously explained on the demo2 program, is responsible for the camera image processing, simulating a virtual LiDAR by calculating the distance of objects around the robot.
- The Raspberry Pi - Arduino communication node. Also explained previously. Ensures the communication between the Raspberry and the Arduino, by sending movement commands for each motor, and receiving the current and odometry of each one.

```
ros2 launch reactive_robot_demo reactive.launch.xml
```

In a couple of seconds, the robot should start moving by itself.

Again, to check if everything is running properly, with both nodes running, run the `ros2 node list` and `ros2 topic list` commands on one new terminal. You should see the following nodes and topics running.

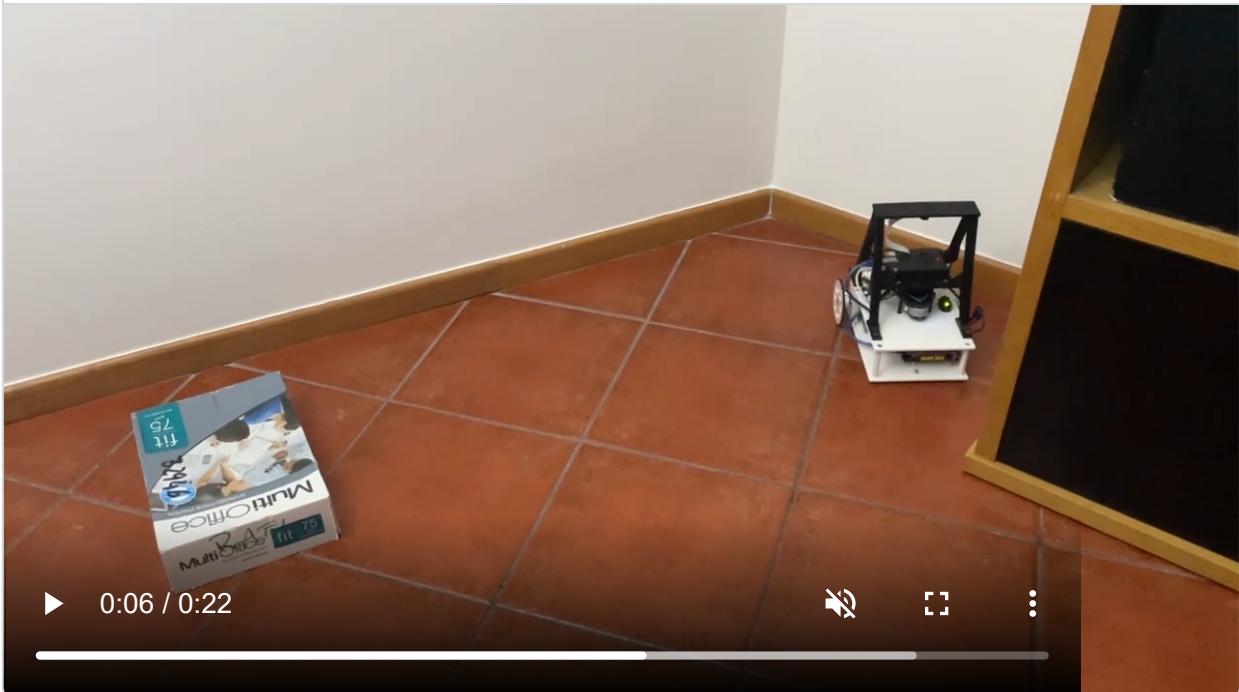
The image shows two terminal windows side-by-side. The left window displays the command `ros2 node list` with the output: `/demo`, `/serial_com`, and `/simulated_lidar`. The right window displays the command `ros2 topic list` with the output: `/motor_current`, `/motor_odometry`, `/motor_speed`, `/parameter_events`, `/rosout`, and `/static_laser`.

```
pi@raspberrypi:~ $ ros2 node list
/demo
/serial_com
/simulated_lidar
pi@raspberrypi:~ $ 

pi@raspberrypi:~ $ ros2 topic list
/motor_current
/motor_odometry
/motor_speed
/parameter_events
/rosout
/static_laser
pi@raspberrypi:~ $ 
```

If the robot is doing what it is supposed to do in this demo, you should see it behave similarly to this video. It will turn left to avoid obstacles when it is near them.

▢ Demo3_reactive_robot.mp4 ▾



References

- [1] J. Ferreira, F. Coelho, A. Sousa, and L.P. Reis. Bulbrobot – inexpensive open hardware and software robot featuring catadioptric vision and virtual sonars. *Advances in Intelligent Systems and Computing*, 1092 AISC:553–564, 2020. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85082111392&doi=10.1007%2f978-3-030-35990-4_45&partnerID=40&md5=59552fe8c6dcbf19cf472e7a598c6a9a, doi:10.1007/978-3-030-35990-4_45.
- [2] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Stephane Magnenat, Jean-Christophe Zufferey, Dario Floreano, and A. Martinoli. The e-puck, a robot designed for education in engineering. *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, 1, 01 2009.
- [3] L.F. Llamas, A. Paz-Lopez, A. Prieto, F. Orjales, and F. Bellas. Artificial intelligence teaching through embedded systems: A smartphone-based robot approach. *Advances in Intelligent Systems and Computing*, 1092 AISC:515–527, 2020. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85082141920&doi=10.1007%2f978-3-030-35990-4_42&partnerID=40&md5=1851fe510d7c56521865c94ff82c2ee8, doi:10.1007/978-3-030-35990-4_42.
- [4] K. Afsari and M. Saadeh. Artificial intelligence platform for low-cost robotics. 2020. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85101156220&doi=10.1109%2fICSPIS51252.2020.9340156&partnerID=40&md5=618bc5b89f77c93cfc3480b794dd0485>, doi:10.1109/ICSPIS51252.2020.9340156.
- [5] D. Kumar and L. Meeden. A robot laboratory for teaching artificial intelligence. volume 30, pages 341–344, 1998. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-0004877047&doi=10.1145%2f274790.274326&partnerID=40&md5=e8b8fd080f87f6d59348bf9940641269>, doi:10.1145/274790.274326.
- [6] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. volume 3, 01 2009.
- [7] F. Bellas, M. Naya, G. Varela, L. Llamas, M. Bautista, A. Prieto, and R.J. Duro. Robobo: The next generation of educational robot. *Advances in Intelligent Systems and Computing*, 694:359–369, 2018. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85042229481&doi=10.1007%2f978-3-319-70836-2_30&partnerID=40&md5=b61b77ac66ec91b5caed9ec0e5e2b191, doi:10.1007/978-3-319-70836-2_30.

- [8] 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 and Robotic Systems: Theory and Applications*, 94(3-4):761–775, 2019. URL: [https://www.scopus.com/inward/record.uri?eid=2-s2.0-85047239237&doi=10.1007%2fs10846-018-0866-9](https://www.scopus.com/inward/record.uri?eid=2-s2.0-85047239237&doi=10.1007%2fs10846-018-0866-9&partnerID=40&md5=b4da0ec639a668b405cd210ac622d910).
- [9] J.M. Soares, I. Navarro, and A. Martinoli. The khepera iv mobile robot: Performance evaluation, sensory data and software toolbox. *Advances in Intelligent Systems and Computing*, 417:767–781, 2016. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84951800623&doi=10.1007%2f978-3-319-27146-0_59&partnerID=40&md5=06ddd1376a1b82efb7f24e8be0b7872d, doi: 10.1007/978-3-319-27146-0_59.
- [10] Francesco Mondada, Michael Bonani, Fanny Riedo, Manon Briod, Lea Pereyre, Philippe Retornaz, and Stephane Magnenat. Bringing robotics to formal education: The thymio open-source hardware robot. *IEEE Robotics Automation Magazine*, 24(1):77–85, 2017. doi: 10.1109/MRA.2016.2636372.
- [11] A.Fernando Ribeiro and Gil Lopes. Learning robotics: a review. *Current Robotics Reports 2020 1:1*, 1:1–11, 1 2020. URL: <https://link.springer.com/article/10.1007/s43154-020-00002-9>, doi: 10.1007/S43154-020-00002-9.
- [12] Vítor Ventzelos, Gonçalo Leão, and Armando Sousa. Teaching ros1/2 and reinforcement learning using a mobile robot and its simulation. *Iberian Robotics Conference (ROBOT)*, 2022. Accepted for publication.