FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

REDI 4.0 - Robot for Demonstrations with Behaviour Defined on Paper

Isabel Fernandes Neves

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Professor Armando Sousa

February 17, 2022

Resumo

Os contínuos avanços tecnológicos das últimas décadas levaram a que o mercado de trabalho esteja cada vez mais ligado às áreas da Ciência, Tecnologia, Engenharia e Matemática (STEM—Science, Technology, Engineering, Mathmatics). Assim, é necessário que os planos de estudos das escolas evoluam e passem a abranger disciplinas relacionadas com o raciocínio lógico, como a robótica. Esta inclusão será benéfica para os estudantes, uma vez que contribui para o desenvolvimento de pensamento criativo, perseverança, espírito de equipa, assim como outras competências fundamentais. Neste sentido, o projeto *REDI* foi desenvolvido inicialmente como um robot educativo, cujo comportamento era definido pela ligação de sensores e atuadores através de cavilhas. Estas conexões seriam então interpretadas pelo robot, que seguiria as instruções dadas pelo utilizador.

Após várias iterações deste projeto, o trabalho proposto para esta dissertação será o de criar um robot portátil, que apenas necessite de uma folha de papel para ser programado. Para tal, as instruções para comandar a máquina terão de ser definidas de forma completamente nova, uma vez que o design original de construção envolvia a ligação elétrica entre todas as cavilhas, tornando o projeto caro e complexo. Em alternativa, o comportamento do robot aqui detalhado será definido através de ligações desenhadas em papel. Este terá um Raspberry Pi 4 com duas câmaras e um microcontrolador ESP32 que lê dois codificadores e comanda os dois motores. Uma das câmaras será usada como sensores virtuais, enquanto a outra será usada para capturar a foto que define o comportamento do robot. Este será economicamente viável e robusto (sem peças móveis) e fácil de copiar.

O propósito desejado com esta nova ideia é o de manter os custos baixos para os estabelecimentos de ensino, pois retira a necessidade de ter equipamento caro como tablets, smartphones ou PCs, mesmo para a programação do robot. Os utilizadores, frequentemente estudantes do ensino secundário, podem simplesmente utilizar uma folha de papel para "programar" o robot, sendo esta atividade rápida. Isto torna esta experiência barata e educativa. Esta estratégia traz também a vantagem de todos os estudantes poderem estar a trabalhar em simultâneo e individualmente nas suas ligações, com a sua folha de papel, sem terem de interagir com o robot de imediato mas podendo, em poucos segundos, testar o seu programa num robot partilhado.

Esta dissertação foca-se na preparação do hardware do robot e no algoritmo de processamento de imagem dedicado à identificação das conexões desenhadas em papel, que ditarão o comportamento do robot. Este é lido a partir de um diagrama impresso num papel que contém blocos que simbolizam o robot, onde conexões são desenhadas através de linhas coloridas. O software usa a biblioteca OpenCV para descobrir estas ligações que serão depois traduzidas em instruções (usando uma matriz de ligações) para o robot. Vários exemplos de possíveis implementações foram usados para validar o algoritmo, que foi testado com sucesso. As situações incluem linhas cruzadas com deformação de imagem, que pode resultar da lente ou de desalinhamento entre a posição da câmara e do papel. O tempo de processamento máximo da fotografia do diagrama de programação do robot, para os exemplos abordados, é de 3 segundos e inclui a correção da distorção da lente, a homografia para retificar a perspetiva, o processamento de imagem para deteção

de círculos, filtragem da cor para reconhecimento de linhas e a deteção final das conexões.

Abstract

The continuous technological advances in motion throughout the last decades have lead to a shift in the work market, which is more focused than ever on the areas of Science, Technology, Engineering and Mathematics (STEM). To be maintained, this shift requires that the school programs follow suit, teaching courses related to STEM and general logic—for example, robotics. From early on in their school education, the students should benefit from the development of creative thought, perseverance, teamwork and other fundamental skills.

With this in mind, project *REDI* was first developed as a standalone educational robot, whose behaviour was defined by sensors and actuators, connected through plugs. These connections would then be interpreted by the robot which would, in this way, follow the user's instructions.

After various iterations of this project, this dissertation work aims to create a standalone portable robot, that requires only paper to define its behaviour. To achieve that, the instructions to command the robot's motion have to be defined in a new way, since the original design required electrical connections between every plug, which made the full project expensive and complex. As an alternative, the proposed robot's behaviour will be defined through paper-drawn connections. The robot will feature a Raspberry Pi 4 with two cameras and a ESP32 microcontroller to read two encoders and drive the two motors. One of the cameras will be used as virtual sensors while the other is used to capture the image that will define the behaviour. The proposed robot is cost effective and is meant to be robust (no moving parts except the wheels) and easy to replicate.

The goal of this new idea is to keep costs low and make the project more desirable for schools, since this takes away the need of expensive equipment like tablets, smartphones or PCs, not even for programming the robot. The users, likely secondary school students, can simply use a paper sheet to "program" the robot, making this a cheap and fruitful learning experience with a very low setup time to make the first program. This strategy is also advantageous in the sense that each student can work at the same time and individually on their connections, with their own paper sheet, not having to interact with the physical robot straight away but still being able to, in a few seconds, try out the drawn program in a shared robot.

This dissertation focuses on the hardware preparation and on the image processing algorithm dedicated to the recognition of the connections drawn on paper that will in turn dictate the behaviour of the robot. This behaviour is read from a pre-printed paper diagram of the blocks that make up the working of the robot, where connections are drawn as coloured lines. The software used the OpenCV library to identify connections that will be later translated into commands (connection matrix) for the robot. Various realistic sample programs were used as validation cases, and tested successfully. They included situations with lines crossing and with some image deformation which could be a result of the lens of the camera or misalignments of the camera's position to the target piece of paper. The maximum processing time for the photograph of the robot's programming diagram is, for the examples studied, of three seconds and it includes correcting the distortion caused by the lens, homography for perspective rectification, image processing for circle detection, colour filtering for line recognition and connection logic detection.

Agradecimentos

Felizmente tenho muitas pessoas a quem estar grata. Primeiro, quero agraceder ao Professor Doutor Armando Jorge Sousa, professor da Faculdade de Engenharia da Universidade do Porto e orientador desta dissertação, pelo apoio e paciência que demonstrou durante toda a realização deste projeto.

Aos meus pais por estarem presentes e tolerarem a minha frustração nos momentos mais difíceis. À minha irmã, Rita Neves, por todos os conselhos, motivação e ajuda dados. Sem ti, terminar este projeto pareceria impossível. Ao Duarte Rondão, por perceber muito de processamento de imagem e por estar sempre disponível para responder a qualquer pergunta.

E a todos os meus amigos, que me apoiaram e não ficaram chateados por ter de cancelar os planos.

Isabel Neves

Contents

List of Figures

List of Tables

Abreviaturas e Símbolos

AHP	Analytic Hierarchy Process
CPU	Central Processing Unit
FEUP	Faculdade de Engenharia da Universidade do Porto
MCU	Microcontroller Unit
PWM	Pulse Width Modulation
RAM	Random Access Memory
REDI	Robot Educativo para Demonstrações Interativas
RPi	Raspberry Pi
STEM	Science, Technology, Engineering and Mathematics
USB	Universal Serial Bus

Chapter 1

Introduction

1.1 Context

In a world where technological evolution is constantly happening and kids are getting access to it at an increasingly earlier rate, what is being taught in schools has to change to keep up with the most recent developments in pedagogical research. STEM[1] disciplines should be made accessible starting in primary schools and continuing in the curriculum throughout all academic years. Exercising these skills has been proven to help develop creative thought, perseverance, teamwork and other fundamental skills.[2].

It is no secret that classes can be monotone and students lose interest easily when there is no real interaction between the student and the theory being taught. Including disciplines like robotics in school curriculums could be a great way to fight these as the kids would be able to put their lessons in practice.

This dissertation is the continuation of the *REDI* robot, a standalone educational robot, whose behaviour was defined by sensors and actuators, connected through plugs. This project will change the way the robot is programmed. Instead of using electrical connections to define the behaviour of the robot, it will be defined by drawing lines on a paper that will later be translated into instructions.

Projects like this being implemented in schools along the current curriculum would be a great step to diversify what is being taught to our children, to encourage their creativity and teamwork abilities and to help them be forward-thinkers.

1.2 Objectives and Motivation

This dissertation work aims to create a standalone portable robot, that requires only paper-drawn connections to define its behaviour. They will be detected by taking a photograph of said paper and processing the image on a Raspberry Pi 4. This new approach to the *REDI* robot has the advantage of allowing each student to work at the same time and individually on their connections, with their

[1] Science, Technology, Engineering and Mathematics

[2] https://blog.robotiq.com/7-reasons-to-teach-robotics-at-school

own paper sheet, not having to interact with the physical robot straight away but still being able to, in a few seconds, try out the drawn program in a shared robot.

Even though schools should broaden their curriculums to include more STEM disciplines, one justification for not doing so is the added cost to the school's already small budget. By creating a robot as cost effectively as possible and by not having it rely on another equipment like a tablet or laptop (that would be an added expense) without compromising the quality, projects like this become more appealing. This could be a way to attract interest from schools and be a step towards change in the current study plans.

1.3 Dissertation Structure

In addition to the current introductory chapter, there are four more chapters in this dissertation. Chapter 2 contains the state of the art robots used in different educational projects. In Chapter 3, the entirety of the solution is described. In Section 3.4 the selected hardware is detailed and in 3.5 the entire image processing pipeline is explained. Chapter 4 shows the results and analysis of several sample programs. Finally, Chapter 5 is devoted to some conclusions and future work regarding this project.

Chapter 2

State of the Art

This work will, hopefully, be implemented in schools to help teach young students how to think logically and to learn other STEM skills. As projects for schools and for children have been increasingly created in the last decade, it was necessary to study several existing projects to know what already exists and how *REDI* will differ from them.

2.1 Previous Projects

This dissertation work is the continuation of several projects that will be described in this section.

2.1.1 RODA - RObot Didáctico de Aveiro

The REDi concept was inspired by the RODA robot which is a modular robot built for competitions by Bruno Pires in Aveiro University [1] in 2008. The basis of the robot is that it is programmed by connecting plugs into operational blocks and thus defining the desired behaviour. Figure 2.1 shows the different connection points for the plugs to be inserted on. This allows for someone with minimal knowledge of programming to program a robot and have instant feedback if the instructions given originated the desired actions.

Figure 2.1: RODA robot

It also has two variations for its assemble. As it is built in layers, by switching them between themselves, the robot can be used in the *Micro-Mouse*[1] or *Firefighter Robot*[2] competitions as can be seen in figure 2.2.

(a) Micro-Mouse Variation (b) Firefighter Robot Variation

Figure 2.2: Variations of RODA

2.1.2 Original REDi

The original REDi robot [2] was created to be an educational tool for younger students unlike the RODA robot. Even though the programming logic is the same, i.e. the behaviour is defined by connecting logical blocks with wires, the hardware and software are different and were developed from scratch according to the project's needs. The final model can be seen in figure 2.3 with several wires connected.

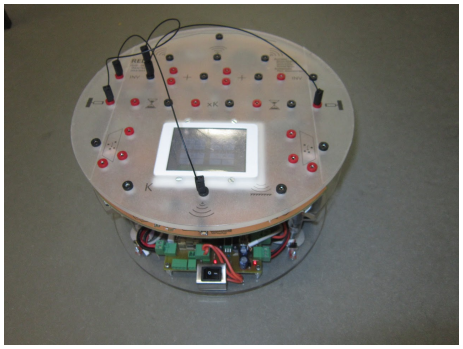
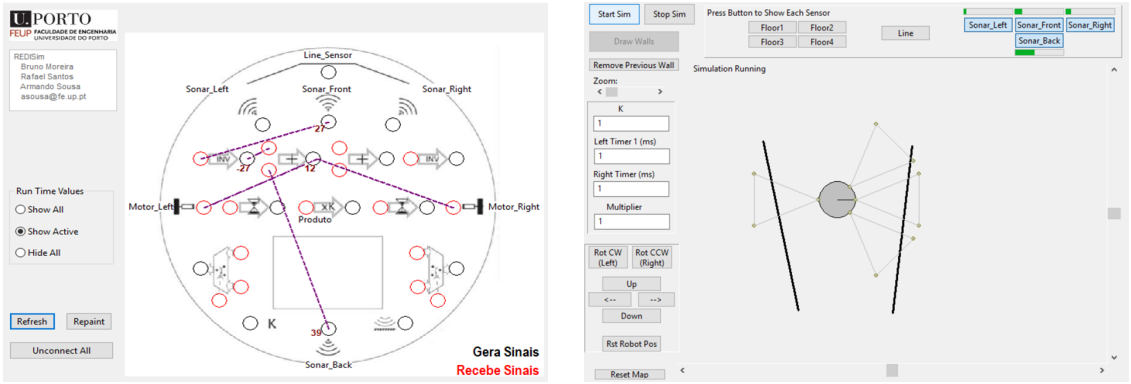


Figure 2.3: REDi robot

This project started in 2015 and consisted in creating the aforementioned robot and an online simulation platform. On one window of the simulator, the desired behaviour is defined by connecting the virtual points to one another as shown on figure 2.4a. On the other window (figure 2.4b), the simulation starts and the robot's actions are observed to see the result of the programming done previously.

[1] http://microrato.ua.pt/
[2] http://robobombeiro.ipg.pt/



(a) Connection of wires (b) Execution of robot

Figure 2.4: Simulator RediSim

2.1.3 REDi3

As a result of a dissertation made in 2018, Vasco Pinto [3] created the REDi3, a robot whose behaviour was programmed on the simulator mentioned above. As the REDi project continues to be for educating young students, the cost of the robot is important because it limits the number of schools that can afford to have it. The different way of programming when compared to the original allowed the project to become cheaper, lighter and smaller.

In figure 2.5 it can be seen that the main chassis is an AlphaBot (mentioned in section 2.3) and the mirror support was designed specifically for this purpose and 3D printed. The V-shaped mirror was used to allow the camera to be a virtual sensor so the robot could see obstacles in its course.

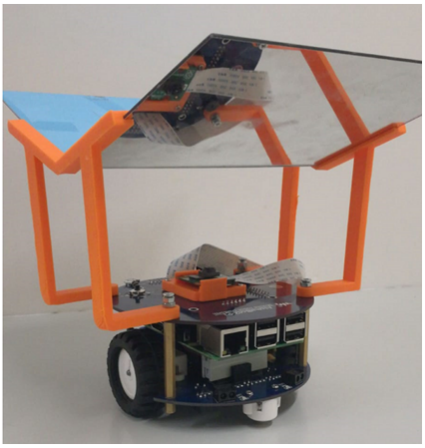


Figure 2.5: REDi3 robot

2.2 Behaviour Defined by Image Processing

Even though the work to be carried out is a continuation of an existing project, the method of defining the robot's behaviour will be new and so it was essential to find out what types of projects

exist that also require image processing and in what way are the commands defined.

By researching, different projects that resort to image processing were found. T-Maze is a tangible programming tool developed by Wang [4]. It uses 3D blocks with images printed on them to create a logical program. When completed, a photograph is taken of the entire sequence. Another interesting project was the *Tactode* [5] project. It uses puzzle-like pieces to define the desired behaviour and that information is decoded through the processing of a a captured picture.

Even so, a similar project to the one proposed where the robot's behaviour is defined by drawing on a paper was not found.

2.3 Robots

As this dissertation also consists of creating a robot, it is necessary to study the different types that exist on the current market. The table 2.7, based on the dissertation [6] by Angela Cardoso, compares several attributes of different educational robots.

Analysing the table, the PiBot was identified as a possible choice as it is *open source*, reasonably cheap, uses Raspberry Pi and has a camera. This project was developed by Vega [7], in 2018, to help robotics instruction in secondary education. One of its goals was to be low cost so that price was not an obstacle to learning. Thus, its structure is a model designed to be 3D printed.

Another interesting robot due to its price and components is the AlphaBot2[3]. Although it is not found in the table 2.7, it is frequently used for teaching [8] and also has a camera. Both robots can be seen in figure 2.6.

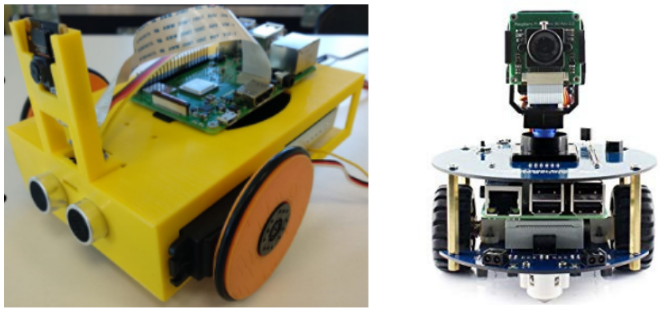


Figure 2.6: PiBot (left) and AlphaBot2 (right)

For the 2019 Portuguese Robotics Open, it was necessary to develop a robot for the Robot@Factory Lite competition. Lima [9] built a vehicle that, like the PiBot, has its chassis printed in 3D, lowering its production cost. This model is relevant as it continues to be developed at Faculdade de Engenharia da Universidade do Porto, where the next version of *REDI* will be created.

[3] https://www.waveshare.com/wiki/AlphaBot2

Robot	Age	Price	Coding	Processor	Sensors	Actuators	Comms	Others
EV3	10+	400€	EV3, RobotC, Scratch, C, C++, C#, Java, Python,...	ARM926EJ-S core @300MHz	touch, color, infrared	178×128 Monochrome LCD, 3 motors	USB, WiFi, Bluetooth	remote control, microSDHC, multiple robots
Cozmo	8+	150€	Scratch, ScratchJr, Python	no info	camera VGA 30fps, cliff, encoders and IMU	speaker, 128×64 OLED display, motors for: head, lift, threaded wheels	Bluetooth, IR receiver, IR transmitter	facial recognition, programmed like pet, plays games
KUBO	4-10	256€	TagTile	no info	RFID	motors, RGB LEDs	WiFi	includes tangible language
KIBO	4-7	190€-420€	wooden blocks	no info	sound, light, distance, barcodes	motors, LEDs	—	includes tangible language
mBot	8+	80€	mBlock	based on Arduino Uno	light, ultrasonic, line follower	buzzer, RGB LED, 2 motors	—	4 ports for more sensors
Boost	7-10	160€	visual	ARM Cortex M0 @48MHz	color, distance	motors, RGB LED	Bluetooth	multiple robots
Dash	5-8	130€-230€	Blockly, Swift	ARM Cortex M0	3 microphones, 3 distance sensors	speaker, motors, RGB LEDs	Bluetooth, 2 IR receivers	multiple accessories
Thymio	6+	155€	visual, Blockly, Scratch, text	PIC24 @32MHz	5 capacitive touch buttons, accelerometer, 5 proximity, 2 ground, microphone, temperature	2 motors, 39 LEDs, speaker,	USB, WiFi, IR receiver	pencil holder, trailer hook, memory card
Ozobot	8+	75€	Blockly, Ozocodes	no info	4 proximity, 1 optical (color)	2 motors, LEDs, speaker	micro USB, Bluetooth	remote control mobile application
Sphero	8+	110€	drawing, block based, JavaScript	ARM Cortex	accelerometer, gyroscope	2 motors, 2 RGB LEDs, 1 blue LED	Bluetooth	remote control mobile application
Robobo	10+	363€	Blockly, Scratch, ROS, Java	base: low capacity phone: depends on model	base: 9 IR proximity, 4 odometric encoders phone: 2 high-res cameras, proximity, light, temperature, gyroscope, accelerometer, magnetometer, GPS, microphone, touch screen	base: 4 motors, 9 RGB LEDs phone: speaker, high-res LCD screen	base: Bluetooth phone: 3G/4G, WiFi, USB	includes interactive lessons, accessories
PiBot	2+	180€	Python	based on Raspberry Pi 3	camera, infrared, ultrasonic, encoders	2 DC motors	WiFi USB	simulator included, Open source

Figure 2.7: Comparison between Educational Robots

Chapter 3

REDi 4.0

3.1 Problem Definition

In order to motivate students to learn new skills like robotics, it is necessary to innovate and bring them new and exciting projects.

This dissertation is the evolution of the *REDI* robot which was a standalone educational robot, whose behaviour was defined by sensors and actuators, connected through plugs. Afterwards, a simulator *RediSim* was created making it possible to program and run the robot without having a physical one with you. The problem with this is that students tend to learn more when they can interact with devices they can touch and manipulate. Because of that, the next iteration was made where the instructions were given on the simulator but the physical robot was the one to execute the commands.

As schools have a low budget, it is important for this project to be cost effective. Therefore, it will be created a standalone robot that does not depend on an extra device (e.g. tablet, laptop) but that is more affordable than the original robot. As such, this project has to have a new and affordable way to define the behaviour of the robot in order to eliminate the need of expensive and complex electrical connections.

3.2 General Architecture

Previous work regarding the *REDI* robot used a portion of software written in Free Pascal on the Lazarus platform to read the connection matrix and later execute the defined behaviour from the connection matrix.

The approach taken in the current work is to read the connection matrix from the photograph, taken by the USB camera, of the paper with the drawn connections.

After successful reading of the connection matrix, the previous software written in Lazarus would run with the adaptation of the movement commands that are to be sent to the ESP32 micro-controller.

During the execution of the defined behaviour, another piece of software reads the camera to emulate sensors in order to produce virtual sonars. This is done by assuming that the background is mostly of the same colour and that obstacles will cause clear edges on the image received by the Raspberry Pi camera.

3.3 Proposed Solution

With the creation of a robot being a fundamental part of this project, it was necessary to decide if a pre-existing robot should be bought like the AlphaBot2[1] because of its reduced cost, or if it would be more advantageous to build one from scratch or based on other projects (e.g. PiBot, RaFL) as it would allow for more freedom to customise it.

Before making this decision, it is essential to establish which major components will be used. For movement we will use a ESP32 microcontroller with two encoders and for image processing we will have a Raspberry Pi 4.

To mitigate the cost of the robot, it was decided to define the behaviour of the robot through the drawing on a paper of connections that are later translated into instructions. For that to be possible, a photograph of the paper will have to be captured so the image can be precessed and the connections detected.

To implement virtual sensors and image capturing, two possibilities were considered. The first would be to use one fixed camera for the Raspberry Pi aimed at a chrome lamp mounted on top of it. This system was explored and developed in the article by Ferreira et al [10]. This structure was developed for the Alphabot2 robot so it might have to change. This method would cause the collected image to be curved due to the shape of the lamp, but it would not be a problem as it would still be possible to calculate the distance from the robot to any surrounding obstacle and to capture the photograph of the paper.

The second proposed method would be to implement two fixed cameras, oriented in opposite directions. The first would be pointing towards the ceiling while the other would be facing the floor and would have a 200º lens[2]. When mounted at a distance from the top of the robot, this hypothesis would allow capturing the puzzle through the lower camera (e.g. picking up the vehicle and pointing the camera at the place where the paper is located) and the upper one would detect obstacles close to the robot.

To make this decision, the Analytic Hierarchy Process was used. In this process, which methods to compare and which parameters to assess are identified. Giving weights to these criteria and evaluating each method according to each factor, the final result is achieved through the sum of the multiplications between the given judgments and the weights of the defined criteria. Thus, table 3.1 represents the AHP performed to decide which image collection method to use.

[1] https://www.botnroll.com/pt/kits-para-montagem/2709-alphabot2-plataforma-rob-tica-m-vel-compat-vel-com-raspberry-pi-3-n-o-inclu-a-raspberry-pi.html

[2] https://mauser.pt/catalog/product_info.php?cPath=1667_2620_2621& products_id=096-7651

Table 3.1: AHP of Image Capturing

		fixed camera + chrome lens	2 cameras
Robustness	0,4	0,1	0,9
Cost	0,2	0,5	0,5
Personalization	0,1	0,4	0,6
Resolution	0,3	0,3	0,7
SCORE		0,27	0,73

The robustness criterion is very important as this project is intended to be implemented in schools, an environment where the repair of components will not be immediate. Cost also has to be taken into account as it affects the likelihood of the product being purchased. In this case, personalization is not fundamental, unlike image resolution, which is a decisive parameter for the success of the image processing. Analysing table 3.1, using two cameras seems to be the best decision as it allows the project to be more robust and the image quality to be better.

Once this decision was made, the same process was used to decide which type of robot to use, which is represented in table 3.2.

Table 3.2: AHP of Robot

		PiBot	AlphaBot2	RaFL
Robustness	0,2	0,4	0,25	0,35
Cost	0,2	0,4	0,2	0,4
Personalization	0,3	0,1	0,05	0,85
Raspberry Pi	0,1	0,5	0,5	0
Microcontroller	0,1	0	0	1
Camera	0,1	0,5	0,5	0
SCORE		0,29	0,205	0,505

As before, robustness and cost are criteria that have to be considered. The existence of a location for a Raspberry Pi, for an ESP32 or for a camera is also taken into account but these are not fundamental parameters as any of the robots will have to be customized.

With the decisions taken, the proposed solution will be a differential robot based on the one developed for Robot@Factory Lite with a Raspberry Pi 4, an ESP32, two motors with encoders and two cameras, one of them with a 200º lens. The final *REDI* project will then consist of a robot whose behaviour is determined by lines drawn on a pre-defined paper with a fixed schematic.

3.4 Hardware

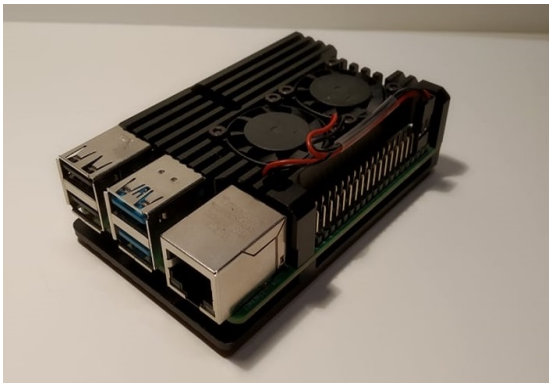
As it was decided that a new robot was going to be built anew, which components to use could be chosen without limitations from previous projects. As such, every element was picked to fit each other and to be non-expensive and of easy access on the market.

3.4.1 Raspberry Pi and ESP32

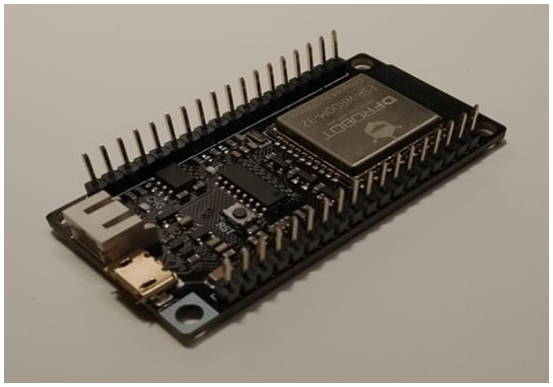
The Raspberry Pi 4 (figure 3.1a) is a low-cost, single board computer of reduced size, in this case featuring 4GB of RAM. This kind of computer allows only one high frame rate camera (RPi camera) but allows other high resolution USB cameras at a low frame rate. Another advantage of RPi programming is Linux compatibility and easy OpenCV interfacing. It also has Bluetooth, Ethernet and Wireless connections that will not be used in the final product but that were useful during its development. In figure 3.1a, the Raspberry Pi is enclosed in a heat sink case (in black) in order to stop it from overheating which could lead to a decrease in performance

As will be seen in 3.4.2, two cameras will be used: the RPi camera is reserved for virtual sensing and the USB camera will be used to photograph the paper with the connections. This means that the frame rate of the USB camera can be low but the resolution will limit the capability to read the connections from the paper.

The ESP32 is a microcontroller that transforms the instructions given by the Raspberry Pi in a way that can be interpreted and executed by the motors. This MCU was chosen instead of the popular Arduino because it is a more complete board with Wireless and Bluetooth connectivity and has a lower power consumption. It can be viewed on figure 3.1b.



(a) Raspberry Pi 4 with cooling case



(b) ESP32 microcontroller

Figure 3.1: Development Boards

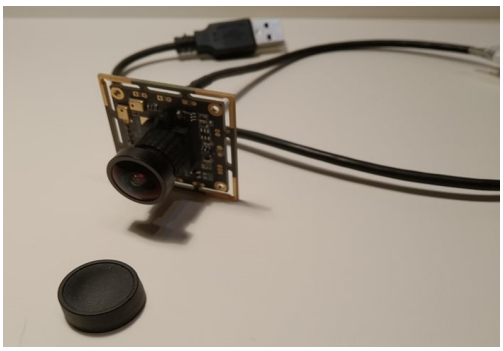
These two components communicate through Serial Port by connecting the Txd pin (pin used to transmit data) of the Raspberry Pi 4 to the Rxd pin (pin used to receive data) of the ESP32 and vice versa. The transmission is made with a Baud-Rate of 115200 and follows the SERIAL_8N1 protocol, which means that the communication starts with one start bit followed by eight data bits,

no parity bit, and one stop bit. This information is then sent as PWM signals to the motors, as can be studied in figure 3.5.

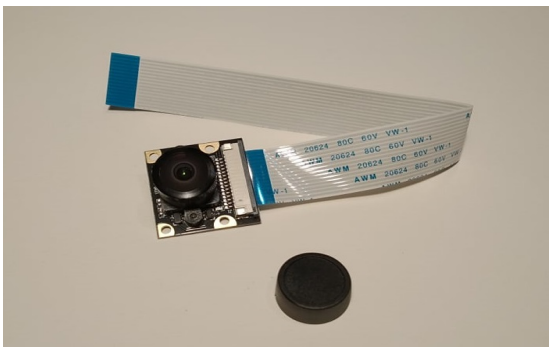
3.4.2 USB and Raspberry Pi Cameras

As previously mentioned, the instructions for the robot will be decoded from pictures taken by a camera. In this case it was chosen a USB one, shown in figure 3.2a, with a 140 degree lens and with a maximum resolution of 1920x1080 pixels for a frame captured in video mode. This camera is positioned on top of the robot, with the lens turned skywards, as seen on figure 3.6a.

The Raspberry Pi camera, shown on figure 3.2b, connects to the microcomputer by its ribbon cable and has a 200 degree lens to double as a virtual sensor. By being placed parallel to the ground with the lens turned down, it is possible to calculate the distance to any obstacles in its view. This features is not yet implemented but it was partially developed.



(a) Raspberry Pi 4 camera with 140 degree lens



(b) Raspberry Pi 4 camera with 200 degree lens

Figure 3.2: Raspberry Pi Cameras

3.4.3 Power

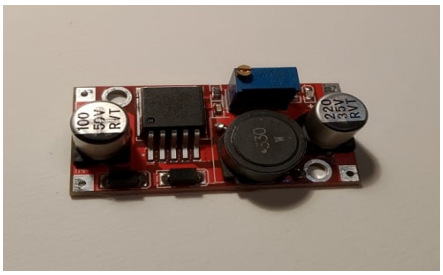
As the various chosen components need different power supplies, it was decided to use two 18650 batteries connected in serial for a 7.2V supply and two converters to reduce it to 5V and 3.3V as the Raspberry Pi 4[3] and the ESP32[4] have to be supplied by these exact values, respectively. Furthermore, the DC motor and its encoders[5] can also use 5V as it is a value in their accepted range.

3.4.4 Electrical Diagram

The electrical diagram for the motors connection with the drivers and ESP32 was reworked a number of times but in the end it ended up being similar to the one proposed in the open repository

[3] https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/
[4] https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf
[5] https://wiki.dfrobot.com/Micro_DC_Motor_with_Encoder-SJ01_SKU__FIT0450

(a) 18650 batteries



(b) Power converter

Figure 3.3: Power sources

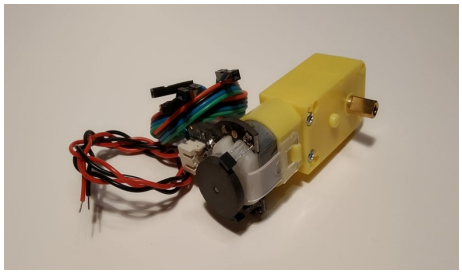


Figure 3.4: DC Motor with Encoder

of *curiores* in GitHub[6], shown in figure 3.5. The motors shown in the picture are the same as the ones used in this project but the model of the drivers and microcontroller are different.

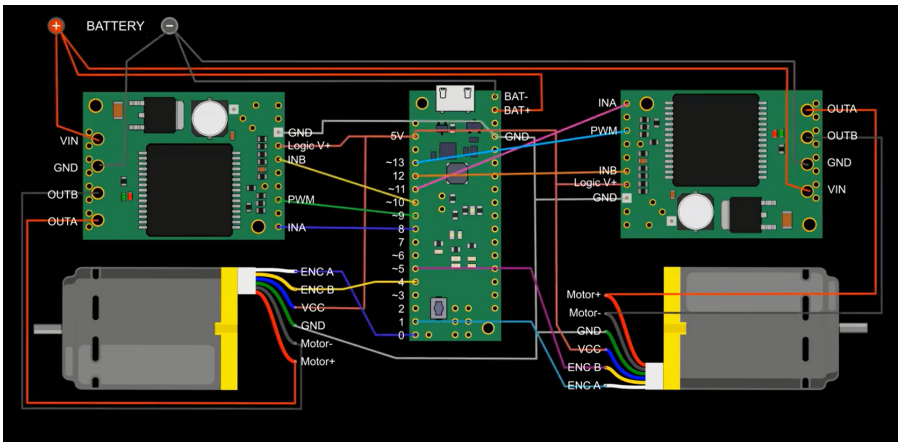


Figure 3.5: Electrical Diagram of the connection of the motors to the encoders [6]

3.4.5 Bill of Materials

As the purpose of this project is to be integrated in schools, the overall cost of the robot had to be low. In the following table 3.3, the sum of all components was reached a total of around 254€.

[6]https://github.com/curiores/ArduinoTutorials/tree/main/MultipleEncoders

Even though the unit prices are rounded up, it was considered that the total cost of the robot's components was 260€ to leave some leeway for other building costs.

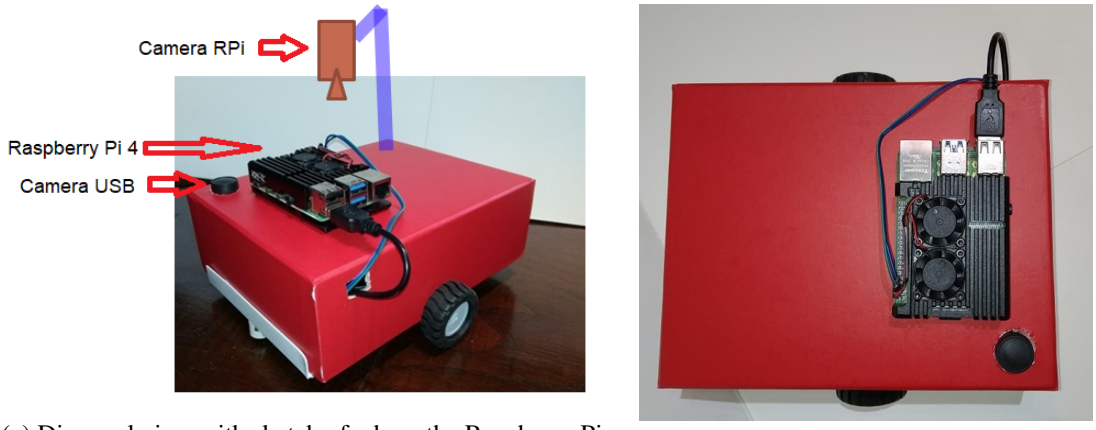
Table 3.3: Bill of Materials

Component	Quantity	Price	Link
Raspberry Pi 4	1	65,00 €	https://bit.ly/2Ybwi2E
ESP32	1	20,00 €	https://bit.ly/2Yiwcqr
Step Down converter	2	8,00 €	https://bit.ly/39YXpAI
Batteries	2	10,00 €	https://bit.ly/2ZQsk0s
USB Camera	1	40,00 €	https://amzn.to/3B8Jq7G
Raspberry Pi Camera	1	40,00 €	https://bit.ly/3l0wLhq
Driver	1	16,00 €	https://bit.ly/3D2kCP5
Motor + Encoder	2	18,00 €	https://bit.ly/3omAS9y
Castor Wheel	1	3,00 €	https://bit.ly/3A3Nc0B
Wheels	2	6,00 €	https://bit.ly/3orVJIy
Cooling Case	1	12,00 €	https://bit.ly/3a1FskY
Battery Supporter	2	2,00 €	https://bit.ly/2Y8qpDq
Pressure Button	1	7,00 €	https://bit.ly/3ipfVqM
Battery Charger	1	7,00 €	https://bit.ly/2Y3GWbU
TOTAL		254,00€	

3.4.6 Final Assembly

All the components mentioned above in table 3.3 were then assembled. Even though the plan was to build a robot based on the one from Robot@Factory Lite, the health pandemic that occurred limited the access to the faculty and to the resources needed to adapt and print a 3D chassis. As such, the prototype was built on a sturdy 16.5x20.5x6.5cm cardboard box as can be seen on figure 3.6.

Every component was placed on the inside of the box with the exception of the Raspberry Pi 4. This was decided so that, even though each element is accessible, they will be hidden from view and protected by the cardboard. On the bottom right corner of figure 3.6b, the USB camera can be seen peeking from below also in an attempt to preserve it while having it be able to take photographs. The necessary connectors pass from the top to the bottom of the box through a small gap on its side, as seen on figure 3.6a.



(a) Diagonal view with sketch of where the Raspberry Pi Camera would go

(b) Top view

Figure 3.6: Prototype of Robot

3.5 Software - Image Processing

3.5.1 Introduction

One of the objectives of this dissertation is to program a robot by drawing connections on paper. To do so, a picture will be taken of said paper; but that in itself is not enough for the instructions to be deciphered. Thus, it was necessary to process the photograph.

Image processing entails taking a picture and extracting meaning from it. In this case, the desired information is composed by the connections between points in the pre-defined schematic which will then be interpreted into commands for the robot. However, first, these connections have to be discovered from the image data.

An image is a matrix of pixels. Each pixel is used to represent a colour. In this dissertation, the colour will be defined mostly in the RGB colour space. It is composed of three channels that respectively represent red, green and blue. Each ranges from 0 to 255, and the weight of each value impacts the pixel colour equally. The HSV colour space will also be used in this work; it too has three channels, but they signify hue, saturation and value. The first one ranges from 0 to 179 and represents the pixel colour while the last two range from 0 to 255 and indicate its intensity and lightness, respectively.

To process the pictures taken, the OpenCV library was used, since it is open-source, comprises a large catalogue of functions and tools, and is recommended for projects that involve computer vision or image processing. It is also compatible with Python, the programming language selected for having many free open-sourced libraries created by online communities.

3.5.2 Pipeline

In the following subsections, the whole process from taking a picture to the robot moving will be explained in a more detailed way. On figure 3.7, a more generic summary is shown in the form of a diagram.

3.5.3 Schematic of Connections

As mentioned previously, the basis of this project is to have a robot move in a way determined by the connections drawn on a sheet of paper. For that, a reference schematic was needed: that is, a design that is the same in every paper sheet, so that the relative position of the connection points is known. From fere on out, connection points are the little circles on the paper that were the plugs on the original robot and it is to and from them that connections are established. The chosen model resembles the surface of the original REDI robot and can be seen in figure 3.8. The numbers depicted in this figure are only here represented for indicative purposes, and do not appear on the actual project's paper sheets.

The different symbols in figure 3.8 [2] are enumerated in red and represent several sensors, actuators and operations. Number 1 is a line sensor; 2,3, 4 and 17 are proximity sensors and 18 is a floor sensor. The two actuators present are the motors at 9 and 13. Numbers 5 and 8 are inverters,

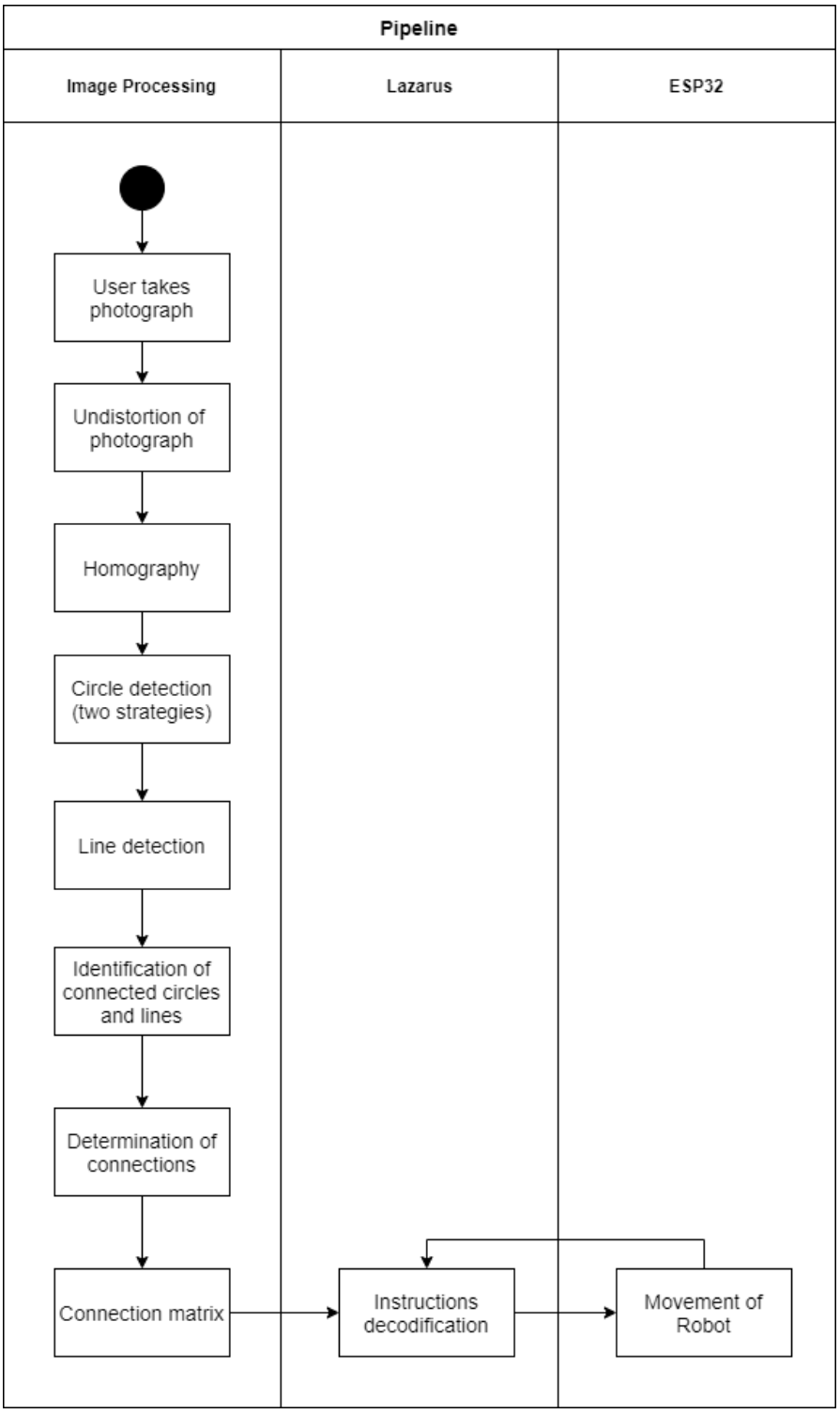


Figure 3.7: Pipeline

6 and 7 adders, 10 and 12 timers and 11 is a multiplication by the constant at number 16. There are also two multiplexors in 14 and 15.

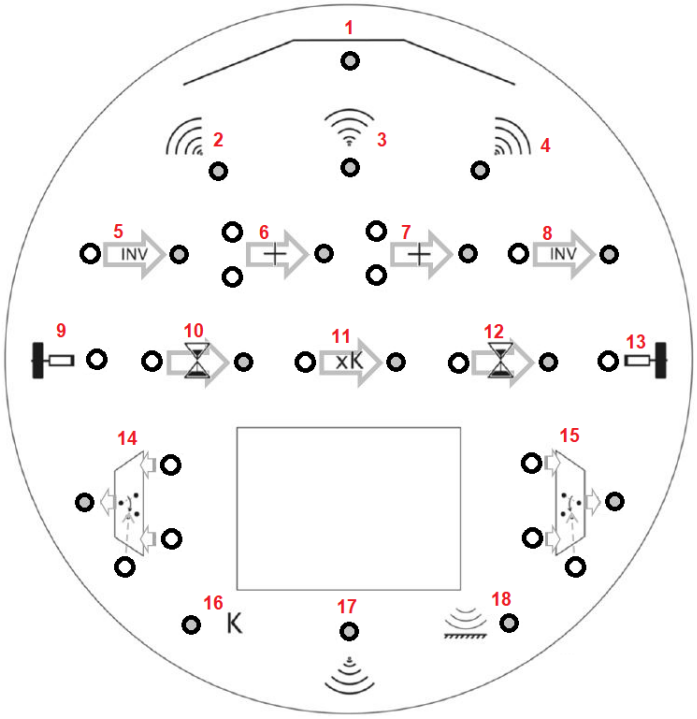


Figure 3.8: Schematic of the Robot's Connections

3.5.4 From Photograph to Connection Matrix

This section is dedicated to the process of analysing the photograph taken by the USB camera to extract the information drawn on by the user to ultimately provide the robot its commands. The way to do it is by using different computer vision algorithms that will be expanded upon in the following sections.

3.5.4.1 Calibration

The calibration of a camera is divided into intrinsic and extrinsic parameters. Intrinsic parameters include internal elements such as lens and image sensor; extrinsic parameters deal with the position of the camera to the world.

When a camera has a wide-angle lens, the resulting image can have a curved look around the edges. This event is called radial distortion and can be corrected by calibrating[7] the camera so the final picture is true to the object that is being captured. For the calibration to be successful, there are three major elements that have to be calculated:

- the distortion coefficients, associated to the distortions of the camera;
- the camera matrix: unique to every camera, with values such as focal length and optical centres;

[7] https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html

- translation and rotation vectors, associated to the images captured by the camera.

For these last ones to be obtained, it was necessary to print out an image of a distinct pattern in order to have a dataset of pictures with easily known points. The most commonly used design is a chessboard, as it is composed by alternating black and white squares, which provides an easy relation between the real measurements and the ones calculated in the photograph. It was important to collect an ample dataset of images, from different perspectives and various distances to the camera, as can be seen in figure 3.9. For each picture in the saved set, `cv2.findChessboardCorners(image, chessboardDimension, flag)` finds the chessboard corners; when done, this information is used in `cv2.calibrateCamera(objectPoints, imagePoints, imageSize, cameraMatrix, flags, criteria)` to determine the above mentioned elements.

Figure 3.9: Example of pictures for the chessboard dataset

As these values are always true for this camera regardless of the picture captured, the calculations may be done only once to save execution time. With this data, for every new photograph, `cv2.getOptimalNewCameraMatrix(cameraMatrix, distCoeffs, imageSize, alpha, newImgSize)` calculates the new camera matrix and region of interest. In this function, alpha (the scaling parameter) equalled 1 so that no pixels from the source image were lost. At last, the photograph was undistorted with the use of `cv2.undistort(src, cameraMatrix, distCoeffs, dst, newCameraMatrix)`.

3.5.4.2 Homography

When the photograph is first taken, it generally is never in an ideal positioning to be analysed as its perspective is, almost always, askew and there will be irrelevant information that could be erased. To this effect, it was necessary to calculate the homography matrix H to then apply a perspective transformation to the image. The above mentioned matrix is the relation between the points in one picture to the corresponding ones in another image and is the result of the function `cv2.findHomography(`pts_{src}`,` pts_{dst}`)`.

To simplify which points to use for this transformation, ArUco markers were placed in the corners of the paper. An Aruco marker is a "synthetic square marker composed by a wide black border and a inner binary matrix which determines its identifier (id). The black border facilitates its fast detection in the image and the binary codification allows its identification and the application

of error detection and correction techniques."[8] [11] As the ArUco Library is open-source, very easy to use and is integrated in the OpenCV library, it was decided that it was a good choice to use it to aid correction of perspective and orientation.

Consequently, ArUco markers were placed in the corners of the reference paper to provide easy points for the homography function. Beginning with only one marker, it was observed that the final perspective correction was not ideal as the image was mostly corrected around the corner with the ArUco, leaving the rest of the picture with distortion that was still significant. Applying two markers, each to one corner, the result was improved but there were still some unwanted warp. Placing three ArUcos, as seen on figure 3.10, proved to be enough to produce a final perspective correction with minimal distortion. After calculating the resulting matrix H, `cv2.warpPerspective(img, H, img.size())` was run and the photograph was returned in its ideal perspective, with the irrelevant information to the analysis erased.

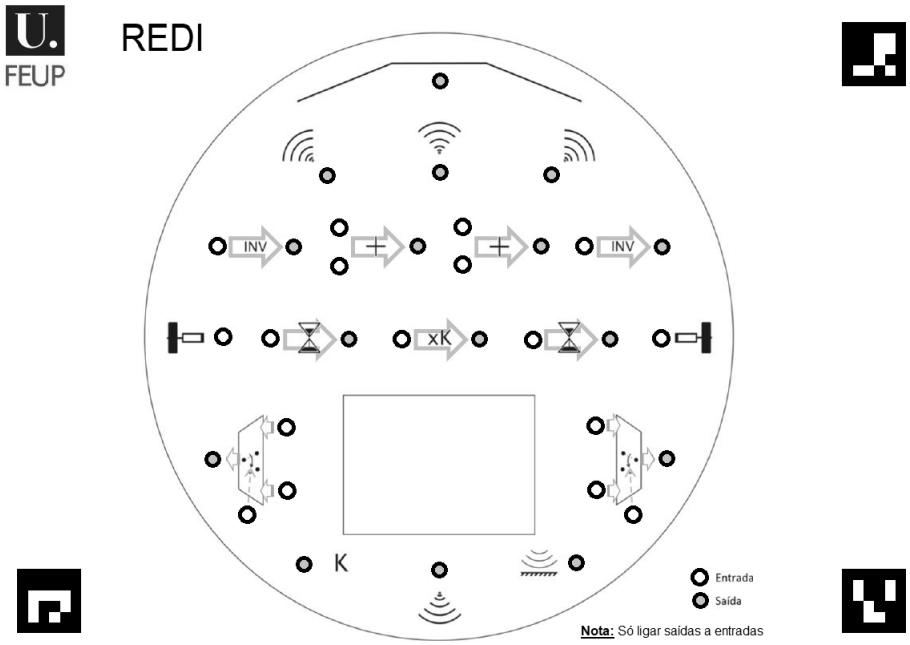


Figure 3.10: Schematics that will be given to the students to draw on

3.5.4.3 Connection Points Detection

After applying the homography function, the next step was to determine where the connection points were located. The first approach was to use the Hough Circle Transform [12] as it is a standard function to detect circles. In a simplified description, this method works in the following manner: a circle is defined by the parametric equation 3.1, where a and b are the coordinates of the circle's centre and r is its corresponding radius. The algorithm states that, for every (x,y) coordinate, all possible values of the unknown parameters (a,b,r) are stored in what is called an

[8] https://docs.opencv.org/4.1.0/d5/dae/tutorial_aruco_detection.html

accumulator matrix. After all coordinates are checked, the maxima of the matrix are taken as the parameters that define the circles.

$$(x-a)^2+(y-b)^2=r^2 \tag{3.1}$$

In the cv2 library[9], this function is made faster by using a gradient method, also called the *2-1 Hough Transform* as detailed in [13], which uses the normal of each edge point.

This is supposed to be a fail-safe method but, when tested, the results were not good enough for the purposes of this project. It either returned to many circles that were not even in the picture or it did not return the important ones. This could probably be improved by dividing the image in several parts and applying this method to each section, instead of using it for the whole page at once.

The next approach was to use the function `approxPolyDP(contour, precision, True)`, which is a feature of the contour method. It takes the contour shape and approximates it to a closed polygon with less vertices, where the distance between them is less then a specified precision value. It uses the Douglas-Peucker algorithm[10] which takes a line composed of several segments and finds a close trajectory with less pieces. Thus, by narrowing down the results to only those with 8 or more segments (value used because it is considered that a closed polygon with more than 7 parts has to be a circle) and discarding the ones whose area is either too large or too small to match the estimated area of the connection points of the pre-defined schematic, a list was made with the centres of each resulting circle.

Even though this method was more viable than the Hough Circles Transform, it was not 100% successful as some circles got deformed by the lines drawn by the users. As such, a safeguard was added to ensure that every connection point is detected. As the design printed in the paper sheet is always the same, the position of each element relative to the paper is fixed. As a reference photo exists, the radius and centres of each circle were hardcoded into the script. This reference comes in the form of dictionary *dict_ref*: it associates the coordinates of each schema's connection point with both a label indicating whether that is an input or output, and with a tag to be recognised by the Lazarus program (these values have no correlation to the labels on figure 3.8).

With this information, a translation vector was calculated in order to provide a reasonable approximation of where the missing points should be in the picture. With these two steps, a dictionary *dict_circles* was saved of the centres of the circles in the photograph with its corresponding coordinates in the reference frame.

3.5.4.4 Line Detection

The instructions for the robot are defined by the user drawing lines between the connection points with a coloured pen to differentiate the sketching from the print of the paper sheet. Therefore

[9] https://docs.opencv.org/4.5.1/d3/de5/tutorial_js_houghcircles.html

[10] https://en.wikipedia.org/w/index.php?title=Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm&oldid=1040270574

detecting the lines drawn on the paper is necessary to identify which instructions are being given by the user.

To do this, the image was converted into the HSV colour space (briefly explained in subsection 3.5.1). This conversion was made because this colour space is more invariant to shades of grey and distinguishes vivid colours more easily than the RGB one. This happens because one can limit the saturation and lightness values to exclude most shades of grey in HSV while including every hue, where in the RGB colour space one can only manipulate the amounts of red, green and blue in the final colour.

By creating a lower and upper threshold of colours and only saving the pixels with values in that range with the `cv2.inRange(img, lower_thresh, upper_thresh)` function, the drawn-on lines were detected and the background eliminated by using the bitwise operation `cv2.bitwise_and(img1, img2, mask = mask)`. To improve the connections for analysis, as the previous operations decrease their quality, they were dilated with a kernel of size 7 and put through a median filter to remove unwanted noise without blurring its edges.

The first method used to determine which lines represent which connections was the Hough Line Transform. It works with the same principle of the Hough Circle Transform explained in subsection 3.5.4.3. As the desired object of detection is a line, the parametric equation will be the one defined in equation 3.2 where "r is the length of a normal from the origin to this line and θ is the orientation of r with respect to the X-axis"[11]. Consequentially, every possible values of r and θ for the different (x,y) tuples are the ones collected on the *accumulator matrix* and the maxima are accepted as the parameters that define the line. The problem with this method is that it only detects straight lines and the drawing may contain curved lines.

$$x\cos(\theta)+y\sin(\theta)=r \tag{3.2}$$

With that in mind, it was decided to use the contour function `cv.findContours(image, mode, method)` as it returns a vector of edge points which can be manipulated.

3.5.4.5 Connection detection

Following the identification of all the connection points and drawn lines, a list of every circle that intersected the contour of the detected lines was saved. Thus, all other connection points would be ignored and every contour that did not intersect with a circle would be considered unwanted noise and eliminated.

Afterwards, each contour was analysed individually. Even though the contour pixels were known, it would be better to be able to separate the data of each line. The chosen solution was to implement the `trace_skeleton.from_numpy(img)` function that returns a series of polylines.

[11] https://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm

A polyline is a continuous line composed of one or more segments. This means that the data is now separated into individual lines. According to the information on the functions' repository[12], using Python to run this method would be very inefficient as it would take too much time. Thus, to expedite the process, this function was used on C++ but with a wrapper in python to still be able to run with the rest of the script.

To be able to implement the aforementioned function, it was necessary to first skeletonise the image using the function skeletonize(img)[13] from the *scikit-image* library. This library is open-source and is used for image processing with the Python computer language. This function is a "method for extracting the skeleton of a picture by removing all the contour points of the picture except those points that belong to the skeleton" [14].

With this done, the input and output circles were identified, and each input was studied individually. This analysis consisted of: 1) For circle a, the polyline that connects to it was detected by choosing the closest one to the centre of a; 2) Its slope was determined; 3) The slopes of the hypothetical lines that a makes with every output circle were calculated; 4) The difference between the result of step 2 and every outcome of step 3 were computed; 5) The output circle b which hypothetical line to a had the minimum result of step 4 is considered the other end of the connection, i.e. a is connected to b.

Finally, a connection matrix is initialised and its positions (a,b) and (b,a) are filled with 1.

The first methods that were tried to identify the connections were based on "following" every line, that is to say, one would start from one end and collect every point that was identified as belonging to that line, until the other end was reached. The main problem found by doing this was how to know which points belonged to which lines different ones crossed. By using the algorithm described above, this problem is solved as the information needed does not depend on the middle of the lines where crossings happen, but on the periphery as that is where the input and output circles are.

3.5.5 Movement

As will be mentioned in the future work section, the idea is to integrate current and previous work. The current project will build a connection matrix (a "program") that will be sent to the previous software in order to be executed. Prior work also implemented a very simple set of virtual sensors that would be continuously read and fed into the executing program that actually implements the wanted behaviours (calculating outputs based on operations with the available signals).

[12] https://github.com/LingDong-/skeleton-tracing
[13] https://scikit-image.org/docs/dev/auto_examples/edges/plot_skeleton.html

Chapter 4

Results and Discussion

4.1 Introduction

In this chapter, six different schematics will be shown and will be denominated as A, B, C, D, E and F, with increasing levels of perceived complexity.

Three experiments were considered essential: the first one was to study the results of every important step of one example; the second one was to, at a constant distance, run various scenarios with incremental difficulty, and check that the final connections are correctly drawn; the last one was to take photographs of one schematic at different distances. After verifying that the results of the second experiment were as expected, the script was run ten times to average the execution time of each example.

Another test consisted on analysing several coloured pens to identify which colours were the best to draw with, and to also experiment with varied thicknesses of pen tips to see what effect this has on the lines' detection.

As a result of the current world situation, it was not possible to go to schools and trial this project with the students that are the target audience of this dissertation. As such, there will be no user evaluation to further demonstrate the success of this work.

4.2 Step-by-Step Pipeline Results

As mentioned above, the result of every step of the image processing algorithm was computed for schematic A, and will be shown in the following figures.

Figure 4.1 shows the photo taken from a height of approximately 20cm. As the lens of the USB camera has 140 degrees, the previous picture will be calibrated with the result being presented in figure 4.2.

To correct the perspective of the photograph, a homography algorithm was applied with the use of the ArUco markers, as can be seen in figure 4.3 where the perspective is comparable to the one in the reference schema, as intended.

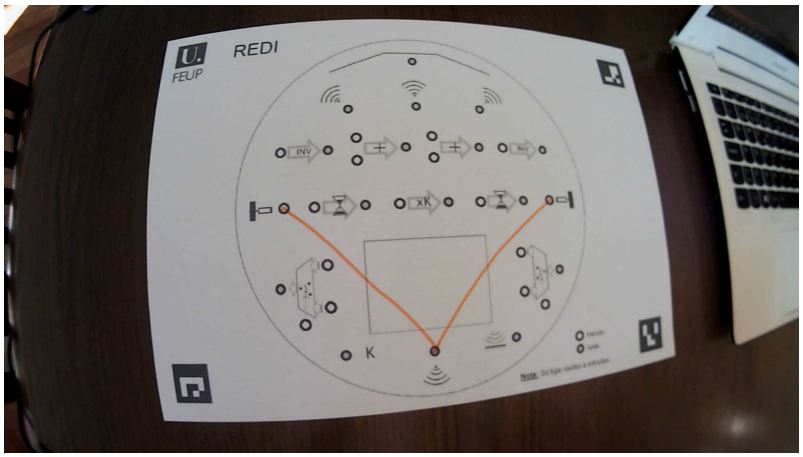


Figure 4.1: Schematic A, Step 1: photograph of the connection diagram

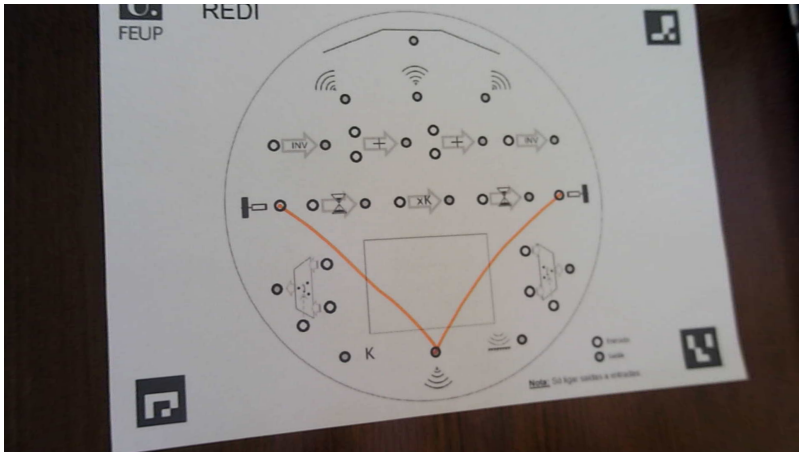


Figure 4.2: Schematic A, Step 2: after calibration

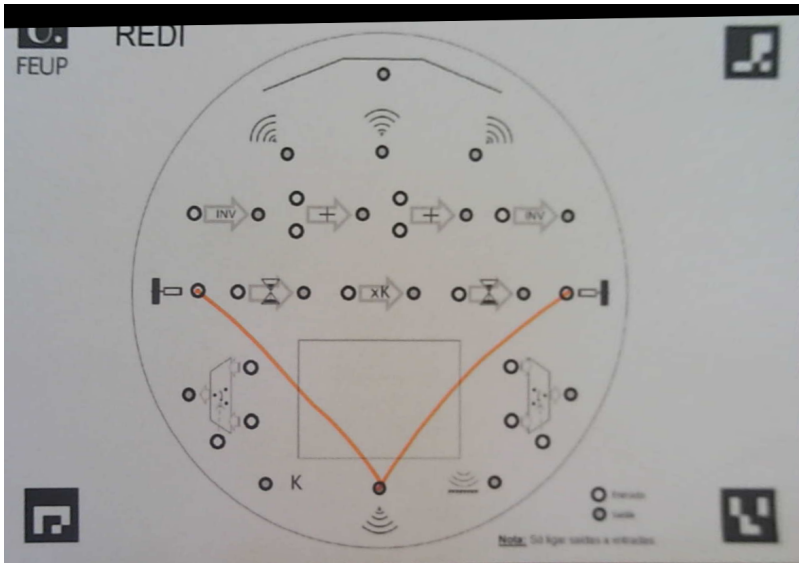


Figure 4.3: Schematic A, Step 3: after homography

Figures 4.4 and 4.5 show the detection of the connection points at different stages of the script. The first one is the result of the polygon approximation method where contours that may resemble circles are chosen. As this is not fail-safe, meaning that some of the circles may not be found by the algorithm, an extra function was added so ensure that every connection point is identified. Since schematic A represents a fairly simple design, the circles are all detected with the first method.

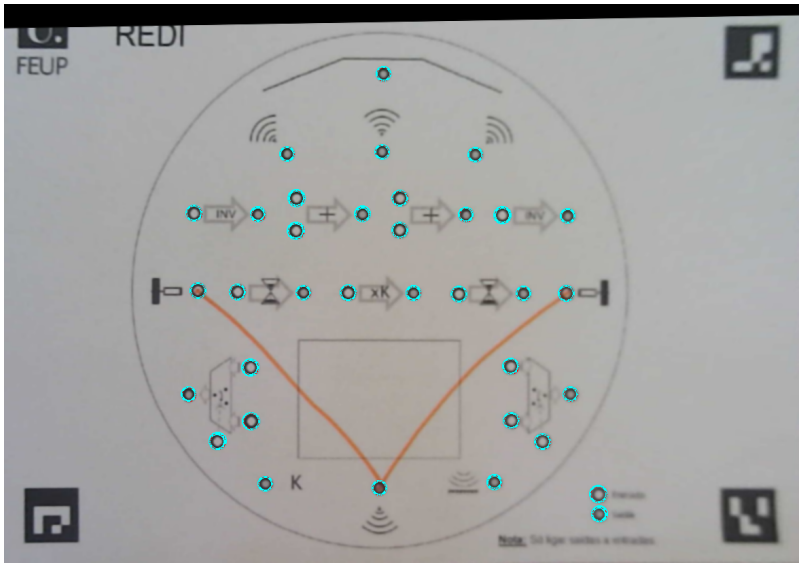


Figure 4.4: Schematic A, Step 4: detection of connection points with polygon approximation

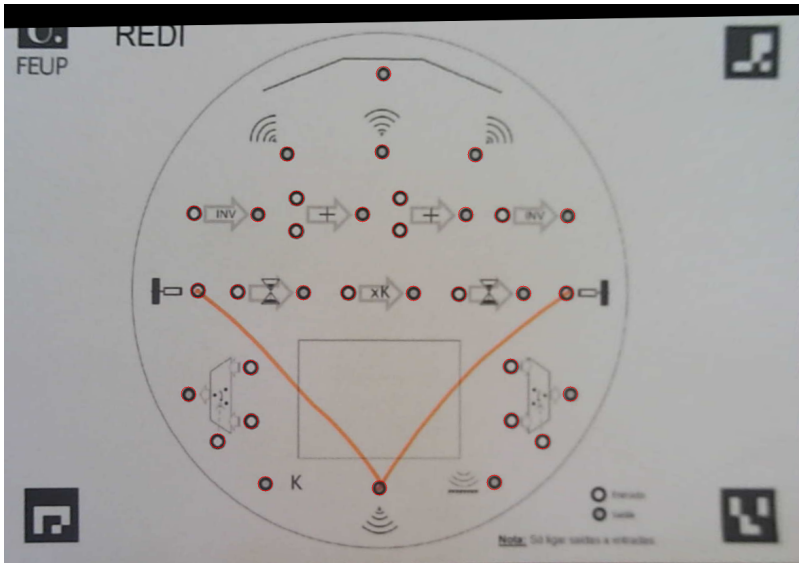


Figure 4.5: Schematic A, Step 5: extra detection of missing connection points

To extract the lines drawn by the user, a colour filter was applied and the outcome was displayed on a blank background as shown on figure 4.6. As the result is thinner and more disconnected than the true lines on the sheet, they were dilated and smoothed over to achieve some measure of continuity, as it can be seen on figure 4.7.

Figure 4.6: Schematic A, Step 6: detection of lines

Figure 4.7: Schematic A, Step 7: lines after treatment (dilation and blurring)

Figure 4.8 is the skeleton of the previous image, which means that pixels were removed from it starting from the edges until no more pixels can be erased without altering the shape. This way, the lines are described with a thickness of approximately 1 pixel.

After processing everything, the final connections are deducted and presented on figure 4.9 in the reference schema with straight lines of different colours to distinguish between them.

4.3 Pipeline Performance with Complex Validation Cases

To guarantee that the image processing algorithm works for more difficult cases than the shown above, five more schematics (B, C, D, E and F) were put to the test. All of them were captured at

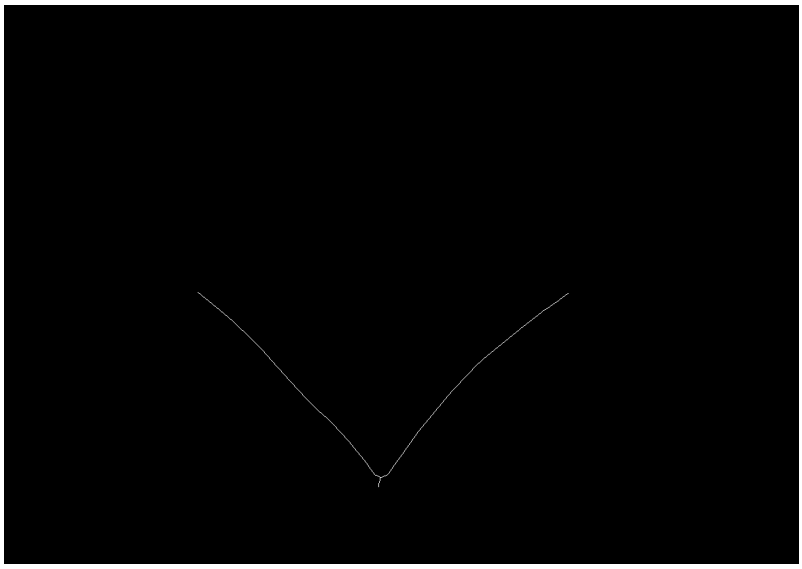


Figure 4.8: Schematic A, Step 8: skelleton of the lines

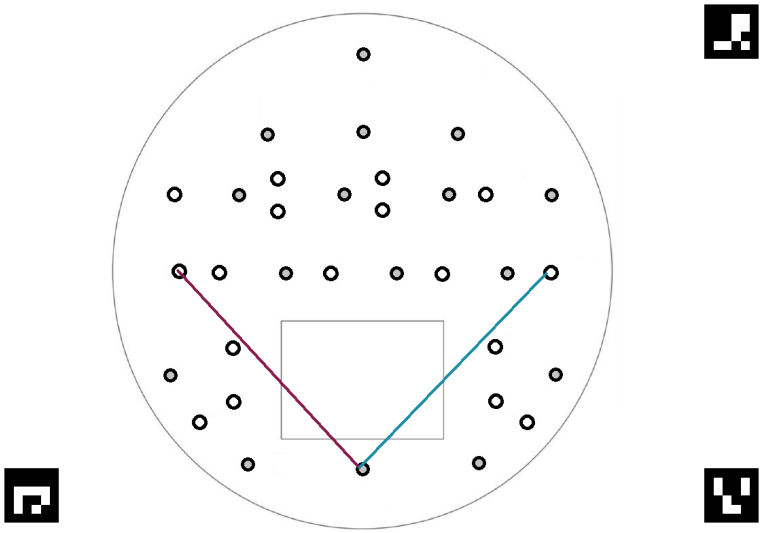


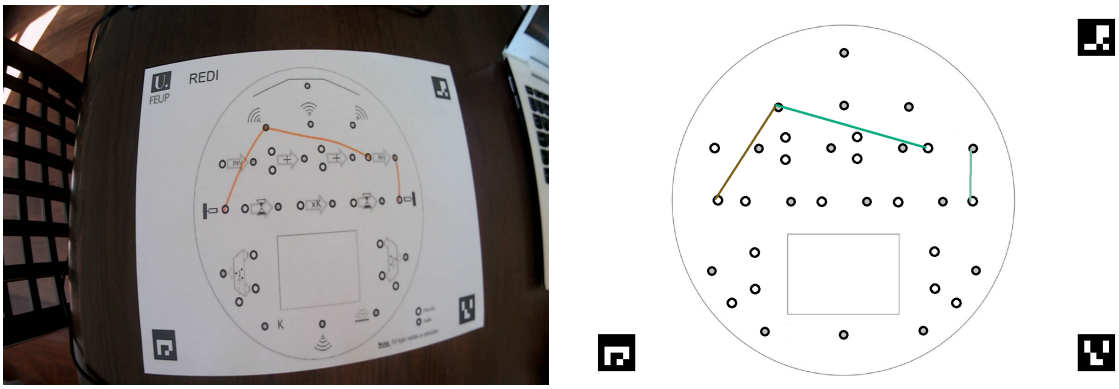
Figure 4.9: Schematic A, Step 9: Reference image with connections detected, final result

a height of approximately 20 cm.

Design A (figure 4.1) was chosen because it was considered simple enough to be a base example. Even though there are two connections, the fact that there is only one output circle for two inputs and that the lines have completely different slopes makes inferring the connections simple enough for the algorithm, as it can be seen on the previous section.

Schematic B (figure 4.10a) was a natural progression from A, as there is a similar connection to the previous example but with an added unconnected line. As the script processes one contour at a time, it was necessary to verify that the separate link was also analysed. As can be seen on figure 4.10b, the experiment was a success.

The connections of model C (figure 4.11a) are still accessible, but the long length of one of

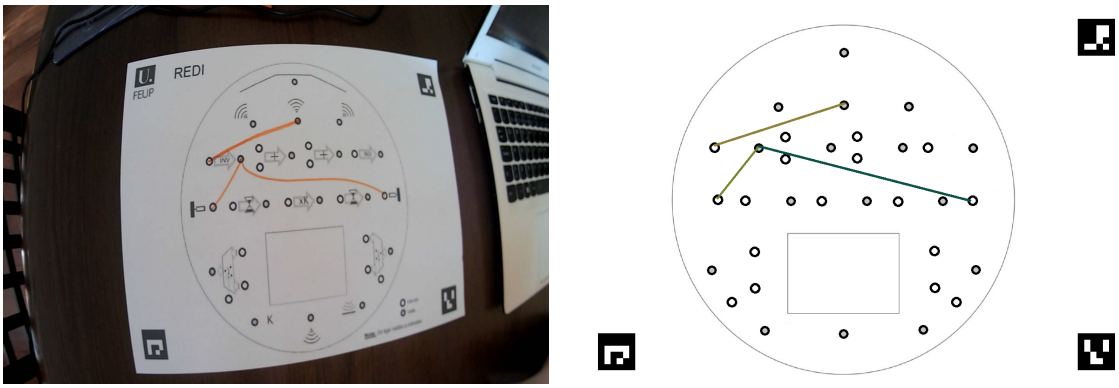


(a) Photo taken

(b) Reference image with connections detected by the algorithm (brown, green and light green straight-lines)

Figure 4.10: Schematic B

them makes it interesting to study. One concern was the possibility that the dilation process would make the line intersect another circle, and that the resulting connections would consequently be wrong. Presented in figure 4.11b is the reference schema with the algorithm's detected lines.



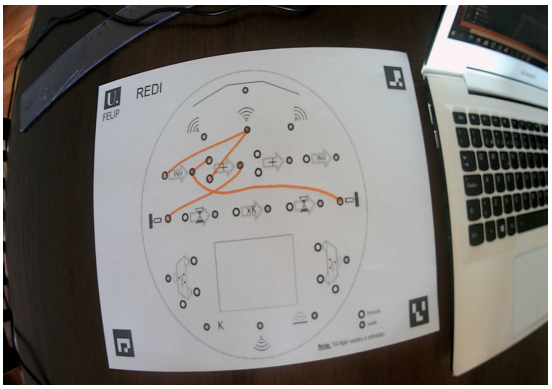
(a) Photograph of the connection diagram

(b) Reference image with connections detected by the algorithm (brownish, green and dark blue straight-lines)

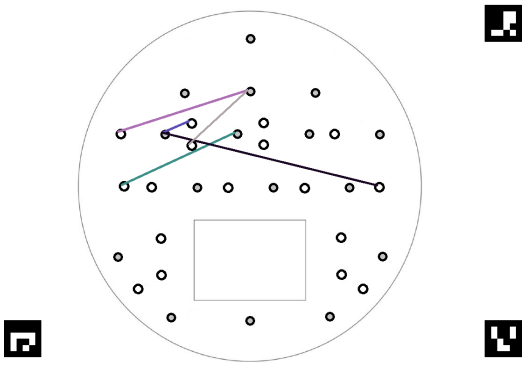
Figure 4.11: Schematic C

To up the ante, schematic D (figure 4.12a) was drawn. It has a four way intersection and had several connections that could be misinterpreted. Even so, figure 4.12b shows that every line was correctly detected.

Next, schematic E (figure 4.13a) was drawn to verify if there would be a problem because of the similar slopes of the lines both on the left and on the right of the paper, together with their closeness. The results shown on figure 4.13b demonstrate that every connection was well detected as expected. The right side was easily resolved because, as before, there was only one output circle for two inputs. The left did not have the chance to go wrong because the lines did not intersect at any point, and so were analysed as separate contours.



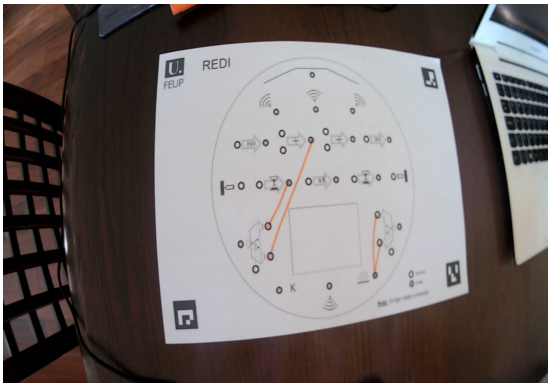
(a) Photograph of the connection diagram



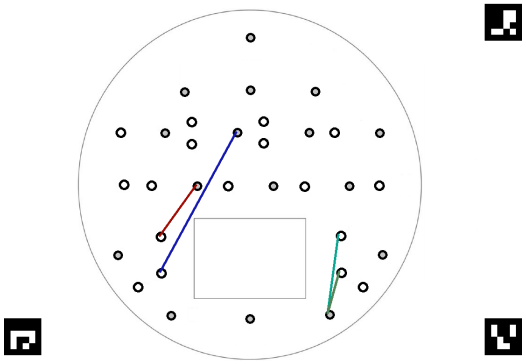
(b) Reference image with connections detected by the algorithm (pink, purple, grey, green and black straight-lines)

Figure 4.12: Schematic D

To rectify that oversight, model F (figure 4.14a) was drawn in a similar way to E but with the left lines intersecting, to force them to be examined in the same contour. But the result seen on figure 4.14b demonstrates that the algorithm was successful and the connections were detected correctly.



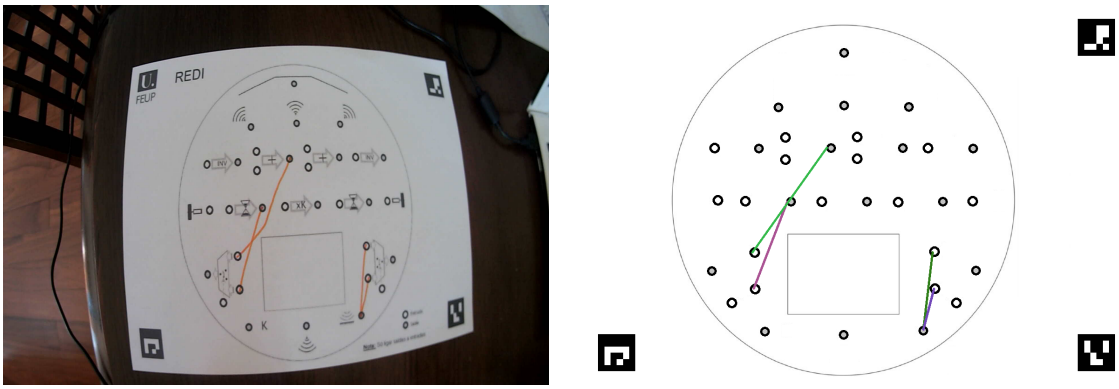
(a) Photograph of the connection diagram



(b) Reference image with connections detected by the algorithm (red, dark blue, light and dark green straight-lines)

Figure 4.13: Schematic E

These are some examples that highlight the capacity of the algorithm to recognise and adapt to the different designs that can be imputed by the users. The algorithm proves to be robust, given the constraints it was designed for. Further considerations on how to properly draw-on the connections and overall best practices for success will be later discussed in section 4.7.



(a) Photograph of the connection diagram

(b) Reference image with connections detected by the algorithm (green, pink, dark green and purple straight-lines)

Figure 4.14: Schematic F

4.4 Distance Analysis

In order to study at what distance the photograph should be taken, various images of schematic A were taken and processed at different heights. Starting at around 10 centimetres it was clear that it was not possible to capture the entirety on the paper sheet. Going up to 20 centimetres the results were very good and are demonstrated on section 4.2.

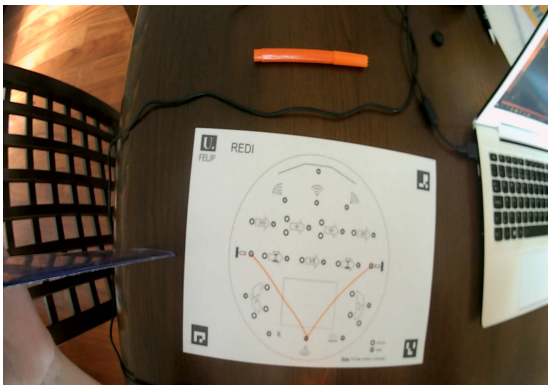
Taking a picture at 30 centimetres resulted in figure 4.15a. Even though it caused the image to be a little more blurred around the edges (figure 4.15b), the algorithm was still able to process it, detect every circle (figure 4.15c) and establish the right connections (figure 4.15d).

Going up to 40 centimetres (figure 4.16a), the program was able to calibrate the photograph and apply the homography method but, as can be seen on figure 4.16b, the image lost too much quality and became too blurry for the algorithm to be able to detect the circles.

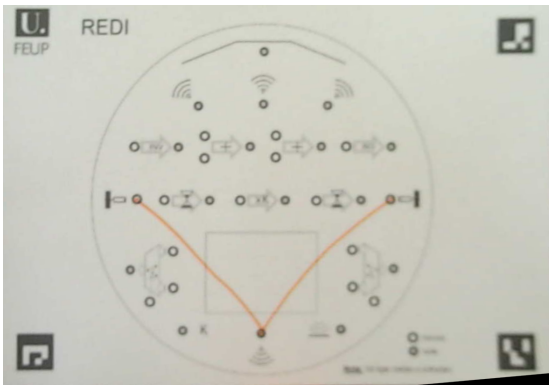
In conclusion, the ideal distance for the photograph to be taken would be between 20 to 30 centimetres to guarantee that the image after homography has enough quality to be processed. It is important to remember that some quality (quantified in amount of pixels) is always lost in calibration and homography: therefore, the initial photograph should be taken in the best possible conditions of light.

4.5 Runtime

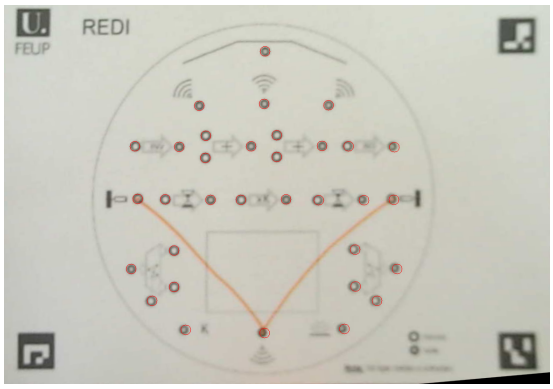
While the complexity analysis was being performed, the opportunity to examine the runtime of each schematic was taken. As demonstrated in table 4.1, every model was executed 10 times; each runtime was noted and the average value calculated. It is important to note that every time was measured in milliseconds, and that it does not include capturing the photograph, as that depends mainly on the user.



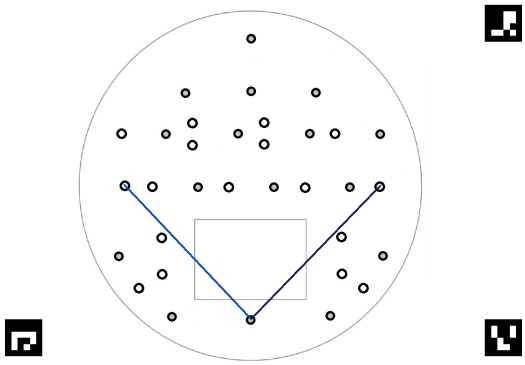
(a) Photograph of the connection diagram



(b) Image after homography

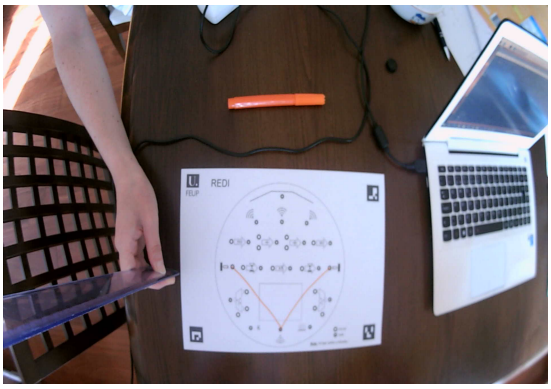


(c) Detection of connection points

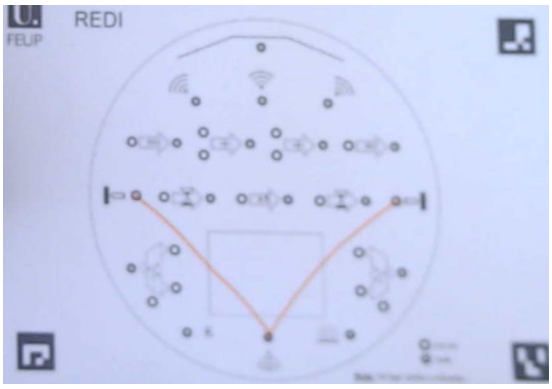


(d) Reference image with connections detected by the algorithm (blue and black straight-lines)

Figure 4.15: Schematic A: Photograph of the connection diagram captured 30 cm from the paper sheet



(a) Pphotograph of the connection diagram



(b) Image after homography

Figure 4.16: Schematic A: Photograph of the connection diagram captured 40 cm from the paper sheet

Examining the average values of each schematic, the increase in runtime can be mainly attributed to two factors: the number of contours extracted and the complexity of the model. Schematic A only has one contour and is a straightforward design, so it makes sense that it also has the lowest runtime. Schematics B, C, D and F all are composed by two contours, and the execution times

Table 4.1: Runtime (in milliseconds)

	A	*B*	*C*	*D*	*E*	*F*
Run 1	2344.9	2618.6	2780.2	2765.4	3035.5	2725.9
Run 2	2624.9	2719.6	2583.8	2731.6	2811.9	2851.1
Run 3	2298.5	2554.0	2679.1	2684.5	3273.2	2844.2
Run 4	2254.4	2513.5	2547.2	2862.6	2998.1	2616.7
Run 5	2506.5	2953.4	2569.6	2716.6	3124.1	2619.4
Run 6	2326.4	2519.9	2568.5	2686.1	2875.3	2648.7
Run 7	2279.6	2566.4	2674.1	2705.0	3043.1	2904.0
Run 8	2633.7	3011.4	2552.2	2765.5	2977.2	2859.3
Run 9	2321.5	2522.1	2940.5	2657.1	2912.6	2683.5
Run 10	2285.2	2488.3	2749.8	2724.0	2885.3	2618.5
Average	2387.6	2646.7	2664.5	2729.9	2993.6	2737.2

are in accordance with the conjecture in section 4.3, stating that the complexity of each design increases from A to F. Model E, even if it is simpler than trial F, has three contours to analyse; so, it is expected for it to take the longest to execute. Thus, a correlation between compute time and number of contours (translated into amount of non-intersecting groups of lines) is demonstrated.

4.6 Colour and Line Thickness Analysis

The colour of the pen used for drawing and the thickness of its line make a difference in the detection of the connections. The algorithm for line detection consists of a filter that removes any pixels with a colour representation in the HSV colour space lower or higher than [0, 70, 120] and [179, 255, 255], respectively. In this case, this means that every colour with a saturation higher than 70 and a lightness higher than 120, will be detected.

To examine which pen colours are acceptable, every colour from a standard 12 pack of pens was drawn on a reference sheet. As can be seen on figure 4.17, two columns were drawn. On the left side, every line has approximately 0.6 millimetres (around 4 pixels) of width and was made by a pen with a thin tip. On the right, a thicker tip was used for a width of around 1 millimetre (approximately 7 pixels). These values are not exact to every line, but are their average.

Figure 4.18 shows the detected lines after the dilation and blurring processes. As is expected, even though the colours from each presented column are very similar, the thicker strokes are more easily detected. Even so, two lines were identified on the left column: red and orange. As these colours were detected on both thinner and thicker strokes, a red pen or an orange one should be used when programming the robot to achieve the best results.

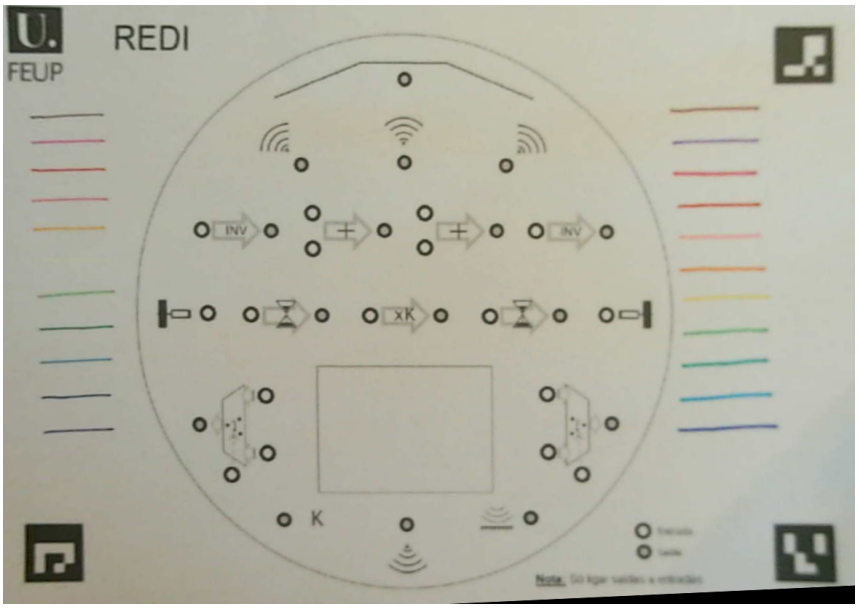


Figure 4.17: Picture of scheme with lines drawn from multiple pens

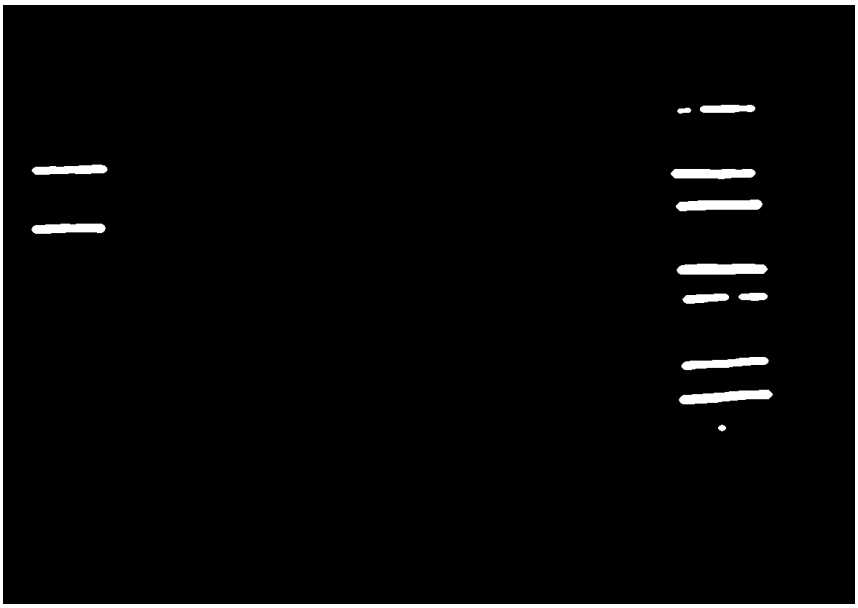


Figure 4.18: Lines recognised by the colour detection algorithm

4.7 Discussion

With the previous experiments, it was demonstrated that the image processing algorithm works when a coloured pen with a tip that draws with a thickness superior to 1 millimetre is used. A few complications occurred when lines passed over the black edges of circles or when going over a grey symbol, but if the line is wide enough then that problem does not happen.

It is important to note that in this dissertation several guidelines were established for the image processing algorithm to work efficiently. It is considered that the only thing that can be connected

to an input is an output, and that it can only be one. The lines, when being drawn, can not touch other circles, so it is fine for the line to be curved as long as it is not excessively and unnecessarily.

An experiment on different ambient light sources was not officially made, but during the semester it was observed that natural light is the best for taking the photographs and that desk lamps should be avoided because of the glare they give; overhead lights also work well.

Finally, it is recommended that every photograph should be captured with the paper sheet on a hard surface by picking up the robot and turning it upside down. Even though it may work to hold the paper down to the camera, good results are not guaranteed as it will cast a shadow on the picture and possibly deform it excessively.

Chapter 5

Conclusions

The main purpose of this dissertation is to have a behaviour of a robot that is defined by drawings on paper. To this end, a robot *REDI5* has been prototyped. The robot features a well known Raspberry Pi 4 with two cameras. It also uses a ESP32 microcontroller to read two encoders and the drive to motors. One of the cameras will be used as virtual sensors while the other is used to capture the image that will define the behaviours. The firmware inside the robot reads the encoders for additional precision of the motor controls. The motors are controlled by PWM signals of the ESP32 microcontroller using an H-bridge. The Raspberry Pi 4 and the ESP32 communicate by serial port, thus allowing the behaviour read from the cameras to actually move the robot. With all electronics, cameras, motors, batteries and chargers accounted for, the budget of this robot rounds the 260€.

This dissertation focused on the hardware preparation and on the vision system dedicated to the recognition of the connections drawn on paper that will in turn dictate the behaviour of the robot. The software uses the OpenCV library to identify connections that will be later decoded into behaviour for the robot. Several realistic sample programs were used as validation cases including situations with lines crossing and some deformation. This deformation can come from the lens of the camera and from misalignments of the camera to the target piece of paper.

The processing operations include correction for distortions caused by the lens, homography for perspective correction, colour image processing for circle and line detection and connection logic detection. The processing time is typically less than three seconds for a camera with a resolution of 1920x1080.

5.1 Future Work

It is necessary to implement the Raspberry Pi 4 camera as virtual sensors for the information obtained to be what controls the robot. This data would be continuously read and fed into the executing program that actually implements the wanted behaviours by calculating the outputs based on operations with the available signals.

On the reference paper in figure 3.10, there are two blocks with hourglasses and one other that has a *K* letter. The first represent timers and the last is an operation that includes a constant defined as *K*. An algorithm to read these values should be implemented.

A way to make the debugging process easier and visible to the user would be a great addition to this work. This could be done by using LEDs and/or by allowing the user to connect to the robot through the VNC platform. Optimizing execution time and making the image processing algorithm more robust so that shadows become a lesser problem when capturing the photograph is also a possible secondary upgrade.

The purpose for the future is to implement a similar robot with the same objectives, but with a modular paper sheet. The hope is to have multiple disconnected blocks, each with an operation, sensor or actuator. When programming the wanted behaviour for the robot, the user would then select only the desired blocks and draw the connections on them. This would allow more freedom to have as many inverters or adders in the final sheet as it would not have a fixed structure.

Bibliography

[1] Bruno Pires. Plataforma robótica multi-funcional. Master of Science Dissertation in Universidade de Aveiro, 2008.

[2] Armando Sousa, Bruno Moreira, Filipe Lopes, Hugo Costa, and Susana Neves. Attractive demonstrations with wire programming robot "redi". In *2015 10th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6. IEEE, 2015.

[3] Vasco Manuel Nunes Pinto. Robot com comportamento definido por fios virtuais e interface em tablet. Master of Science Dissertation at FEUP, 2015.

[4] D. Wang, C. Zhang, and H. Wang. T-maze: A tangible programming tool for children. 2011. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-79960267123&doi=10.1145%2f1999030.1999045&partnerID=40&md5=2b0e4d355ba0a26a9cf44e0496d05bf9.

[5] Daniel Queirós da Silva. Computer vision system for tactode programming. Master of Science Dissertation at FEUP, 2020.

[6] Ângela Cardoso. Tangible language for educational programming of robots and other targets. Master of Science Dissertation at FEUP, 2019.

[7] J. Vega and J.M. Cañas. Pibot: An open low-cost robotic platform with camera for stem education. *Electronics (Switzerland)*, 2018. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85062513448&doi=10.3390%2felectronics7120430&partnerID=40&md5=d0f04eb43a0316ab4bbf76af3c8cd540.

[8] Ana Rafael, Cássio Santos, Diogo Duque, Sara Fernandes, Armando Sousa, and Luís Paulo Reis. Development of an alphabot2 simulator for rpi camera and infrared sensors. ROBOT 2019, 2019.

[9] J. Lima, P. Costa, T. Brito, and L. Piardi. Hardware-in-the-loop simulation approach for the robot at factory lite competition proposal. ICARSC, 2019. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85068441491&doi=10.1109%2fICARSC.2019.8733649&partnerID=40&md5=db3f5e0cef0873e587cf36d4006afba7.

[10] João Ferreira, Filipe Coelho, Armando Sousa, and Luís Paulo Reis. Bulbrobot – inexpensive open hardware and software robot featuring catadioptric vision and virtual sonars. ROBOT 2019, 2019.

[11] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014. URL: https://www.sciencedirect.com/science/article/pii/S0031320314000235, doi:https://doi.org/10.1016/j.patcog.2014.01.005.

[12] Simon Just Kjeldgaard Pedersen. Circular hough transform. *Aalborg University, Vision, Graphics, and Interactive Systems*, 123(6), 2007.

[13] H. Yuen, J. Princen, J. Illingworth, and J. Kittler. Comparative study of hough transform methods for circle finding. *Image Vis. Comput.*, 8:71–77, 1990.

[14] Tongjie Y Zhang and Ching Y. Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984.