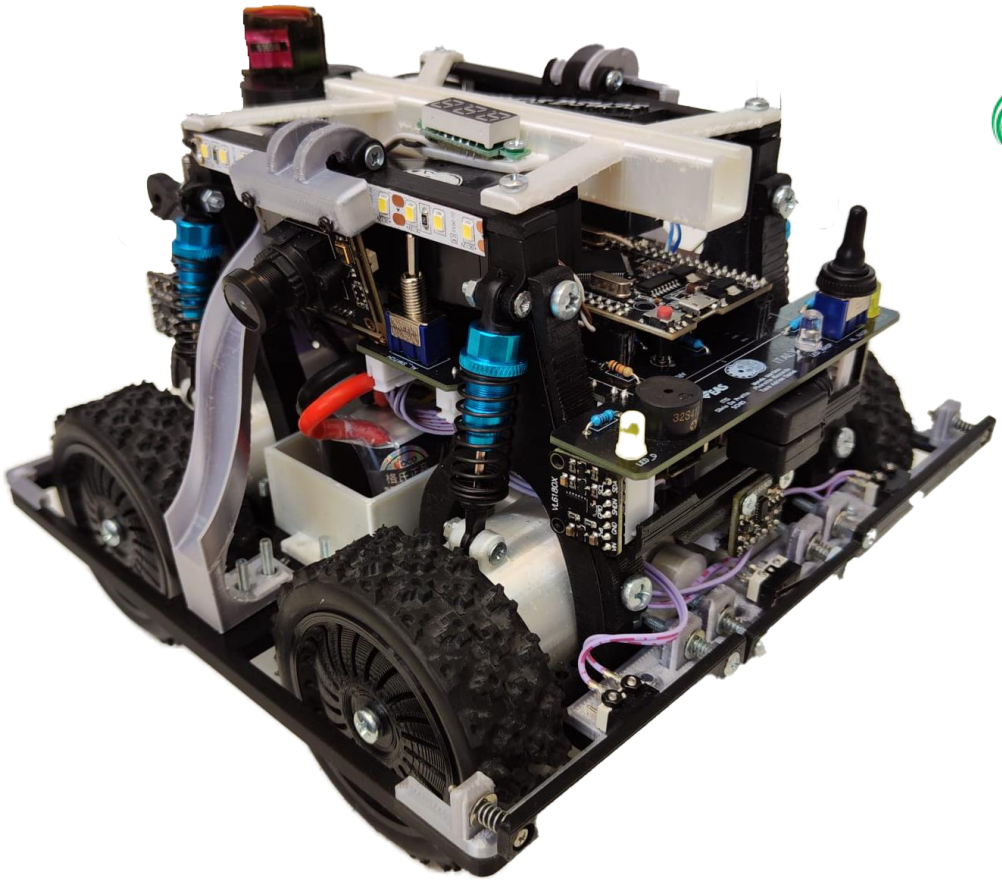
SENSA SCHEI

TEAM DESCRIPTION PAPER



Rescue Maze subLeague robot
BLACK EMINENCE

Tuca Adrian Daniel – Mendo Martino – Pivirotto Riccardo
ITIS De Pretto Schio, ITALY

Abstract

- Our robot is designed and built to be versatile so that it can adapt to any kind of situation it encounters. This is made possible by the rigid structure and suspension system, which makes it easier to overcome the obstacles it will face within the maze. Through cautious mechanical and electrical design, it was possible to compact the entire robot into a minimum size, yet ensuring maximum accessibility and maintenance. Finally, through rigorous programming efforts, including an effective PID system, the robot is able to navigate any path with the utmost precision, thus being able to recognize any victim it encounters, using the two cameras placed on its sides.

1. Introduction

a. Team

- We are a team of three, and we divided the various tasks according to our knowledge and passions:
  - Tuca A. Daniel (captain): Hardware, Assembly, Sensors, Testing
  - Mendo Martino (co-captain): Design, Software, Testing
  - Pivirotto Riccardo (co-captain): Software, Algorithms, Mapping, Testing.

2. Project Planning

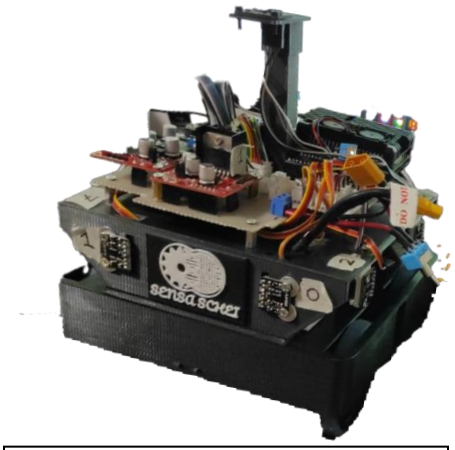
a. Overall Project Plan

- The main reason why we decided to undertake this course is definitely to improve our knowledge, especially computer and electronic knowledge, but also to better understand the importance of team planning and team work, which are fundamental for this kind of project. Besides competing and maybe winning, we want to encourage young and old to approach the world of robotics, which is fundamental nowadays, motivating them to create new inventions applicable in the field for which this competition is also designed, RESCUE.

- Being our first experience, creating an immediately ideal and fully functional design was very hard. Therefore, we decided to make a structure that was easy to assemble and then in case modifiable, using components and sensors that were easy to replace and program, leaving expansion ports open on the PCB for other components not initially considered. Even at the programming level, we have tried to create codes that can be easily modified to fix problems that may arise. It was mainly thanks to the teams of our school who have participated in the competitions the past few years that we have understood many of the problems these robots can have and it was easier to identify them in our robot and solve them.

- Our main goal is definitely to make the robot the best ever, making it able to go through any maze without ever missing a movement or a victim and to do so in the shortest time possible and without unnecessary movements. We then wish to carry on the name our school has created for it by trying to get as many rewards and trophies as possible. Finally, our wish is to transfer the expertise and skills gained during this experience in our future working field.

- The first version of our robot gave us the opportunity to understand the various problems related to mechanical and electrical design. Initially built on a rigid base with no shock absorbers and a too high center of gravity, we discovered how easily the robot could get stuck in the obstacles of the maze, eventually toppling over with a simple staircase. Another serious problem turned out to be the electronic board that had been created on a common breadboard using stiff wires and tin to make the various connections: communications between the various devices and sensors often failed, making it impossible to be programmed. We soon realized that it was because of the continuous disturbances generated on the motherboard due to the work done by hand. Finally, because of the way the entire robot had been created, any regular or extraordinary maintenance was very difficult. This led us to a radical change, redesigning the robot from scratch, adjusting the center of gravity and adding suspensions, while for the motherboard, a PCB was designed and then printed by a specialized firm.



First version of the Robot “The Factory”

- Since this is the first year we are participating in these competitions, we have no previous experience other than our schoolmates who won the world championship two years ago. Thanks to them, we were able to understand from the very beginning some of the problems related mainly to the cameras, and we also guessed what were the first steps to be taken in order to create a functional algorithm for maze mapping.

b. Integration Plan

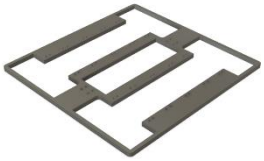
- The robot was initially designed via special software for both the physical and electrical components. This made it possible to manage the available space in the best way, while also ensuring subsequent hardware expansions. To ensure the efficiency of the whole system, individual components that connect externally to each other were used to guarantee immediate replacement in case of failure.

3. Hardware

- The robot was built from the beginning in a 3D virtual environment, so that we could best study the available space and ensure a solid structure, giving each sensor and component its appropriate space. We gave the structure modularity for later assembly and disassembly, to ensure convenience in access to all sensors and for further addition of components not initially planned. The structure consists of a base and 4 columns that support the weight and maintain the balance of the entire system. To ensure the modularity mentioned earlier, most of the pieces are small, easily removable without interfering with other components already installed. The size of these parts was also limited by the fact that, while using the 3D printer, we wanted to waste as little PLa material as possible at the same time making it as efficient as possible. For the best functioning of this system, the various mechanisms that make it up, for example the suspension system and the release of the kits mechanism, were also designed to be as simple as possible, while still ensuring that they function together with more complex mechanisms.

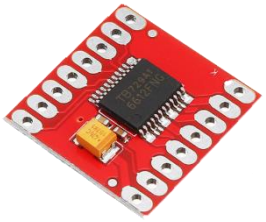
a. Mechanical Design and Manufacturing

- BASE: it is the largest piece of the robot, printed in PLA 20x20cm large and 5mm thick, it helps with the attachment of the motors and various components of the robot. Designed to also provide protection to the wheels, surrounding them with its outer edge, thus helping the robot not to get stuck or hooked on external objects.



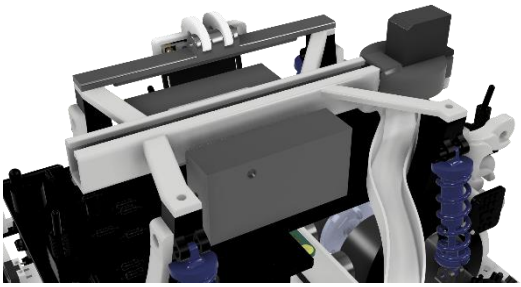
- SUSPENSION: As mentioned initially, with the first version of the robot we encountered problems in overcoming obstacles due to the stiffness of the robot itself. Therefore, with the second version we immediately decided to introduce a suspension system for each present wheel. These suspensions are hooked between the high end of the support that surrounds the motors (which are also hooked to the base through bearings to give them mobility) and the top of the support columns. In standard functioning, i.e., when the robot is resting on flat ground, the shock absorbers are already in compression, so that overcoming an obstacle by elevating one of the wheels, none of the other wheels will risk spinning out of control since with maximum spring extension it will be able to touch the ground. This ensures smooth overcoming of any obstacle and a better distribution of forces within the robot.

- MOTORS: Our robot uses four 12-volt JGB37-520 DC motors with gearmotors that drive each of them at 110 rpm. They are equipped with incremental encoders that count the rotations of each motor, enabling us to derive the individual speed. With the use of the PID system we are then able to keep an identical speed between them. These motors are managed through two TB6612FNG motor drivers that allow us to send two digital signals to indicate the direction of movement and a PWM signal for the low voltage speed, then converting it into the voltage that is applied to the ends of the motor. In addition, there is a 5-volt servomotor on the robot that, managed via PWM signal, allows the rescue kits to be moved to the correct slide on release.



TB6612FNG driver

- RESCUE KIT RELEASE: For the release of the kits once a victim has been found, we have designed a simple mechanism so that no glitches can occur in the release phase. In fact, it consists of a horizontal channel placed on top of the robot, which stores the 12 kits and immediately after them a cap to which a rubber band is hooked that pushes these cubes up to the slot of the servo motor, thus keeping this slot always charged thanks to the force of the rubber band. As they are released, the servo motor moves the cube to the slide on the affected side thus facilitating their fall. The cubes have also been designed: the sides of each of them have been blunted to lessen the bounce effect thus ensuring that it falls into the victim's area.



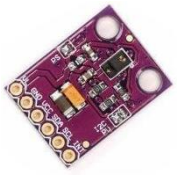
b. Electronic Design and Manufacturing

- SENSORS: We decided to install various sensors on the robot, so that it would have total knowledge of its surroundings thus helping its orientation and obstacle detection:

  - VL6180X: We mounted 6 of these sensors, one in the front and back and two on the sides. They are laser proximity sensors and detect a distance up to 255mm which is more than enough for our purpose, and they are connected to the I2C network of the Raspberry Pi 4. We chose these sensors because they have high accuracy, they are not affected by external factors such as temperature, and most importantly they are very convenient to program and manage.

  - BNO055: triaxial gyroscope, accelerometer with 9-axis magnetometer, used in the IMU (Inertial Measurement Unit) mode, which calculates orientation in space, as Euler angles, from the acceleration data of the gyroscope and accelerometer. We chose this gyroscope because of its ease of use via I2C communication and the breadth of data it can provide us. Through this gyroscope we understand how we are positioned in the maze, perform perfect 90-degree turns, and detect rises and falls.

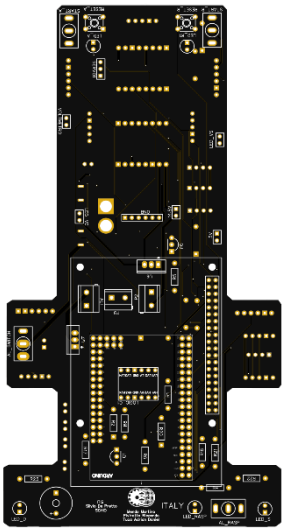
  - APDS9960: is the color sensor that allows us to distinguish blue, black, and silver plates to choose the movements that the robot should then perform. It communicates through the I2C protocol with the Raspberry and was chosen after testing a vastness of sensors, as it is the fastest to detect the correct value and does not vary too much with different lighting, making it the safest choice to undertake.



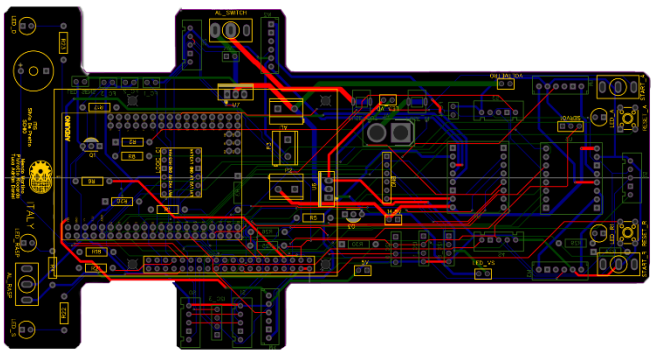
- MICROPROCESSOR/MICROCONTROLLER: The robot is equipped with two types of boards. The first is the Raspberry Pi 4 microprocessor, which is in charge of reading and managing all the sensors logic, detecting victims through the cameras placed on the sides and sending through I2C communication the commands to move the robot. It was chosen because of its great capacity in data processing, video signal management and in interfacing with sensors of all kinds.

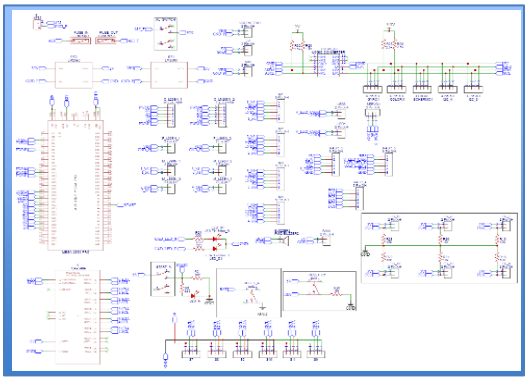
An Arduino Mega 2560 Pro, which performs speed control through the PID system, was used for motion management, i.e., motor control. It was chosen for the amount and stability of PWM signals that facilitate programming and for its small size, which makes it fit perfectly to our robot. As mentioned earlier, the two boards communicate through the I2C protocol, where the Raspberry acts as the Master while the Arduino acts as the Slave, creating a series of commands and functions necessary for the robot's movements.

- PCB: For the connection of all the sensors and the two boards, we designed a "motherboard" which was then sent to a specialized company for printing. It was made in such a way as to occupy in the most efficient way the limited space we had and to be the most electrically safe. In fact, it consists of several JST-HX connectors to connect all the sensors individually while the components that are directly connected on the PCB are easily removable and replaceable in case of failure, not having to change the whole motherboard. The PCB is 1.6mm thick and consists of four layers, which contain all kinds of tracks needed by the robot, from motor power circuits to signal communication. As previously explained in section 2.a with the initial manually tinned board with stiff wires, it was difficult to communicate between the two boards assuming interference between Low-voltage signals and the High-voltage circuit that powered the motors as the cause. Therefore, special care was taken with the new PCB to keep these types of circuits separate, confirming our assumption, as no more interference occurred.



- POWER SUPPLY: The entire circuit is powered by a 14.8V 2300mAh LiPo battery, which is interrupted through a main switch and then connects directly to the motor drivers and the two XL6019 buck-boosts, which lower the voltage at 14.8V and 5V to power all the sensors and the two boards. The motors are then powered at 14.8V but thanks to the PID controller they are no longer affected by exceeding a certain voltage threshold. There is a second switch on the PCB, which acts as a startup for the Arduino code without ever turning it off, so the whole circuit can continue to be kept on without the motors starting to move the robot. Finally, the whole circuit is protected with a 6A fuse connected before the main switch.

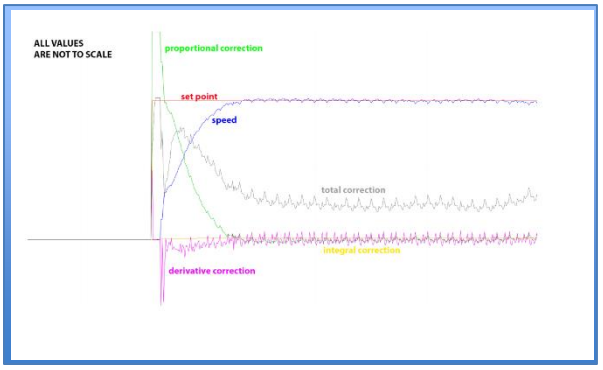




4. Software

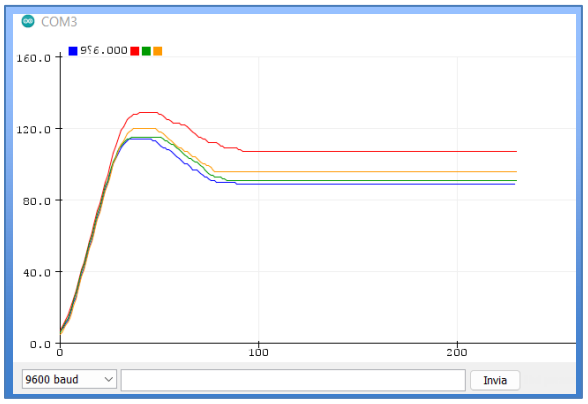
a. General software architecture

The robot software is divided into two separate parts, one found on the Arduino which manages the motors through PID control and the other on the Raspberry which manages all the logic. The two softwares communicate with the I2C protocol as previously mentioned.



PID calculation on single motor

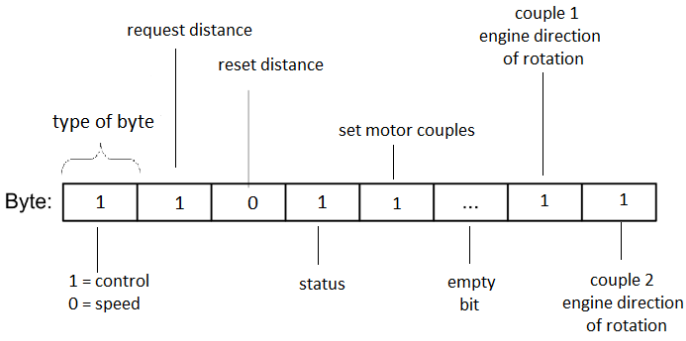
- ARDUINO: is programmed entirely in C through the Microchip Studio IDE application for AVR programming, used in such a way as to have better management of the accessible ports and registers of the microcontroller. It is used exclusively to manage the 4 motors on the robot and the movements that need speed control, as well as to provide better control of the movements. For this reason, we used motors with encoders mounted on them, which are useful for measuring the speed, so as to regulate it so that it is synchronous between the motors. To achieve this synchronization, we used software-based Proportional-Integral-Derivative (PID) control, which allows the desired speed to be kept constant. The 4 motors are independent, and for each one the value is calculated with the control. The PID is a negative feedback control that acts as a correction network; the process acquires the instantaneous motor speed as input and compares it with the reference value, derives the error as subtraction of the two and proceeds to calculate the three corrections, proportional, derivative and integral.



Speed curve of the 4 motors

- RASPBERRY PI: The robot's main code is structured on a multitude of libraries and secondary codes containing the classes (movements, servomotor, cameras, limit switches, mapping) that flow into one main code, which allows them to run in parallel thanks to "asyncio," "multiprocessing," and queues, parallel processes that manage to communicate with each other. For example, the "movements" class, which includes everything needed to move the robot autonomously, from reading the sensors to sending data to Arduino, waits for a positive result from the cameras when they detect a victim in order to stop in time.

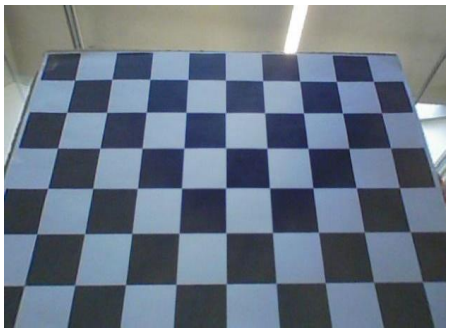
- SERIAL COMMUNICATION: Communication between Arduino and Raspberry Pi takes place via an appropriately managed serial bus, it is a 1,000,000 baud protocol, 8 data bits for transmission, 1 parity control bit and 1 bit stop. The communication is bidirectional, but only the Raspberry Pi makes the requests to the Arduino, and it responds with the desired values. The data protocol has been structured exactly on the desired data. The Arduino is programmed in such a way that if it receives nothing for more than a second it blocks the motors by setting their speed to 0.



- MOVEMENTS: The robot is designed to be managed entirely from raspberry by transmitting motor direction and speed information continuously in order to allow state-based programming within the Arduino code. This allows fast adaptation to external events such as speed bumps and obstacles.
  To keep the robot as straight as possible, an additional control was added: PID control on the gyroscope. It allows the robot to re-establish its course in case of any deviations found within the maze, allowing incredible stability in terms of accuracy and positioning.

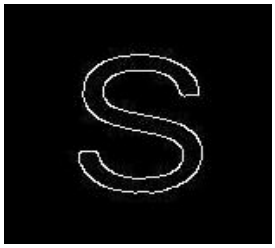
- CAMERAS*:* For victim detection within the map, we used two USB100W04H cameras with wide-angle lenses, placed one on the right side and the other on the left side of the car. By then connecting these to the raspberry pi and using the OpenCV library, we were able to capture the images, analyze them, and detect any colors or letters. The 180° lens was chosen to see as much as possible inside the maze, however it has a high fisheye effect, which by twisting the shape of the victims found, makes them difficult to be recognized. For this reason, we decided to calculate matrices to apply to the captured image, so as to 'straighten' it and improve the analysis. To obtain these correction matrices, it was necessary to take about thirty photos of a black-and-white chessboard, detected through the function "cv2.findChessboardCorners." Then, using another function of the OpenCV library, "cv2.fisheye.calibrate," we were able to calculate the corrections necessary to make the image orthogonal, that is, with all lines parallel and perpendicular to each other. Finally, by means of the functions "cv2.fisheye.initUndistortRectifyMap" and "cv2.remap()," we apply these matrices to each image captured by the cameras.

Distorted image (original)

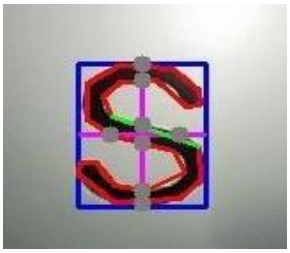


Straightened image (recalculated)

- FINDING LETTERS: For the letters, on the other hand, the 'straightened' image is converted to shades of gray and blurred, and then we apply the "cv2.Canny" function, which performs a binarization of the image, converting the pixels of the figure outlines to white and the remainder to black. We then derive the vertices of these contours again through the function "cv2.findContours" and calculate the area for each. If one of these is between two given values, thus indicating the probable presence of a letter, the associated contour is saved. By means of the latter we then use the function "cv2.boundingRect" to construct a rectangle around the letter and its directrices, and the function "cv2.approxPolyDP" to make a polygon that echoes the shape of the letter. Next, we go on to find the longest side of this polygon, calculating its angle of inclination and the difference between the vertical side of the rectangle and it. Thanks to these two values we are able to hypothesize what the present letter might be. At this point we go on to analyze the number and position of the color change points in the intersection between the directrices and the letter. Again, as in the previous step, each letter will have its own characteristics, which vary to a limited extent. Therefore, by going precisely to 'exploit' the differences between one victim and another, we are able to identify the detected letter with sufficient accuracy.



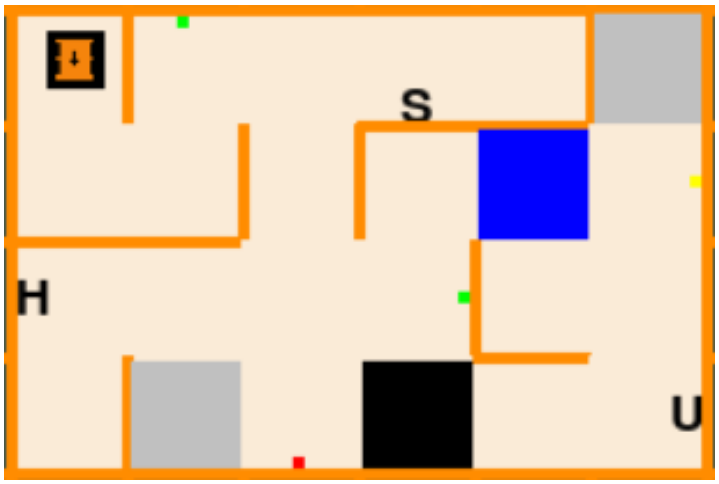
"Canny" function



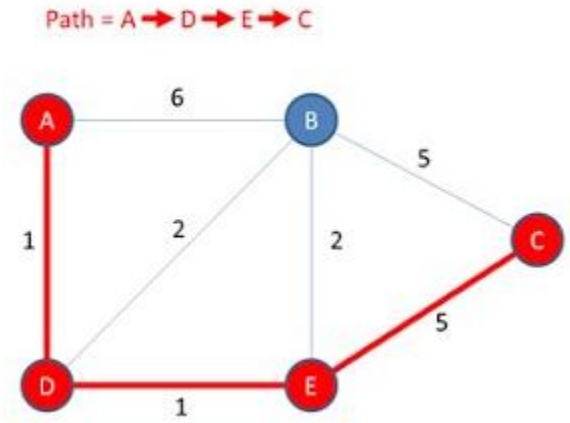
Final analysis

- DETECTION of COLORED VICTIMS: Having then obtained the 'straightened' image, it is necessary to check whether colored victims or letters are present. As for the former, the image is initially converted from RGB to HSV (Hue, Saturation, Value). This color space describes colors in terms of hue (saturation or amount of gray) and brightness, so in order to achieve the most accurate result possible, white LEDs were applied. Then, through separately realized thresholds and the function "cv2.inRange," a binarization of the image is carried out, transforming the pixels within the thresholds to white and the rest to black. We made a threshold for each color to be detected (red, yellow and green), which is composed of two HSV colors, one minimum and one maximum, so we would have three binarized images. Next, after slightly blurring this last one, we use the function "cv2.findContours" so as to get a list with the vertices of the detected contours. Finally, we calculate the area for each contour, and if this is greater than a given value (so as to avoid erroneous recognition), the color is identified according to the binarized image under analysis.

- MAPPING: Within the maze the robot must be able to move completely autonomously, so since we cannot know how the maze will be structured, it was necessary to introduce a mapping algorithm. Through the latter, we can then record in a data set the part of the maze that is explored as the robot goes along and choose the most efficient path to follow. In the first competitions we participated in, the robot used the "right hand rule" to move, that is, if possible, he always gave the right-hand priority. This method is viable in rather simple mazes, thus without cycles or other complexities. So, we had to introduce an algorithm that would allow the robot to move efficiently, thus exploring unvisited cells and following minimal paths. It is therefore used from the very first movement the robot makes within the maze.



- ALGORITHM: The algorithm chosen is Dijkstra's, since it allows searching for the minimum paths in a graph with or without sorting, cyclic and with positive weights on the arcs. In our case, the maze corresponds to the graph, the cells to its nodes, and the weight of each arc is worth 1 (although some exceptions were added in the process). Initially, all reachable cells, i.e., those that do not have at least one wall, are taken and checked which ones are still unexplored, and then saved in an array ("visitable_cell"), so that the shortest path among them can be calculated and taken. In case there are no cells to explore, the shortest path back to the starting cell is calculated, again using Dijkstra's algorithm. The first cell within "visitable_cell," which will be the "target," is then taken and a dictionary ("unvisited") is created in which each cell in the map will get a value equal to plus infinity, while the starting cell will take value 0. This number represents precisely the weight of each arc, i.e., the 'distance' to be traveled to reach the respective cell, so it will be modified during the course of the algorithm. It then begins a while loop which will end when there are no more cells in the 'unvisited'. Within it the cell that in the "unvisited" possesses the minimum value is taken, which will be defined as the "currCell," so the first one will be the starting one. Next, the "currCell" is saved in a second dictionary ("visited"), in which all the cells examined will end, and if in addition this one is equal to the "target", the while loop will be terminated. Otherwise, the four walls of the "currCell" are analyzed, so as to figure out where the robot might move. So, for each wall that is not present, the respective adjacent cell is taken, which will be renamed "childCell," and checked that this is not present within "visited," as this



would create errors in the case of dead ends. Then the distance required to reach the latter is calculated by increasing the value of "currCell" by 1, and if this is less than the value that the "childCell" possesses in "unvisited," it will be assigned that. However, there are some exceptions, for example in case the "childCell" is a black plaque, a very large number will be added to its new value so that the robot will be prevented from passing it again. Also, according to the previous condition in a third dictionary ("revPath"), used to realize the final path, the "childCell" is saved with the respective "currCell". Finally, the "currCell" is deleted from the "unvisited" and the cycle begins again. You will eventually get as many arrays as there are cells in the "visitable_cell," each of which will contain the path to reach them for which the shortest one will be chosen from among them.

b. Innovative solutions

- Many solutions have been tried, including unsafe and practical ones that have led to different results, helping us to get where we are now, especially in programming, where we only discovered while writing codes how a programming language really works.
- A pivotal point of our project is used to search for the simplest and consequently most innovative solution to accomplish the required task: the VL6180X distance sensors were connected through an I2C multiplexer in order to ingeniously circumvent the problem of sensor addresses, which were not easily changeable.
- Even with cameras, solutions have been sought that can ease the workload, and one of these is definitely the function that allows correction of the image distorted by the fisheye lens.

5. Performance evaluation

- Due to the short time we had for the creation of this robot (a little more than 6 months), we did not have a chance to thoroughly discover and solve all the existing problems, in fact it is lacking in many aspects but still manages to perform its task with minimal errors, always managing to finish its race. We are already working to fix these problems and make our robot even better-performing than all past robots, and we are ready to earn the title of world champions in the shortest possible time.

6. Conclusion

- As this report isn't meant to accurately document all the work, in order to fully answer all your questions, you can direct links to the full documentation, source code, all images and videos and our GitHub page in the appendix. We also believe that sharing our work with the RoboCup Community can also turn out to be useful for future projects.

Appendix and References

- **GitHub page for source code:**

  Click here or scan QR code

- **EasyEda PCB schematic and traces:**

  Click here or scan QR code