RoBorregos Soccer Open

Yaír Reyes Coronado Jocelyn Anahid Velarde Barrón Mauricio Degollado Saucedo

***Abstract*—The RoBorregos Soccer Open project details the design and development of autonomous soccer robots by a Mexican robotics team, RoBorregos, for competitive participation in international RoboCup events. The team, comprising electronics, software, and mechanical specialists, leverages advanced technologies including hyperbolic mirrors, sophisticated vision systems, and precision mechanical components. Key innovations include a custom-designed Teensy 4.1 microcontroller integration for enhanced processing, and novel approaches in dribbler and kicker designs for optimal gameplay. The project's comprehensive development is documented from initial design challenges to final implementations, encompassing detailed mechanical, electrical, and software innovations tailored for robust performance in competitive soccer robotics.**

Contents

I Introduction to team 1
I-A Team Background . . . 1
I-B Project management . . . 1

II Overview of the robot 2
II-A Design, Capabilities Strategies . . . 2
II-B Materials: . . . 2

III Mechanical 3
III-A Mirror Design . . . 3
III-B Dribbler . . . 4
III-B1 First design . . . 4
III-B2 TMR competition dribbler . . . 4
III-B3 Final version . . . 4
III-C Vision . . . 4
III-C1 First design . . . 4
III-C2 TMR Vision . . . 4
III-C3 Final Version . . . 4
III-D Kicker . . . 4
III-E General Design Over Time . . . 4

IV Electrical 5
IV-A Choosing our main micro controller . . . 5
IV-B Line detection . . . 6
IV-B1 Choosing the best resistance . . . 6
IV-C Kicker . . . 7

V Software 7
V-A Communication . . . 7
V-B Control . . . 8
V-C Logic . . . 9
V-D Vision . . . 10

VI Reflection 11

References 11
VI-A Links . . . 11

I. Introduction to team

A. Team Background

We are RoBorregos, a Mexican robotics team established in 2015 at Tecnológico de Monterrey (ITESM). Comprising approximately 40 undergraduate students, we actively participate in various national and international robotics competitions, including RoboCup Junior and RoboCup Major.

This year's soccer team includes Yaír Reyes, our team leader who manages the electronics; Jocelyn Velarde, who is responsible for software development; and Mauricio Degollado, who handles mechanical design.

Fig. 1. Photo of the team. From left to right. Mau, Jocelyn and Yaír

RoBorregos is participating in the Soccer Open category for the second time, with our first participation being in Canada in 2018. This new generation of the team drew inspiration from that experience to design this year's robots. Since January, we have been diligently working on our robots, focusing our research and development efforts in all the areas.

B. Project management

To organize all of our tasks we decided to use all of GitHub's capabilities. We started by placing all the

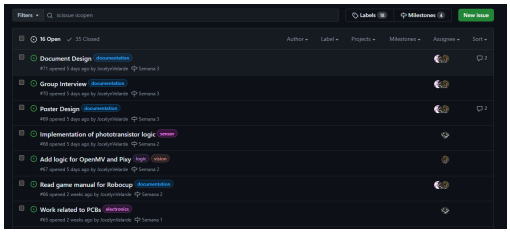

Fig. 2. Issue registration on GitHub

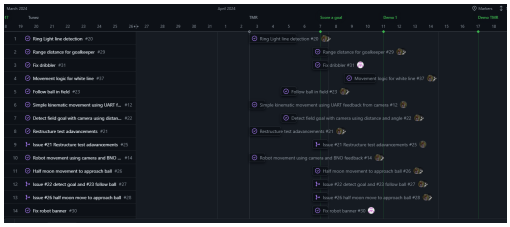

Fig. 3. Milestones and Gantt board

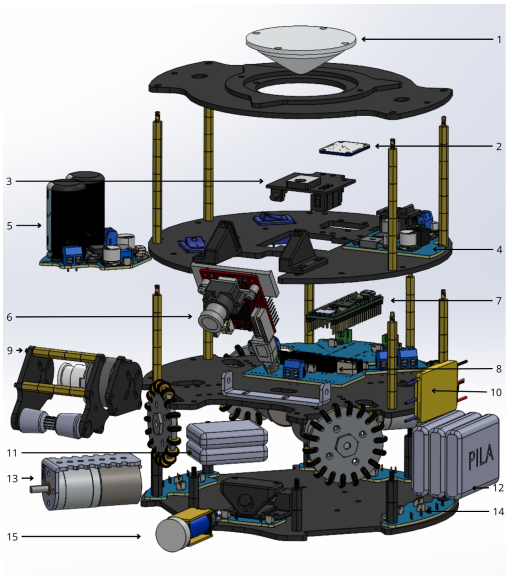

Fig. 4. Exploded View

issues, writing a description, title and corresponding tag. As shown in figure 2. This issues are linked to specific milestones to keep everything organized on the Project tab for our Kanban board.

II. Overview of the robot

A. Design, Capabilities Strategies

This year, we had two significant iterations of our robots: one for the *Torneo Mexicano de Robótica* (TMR) and another for the final competition in Eindhoven. In this overview, we will discuss the final version of the robot.

Prices in dollars and for each one

1) Hyperbolic Mirror - 20 usd
2) BNO Gyroscope - 35 usd
3) Pixy 2 Cam - 70 usd
4) Power PCB - 12 usd
5) Kicker PCB - 20 usd
6) Open MV Cam - 65 usd
7) Teensy 4.1 - 35 usd
8) Main PCB - 8 usd
9) Dribbler - 20 usd
10) ESC - 10 usd
11) Lipo batteries (Kicker PCB) - 8 usd
12) Main Lipo Batteries - 20 usd
13) 4.4 HP Pololu Motor - 49 usd
14) Line detection PCB - 8 usd
15) Kicker - 8 usd

B. Materials:

- 3D printed PLA or ABS pieces for battery holders and outer covers.
- Main chassis is laser-cut MDF, painted black.
- Aluminum brackets for motor mounts.
- Aluminum dribbler holder.
- 3D printed silicone molds for components requiring friction, such as the dribbler and kicker.
- Assembly is secured with brass hex spacers.
- Brass rods

Descriptive circuit diagram

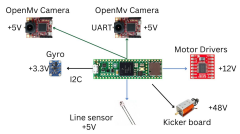

Fig. 5. Descriptive circuit diagram

Logic

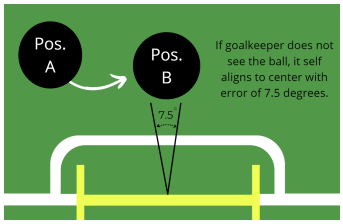

Fig. 6. Goalkeeper logic

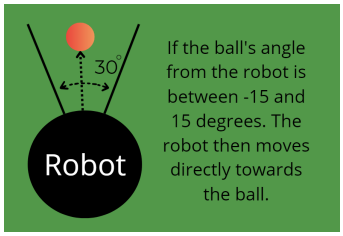

Fig. 7. Direct Approach

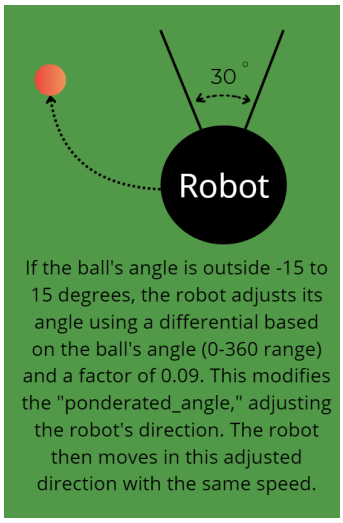

Fig. 8. Adjust Angle

III. Mechanical

A. Mirror Design

Initially, we attempted to thermoform our mirror shape but had to abandon this approach due to the unavailability of a suitable material supplier. In our eagerness to commence testing, we experimented with folding mirror sheets into conical shapes, which yielded unsatisfactory results. Ultimately, we designed a precise model in SolidWorks and collaborated with METALWORK AND STAMPING to access advanced manufacturing machinery. This partnership enabled us to fabricate the mirror from steel and achieve a high-quality chrome finish.

The code linked here was utilized to approximate the visual field achieved by a chromed mirror, considering minor variations due to the chroming process's thickness. The mirror's shape is defined by the equation:

$$y = \sqrt{0.045 + 0.45x^2}$$

Derivative:

$$y = \frac{1}{2} \cdot 0.9x \cdot \left(0.045 + 0.45x^2\right)^{-\frac{1}{2}}$$

Given the height and distance from the camera to the mirror, the code employs the derivative of the mirror shape to calculate the angles and subsequently determine the distance visible at every 0.1 cm increment.

To refine the accuracy beyond theoretical calculations, a linear regression model was employed to correct for potential calculation discrepancies. This dual approach leverages both the geometrical properties of the mirror surface and empirical adjustments, ensuring a more precise estimation of the visual range.

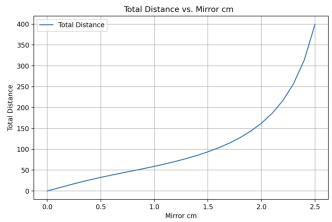

Fig. 9. Mirror Calculation

This is the graph showed by the code comparing centimeters of visibility and "x" axis distance on mirror hyperbola.

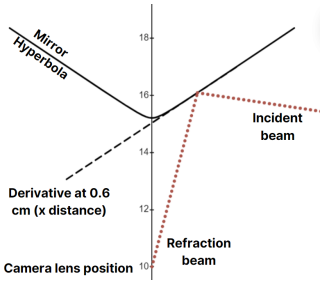

Fig. 10. Calculation at specific distance

We used this graphing calculator to confirm our data from our code at a specific distance of the mirror.

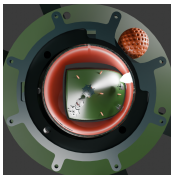

Fig. 11. Mirror simulation in blender

Fig. 12. Test View from mirror sheets

B. Dribbler

1) First design: The initial dribbler was built using a 3D-printed mold, incorporating centering lines, 3D-printed pulleys, and rubber bands. However, due to the insufficient precision of the mold, the self-centering mechanism was discarded.

2) TMR competition dribbler: Our updated dribbler, constructed with aluminum, custom 3D-printed pulleys, and molded dark silicone for optimal grip, maximized the available space efficiently. The design utilized springs to maintain contact pressure with the ball. However, the potential for displacement beyond the allowed diameter remained a concern.

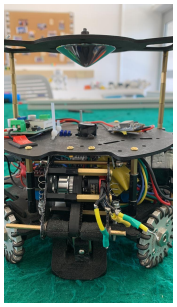

Fig. 13. TMR Dribbler

3) Final version: We decided to reposition the rotation axis of our dribbler to a lower location to ensure compliance with the permitted diameter constraints. Additionally, we upgraded to a more powerful brushless motor, and encased it in a custom 3D-printed cover for enhanced protection and durability.

C. Vision

1) First design: At this stage, we successfully configured the mirror, allowing our camera to capture the entire field without any issues.

2) TMR Vision: Despite achieving full field visibility, we faced challenges with fluctuating lighting conditions, which adversely affected our camera's capability to accurately track the ball in motion.

3) Final Version: Our final version addresses all the camera's visibility issues, resolving the lighting problems. It features two cameras for improved precision and a heavier, more robust design.

D. Kicker

While the core design of our kicker remained largely unchanged, we enhanced our final design by integrating a silicone-molded tip. This modification leverages the frictional properties of silicone to effectively counteract the ball's backspin, resulting in greater kicking distances and higher velocities.

E. General Design Over Time

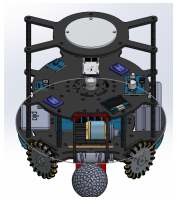

Fig. 14. V1 Idea

lo

Fig. 15. Robot Version for TMR competition

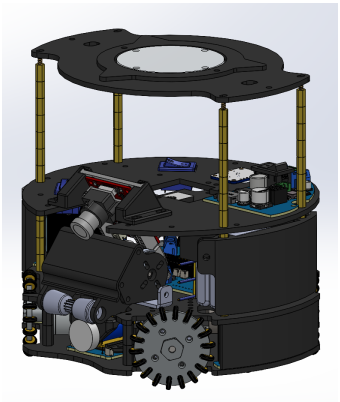

Fig. 16. Final Version

IV. Electrical

For our robots, the following PCBs were designed in EasyEda Pro:

- Main board (MCU board)
- Kicker board
- Line detection board
- Power board

Throughout our journey, we encountered various challenges that had to be overcome to achieve a stable version of the robot:

A. Choosing our main micro controller

For the first iteration of our robot the design started when we selected our MCU, in order to do this we made a research with the main micro controllers that most of the teams use at the international competition. By this we got:

- Teensy (4.0 and 4.1 version)
- ATMEGA2560
- STM32F4 (Different versions)

While the Teensy was the best approach thanks to the integration to Arduino and be the fastest, it was an expensive MCU for the first iteration, meanwhile the STM32F4 were cheaper, we couldn't get a development board in a fastest time.

In search of other MCUs superior to the Arduino (ATMEGA2560) we discovered with a new mexican development board based on the ESP32 and RP2040 micro controllers named *DualMCU ESP32+RP2040 microcontroller board* [1]. The dual integration of these MCUs caught us for choosing this as our main development board for our project.

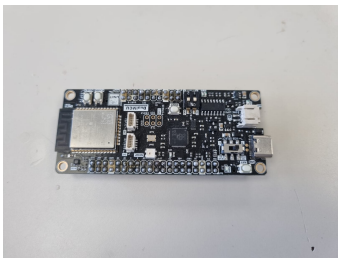

Fig. 17. DualMCU development board

With the microcontroller chosen we started the first version of the main board that had integrated sockets (JST SH and XH), driver for the motors (TB6612FNG) and our new development board, the DualMCU, all embedded in a 4 layer PCB.

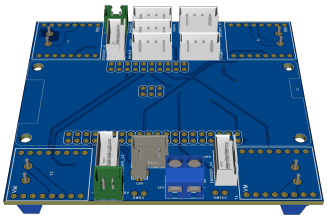

Fig. 18. DualMCU - Main PCB

While this development board worked fine in certain scenarios, in the software we encountered significant software issues with MCU communication, while in the hardware the board was big, needed to be in the edge of the robot due to push some buttons when uploading the code. Moreover, there were inconsistencies in board stability; some units functioned correctly while others did not. Taking these challenges into account, we opted to switch to the Teensy 4.1 as our main MCU. Its compact size and enhanced processing speed offered a more reliable solution for our robot.

With this in mind in the final version of the robot, our main PCB includes a Teensy 4.1 microcontroller, along with JST connectors (SH and XH) for easy integration with other components, as shown in Figure 15. Additionally, we use the TB6612FNG motor driver, with both outputs connected in parallel to meet the current limit requirements (see Figure 16).

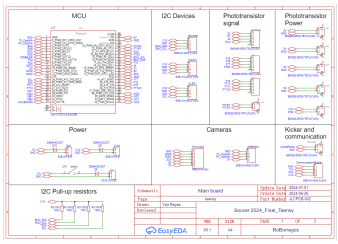

Fig. 19. Teensy schematic for the main board

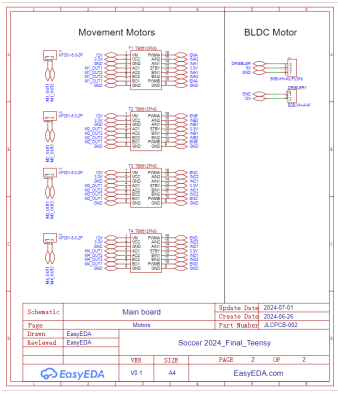

Fig. 20. Motors schematic for the main board

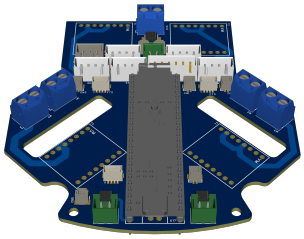

Fig. 21. Teensy - Main PCB

B. Line detection

Be inside the field is one of the main part in order to compete in all the RoboCup's soccer leagues. In order to achieve this most of the teams use three types of sensors:

- Photodiodes
- Phototransistors
- LDR

With all of them been an analog signal output, and just changing the type of connection and led that they choose, however photo transistors are more reliable and are often use as light sensors. Specifically, we opted for the Vishay Semiconductor TEPT5700 due to their reliability and current availability in stock.

1) Choosing the best resistance: The circuit for this type of devices consists in a light source combined with the photo transistors emmisor connected to a current limiter resistance(seen in fig). Crucial to the circuit is the selection of the R2 resistor.

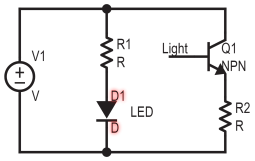

Fig. 22. Photo transistors typical circuit

In our initial robot iteration, we assumed that R2 did not require precise calculation and opted for an experimental value of 10K ohms—a decision that nearly jeopardized our qualification for the world competition. This experimental R2 led to a minimal differentiation between the detected values for the field and white lines, resulting in instability of the robot under different lighting conditions.

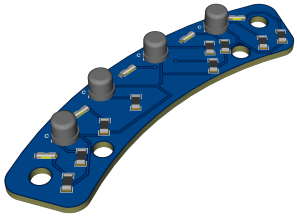

Fig. 23. Photo transistors - Line detection PCB V1

After the almost failure of the TMR we started to calculate R2 getting into an average value of 82K ohms, with the following results, for the white lines a value around 1.77 V, while for the field a value of 0.98 V, values that are easy recognizable for our logic (3.3V) and with a big gap.

We integrated a multiplexer in order to optimize the use of analog pins in the MCU. Also we want to use a Schmitt trigger circuit connected to any interrupt pin making the robot 100 percent reliable thus all the analog signals are proccessed before hand

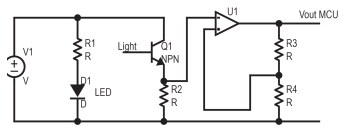

Fig. 24. Photo transistors - Schmitt trigger circuit

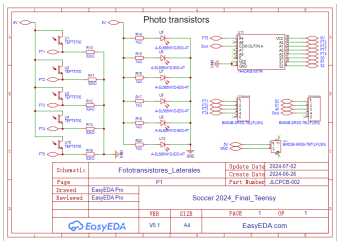

Fig. 25. Photo transistors - Line detection PCB V2

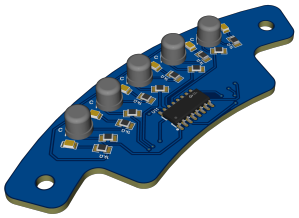

Fig. 26. Photo transistors schematic

C. Kicker

In this category the most popular kicker is an electromechanical one, more specific, a push pull solenoid powered by a lot of voltage and triggered via any circuit or device that has 3.3 V logic. The first iteration of our circuit powered by a 48V capacitors (charged via two boost converter of 24 V connected in series) triggered via a relay. [2]

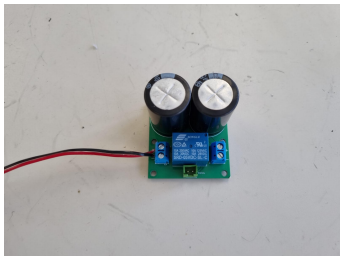

Fig. 27. Kicker board with relay

After conducting further research, we opted to optimize board space by designing a custom boost converter capable of delivering 48V. This PCB includes integrated PMOS circuit protection and an NMOS trigger device, both controlled via optocouplers to enhance stability. We simulate this circuit in LTSPICE for a more controlled testing.

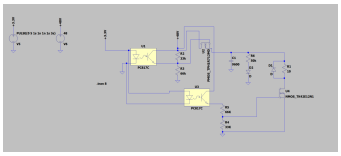

Fig. 28. Kicker circuit

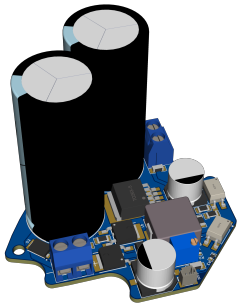

Fig. 29. Kicker board with Mosfet

V. Software

A. Communication

As we started working with a Dual Microcontroller, we had to establish our communication protocols for data transmission. The initial diagram is shown below. Please note that for each channel, we had a specific serial number for sending and receiving data packages.

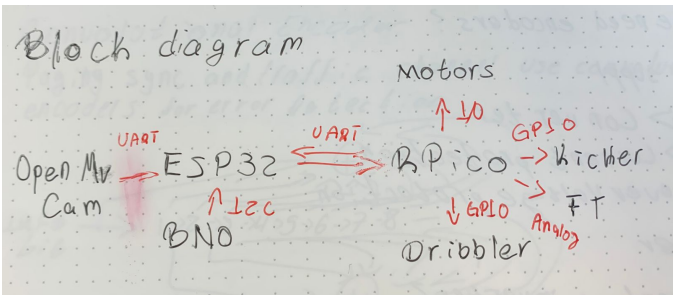

Fig. 30. Communication protocols for dual microcontroller

For the OpenMV camera, we decided to use UART protocol via Serial, sending a String of two decimal figures separated by a comma. Our first approach included parsing the string while we had serial data available. Still, we were facing some unexpected issues, where data was not being sent and received at the same time, even if we had the same baud rate on both ends. We decided that dealing with Strings was not so efficient, as data was slowly processed. To resolve this, we implemented a new

type of logic. Once the data has reached the ESP32 it is processed through the `Receive(uint8_t signal)` method, the algorithm logic is shown below.

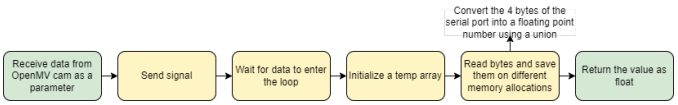

Fig. 31. Data reception on ESP32

We decided to use the union structure as it provides us with an efficient data conversion, allowing direct access to the received bytes without needing explicit bit manipulation. After testing it for a while, everything was working great, but once we incorporated more data variables from our camera, de data packages were to large, and the union was experiencing unexpected bottlenecks.The issue can be seen in the following figure.

```
15:29:03.430 -> MOVE LINE LASTED: 42
15:29:03.430 -> EVERYTHING ELSE LASTED 0
15:29:03.469 -> BNO LASTED: 25
15:29:03.505 -> BALL ANGLE LASTED: 42
15:29:03.539 -> BALL DISTANCE LASTED: 42
15:29:03.624 -> GOAL ANGLE LASTED: 43
15:29:03.663 -> GOAL DISTANCE LASTED: 42
15:29:03.703 -> MOVE LINE LASTED: 42
15:29:03.703 -> EVERYTHING ELSE LASTED 1
15:29:03.703 -> BNO LASTED: 22
15:29:03.742 -> BALL ANGLE LASTED: 42
15:29:03.785 -> BALL DISTANCE LASTED: 42
15:29:03.817 -> GOAL ANGLE LASTED: 44
15:29:03.894 -> GOAL DISTANCE LASTED: 43
15:29:03.941 -> MOVE LINE LASTED: 42
15:29:03.941 -> EVERYTHING ELSE LASTED 0
15:29:03.941 -> BNO LASTED: 22
15:29:03.981 -> BALL ANGLE LASTED: 42
15:29:04.028 -> BALL DISTANCE LASTED: 42
```

Fig. 32. Issues with data reception and mismatch in time spent

Our approach for debugging this issue consisted on counting the milliseconds spent on every task, with this we were able to find out where our bottleneck of data was forming. And it was getting stuck on the union, to solve this we decided to take an easier approach, as we only had a few weeks before the TMR competition, and there were plenty of tasks to get done, other than just focusing on the serial communication. At last, we implemented the following logic shown below.

```
float m1 = cos(((45 + degree) * PI / 180));
float m2 = cos(((135 + degree) * PI / 180));
float m3 = cos(((225 + degree) * PI / 180));
float m4 = cos(((315 + degree) * PI / 180));
```

Fig. 35. Kinematic equations

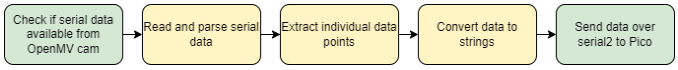

Fig. 33. Final data handling over serial on ESP32

As for receiving this data package on the Raspberry Pi Pico over serial, we used the following algorithm logic shown below.

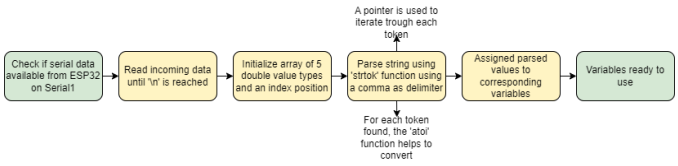

Fig. 34. Data handling on Raspberry Pi Pico

Note that to reach this smooth data transmission, we had to go over 2 different approaches for data handling. For the TMR we used the final logic explained above, and for Eindhoven we are keeping the same logic, as we have reached its optimal implementation.

B. Control

To make the robot work we used two linear controllers of type PID one for the control of the angular velocity and the other for linear velocity. Each of these controllers were determined empirically trough different iterations, and according to the best performances the constants were tuned. For the angular velocity controller it was prioritized that the robot was able to fetch the ball even if there was overshoot, that is why we gave more weight on making our response time so that the robot will be able to fetch the ball independently of its position.

In order to make our robot to move holonomically we implemented the following inverse kinematic equations.

See the figure below to show the diagram of our robot.

We also implemented our own library for class creation. Utilizing the yaw position we were able to calculate setpoint and error for out PID controller, below is the logic followed to match angles on the robot frame

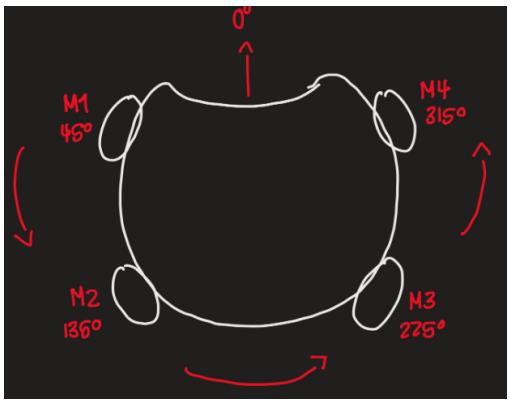

Fig. 36. Robot diagram for wheels

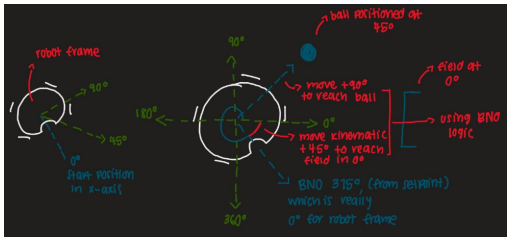

Fig. 37. BNO diagram direction for movement

and the real world. Note that yaw is calculated using the Euler vector for x.

C. Logic

For the algorithm design, which is the main logic, we utilized two main files. One for the esp32 and another one for the Raspberry Pi Pico. Our logic hierarchy follows this structure.

Goalkeeper

- ESP32_Goalkeeper
- Pico_Goalkeeper

Striker

- ESP32_Striker
- Pico_Striker

Fig. 38. Main scripts organization

For the goalkeeper logic we started with a simple approach, using a Bang Bang controller, which consists of tracking the ball using our camera and adjusting position to maintain its centered view. We started by defining an arbitrary `bang_bang_threshold`, which means that the robot will move 'x' degrees from the center. The algorithm is shown on the following code chunk.

```
if(cam_angle < (0 - bang_bang_threshold)){
    myMotors.moveMotorsImu(270, 190, speed_w);
    Serial.println("LEFT");
}
else if(cam_angle > (0 + bang_bang_threshold)){
    myMotors.moveMotorsImu(90, 190, speed_w);
    Serial.println("RIGHT");
} else {
    myMotors.moveMotorsImu(0, 0, speed_w);
    Serial.println("STOP");
}
```

Fig. 39. Algorithm for Bang Bang controller

This algorithm allowed us to always center our robot in relation to the ball, still we were excluding some variables. For a second approach, we used our camera to find the center of the goal, then assign an error margin `goal_threshold` to determine how close the robot needs to be to the center of the goal before it stops moving. So this basically means the following.

1) If the goal is to the LEFT that means (angle ¡ 185 - threshold), the robot moves left to center itself.
2) If the goal is to the RIGHT that means (angle ¿ 185 + threshold), the robot moves right to center itself.
3) If the goal is within the threshold range, the robot STOPS moving translationally.

As an overview, the Goalkeeper algorithm follows the logic on the diagram below.

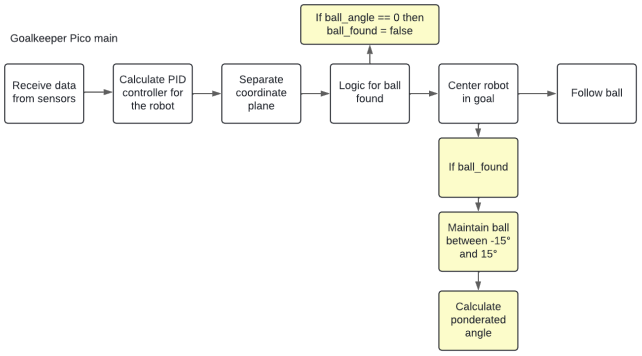

Fig. 40. Goalkeeper algorithm

Observe how is the adjust angle and ponderated angle used to determine the robot's actions, shown in the diagram below.

D. Vision

For the vision system we started developing using Micropython on our OpenMV IDE. For detection we first had to set some constants using VAD values to detect the orange blob, we also added a brightness, saturation and contrast filter to make the image obtained from the camera more clear. The first step when using the camera includes reducing the vision field by placing a black blob to reduce image noise. As shown below.

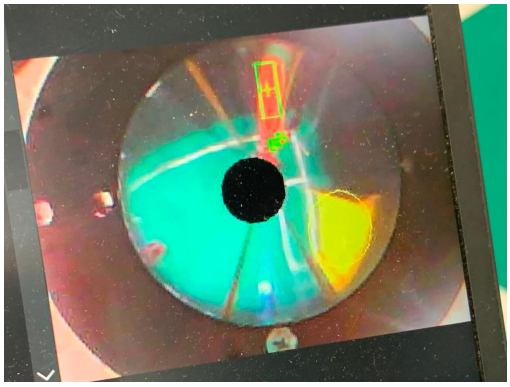

Fig. 41. Robot POV to reduce noise

Still, we were getting a lot of light coming in to the camera from the outsides of the frame, to solve this we applied even more black image filtering as shown below, to only leave the mirror vision.

Fig. 42. Enter Caption

We also located the center of the frame using constants such as `FRAME_HEIGHT`, `FRAME_WIDTH`, `FRAME_ROBOT` to find each exact position of our 3 blobs (yellow goal, blue goal, orange ball).

We created the `locate_blob` method that used `find_blobs` from the OpenMV IDE to locate blobs in the image snapshot for each specific threshold value set and returns a list of blob objects for each set. In the case of the ball we have set the `area_threshold` to 1, this small value means that even when a small area of this color is detected it will be marked as an orange ball to increase the amount of inclusivity in our vision field. For the case of the goals the `area_threshold` is set to 1000 because goals are much more larger in size. Once we have our different blobs detected we can calculate the distance to the blob using the hypotenuse. First we need to calculate the relative center in x and y axis, then find the magnitude distance and finally using an exponential regression model to calculate the real distance with pixel comparison. Note that the expression was obtained by running measurements comparing cm and pixels using a proportion and data modeling on Excel, the procedure is shown below.

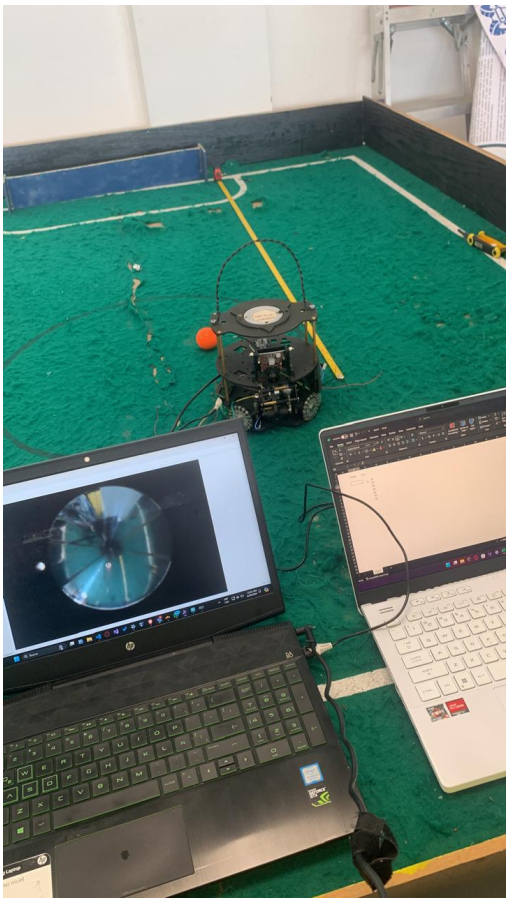

Fig. 43. Pixel regression to calculate distance in cm

```
magnitude_distance = math.sqrt(relative_cx**2 + relative_cy**2)
total_distance = 11.83 * math.exp((0.0245) * magnitude_distance)
```

Fig. 44. Pixel expression

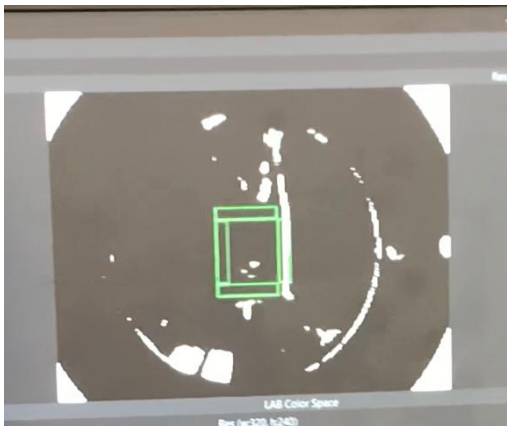

Fig. 45. Line limitations from robot's POV

At first we tried to use the function obtained from the manufactured of the mirror, but we where getting a lot of outliers, that is why we decided to trial and test our own regression model. The final expression calculation is shown below.

For the `goal_distance` we needed to calculate the opposite distance, using sine. To calculate the angle we used the inverse tangent and then just convert to degree measurement. Finally, depending on the blob that is being detecting for each image snapshot we perform the corresponding methods and sent the data package using a format of two floating points divided by commas.

For the implementation of the light ring we encountered some issues before the TMR, as a quick fix, days before the competition we decided to implement an approach using vision. The first approach consisted on implementing a barrier outside our robot to move depending on which front the lines were detected. See the diagram below. After testing this approach we were encountering issues especially on the goal area, we had to think on an easier approach. For this we decided to calculate the distance of our robot towards the goal, with this we were able to limit the robot movement only for the goal areas.

For Eindhoven we implemented a second camera, now using Pixy camera as our mirror camera, and the OpenMV up on the front. This allows us to know exactly

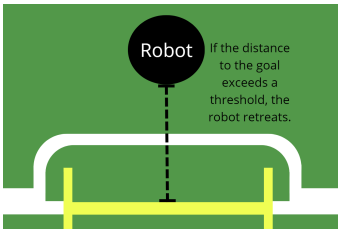

Fig. 46. Goal Distance Logic

when we have the ball, instead of assuming we possess it as explained in the logic section.

VI. Reflection

As we approach RoboCup 2024, our team, RoBorregos, reflects on a year of innovation, collaboration, and project management excellence.

We refined our robot's design and functionality by:

- Improving the mirror design with precise SolidWorks models.
- Enhancing the dribbler system with robust brushless motors.
- Developing advanced line detection and multi-agent coordination.

Collaboration was key, with strategic partnerships and teamwork accelerating our development timeline. We utilized GitHub and Kanban boards to stay organized and focused.

Our technical expertise shone through in our component selection, custom PCB design, and troubleshooting. We're proud of our progress and strategic vision, which positions us for success in the competition.

References

[1] UNIT-Electronics, "DualMCU," GitHub repository, 2024. [Online]. Available: https://github.com/UNIT-Electronics/DualMCU/tree/main.

[2] Nudge Me (channel name) [Online]. Simple Ready Made Solenoid Kicker. YouTube. Published August 29, 2017.https://www.youtube.com/watch?v=GsnxzMAczHU

A. Links

https://drive.google.com/drive/folders/1SdLnZgXB8EmrDErTDHAel__tlK2uMP2B?usp=sharing