OxeBots SSL-EL 2024 Team Description Paper

Ana Livia Santos, Carlos Cerqueira, Davi Azevedo, Eduardo Silva, Erick Suzart, Fábio Miguel Mascarenhas, Felipe Trebino, Greice Kelly, Hanna Andraus, Iarla Queiroz, João Nunes, Larrissa Leanor, Márcio Santos, Matheus Aragão, Matheus Batista, Pedro Braga, Rodrigo Freitas, Ruan Oliveira, Sarah Cerqueira, Samuel Guimarães, Wildson dos Santos, Luciano Rebouças

***Abstract*—The SSL Entry League (SSL-EL) is an entry-level competition where teams compete with three autonomous robots each. In this Team Description Paper, we present the project developed by the Oxebots from IvisionLab at UFBA.**

Our system is structured into three main modules: the strategy module, which is responsible for determining the optimal path and predictions for the robots; the electronics module, which handles the firmware and electronic components; and the mechanical module, which focuses on the physical construction of the robots.

***Index Terms*—RoboCup Brazil, Wireless Control, ROS.**

I. Overview

The SSL Entry League is a robot soccer competition that brings together university students from various institutions, fostering a community where participants can share knowledge and learn from each other about different aspects of robotics. With a goal of gaining more experience in this field, so as to bring the competition closer to our university, the Oxebots team was formed in the Universidade Federal da Bahia (UFBA).

At the match, one's software must be able to receive information from the vision system and referee and, with this data, be able to make strategic decisions that will be sent to the robots in the field, as no user input is allowed in the tournament, all this process must happen totally autonomously. For the first participation in the competition, we decided to divide the Oxebots project into three teams, the strategy team, responsible for the development of the software that will determine the optimal path and prediction for the robots, the electronics team, focused in developing the firmware that serves as the brain of the robots. They ensure seamless communication between the strategy module and the robot's hardware, lastly the mechanical team, who develop the robot's external components. They design and create its parts, such as the wheels and the kicking system, making sure to meet the desired requirements.

The current paper is divided into four parts, there is one part for each of the current team, making a summary of it's development, and a conclusion where we talk about possible future developments of the project.

II. Strategy

The strategy module is tasked with devising the most efficient routes and maneuvers for the robots on the field. This module processes input from the vision system and referee, ensuring that the robots can execute the required actions quickly and efficiently. Performance is crucial, as it dictates the

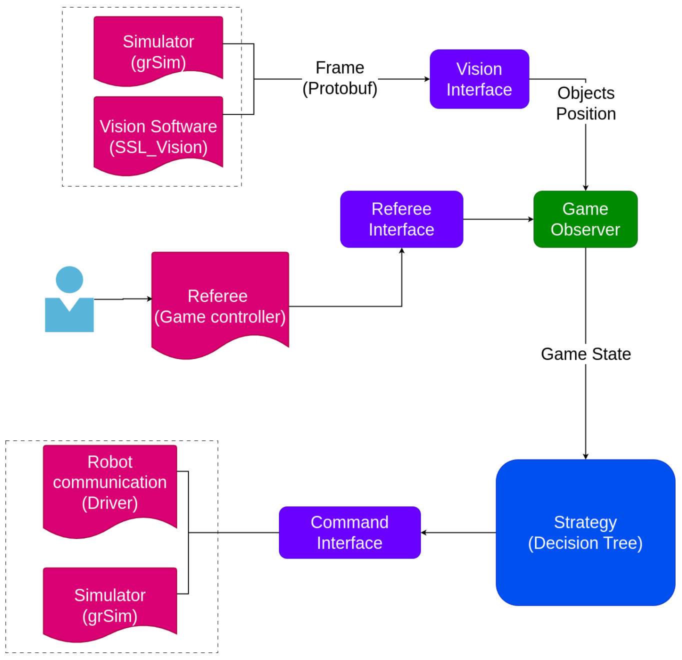

Fig. 1. Architecture of the strategy model.

robot's ability to respond promptly and effectively. To achieve high performance while maintaining a reasonable development speed, we chose to use the Robot Operating System (ROS) [7], version 2, a well-established framework in the robotics field. The architecture of our team is illustrated in Figure 1. As shown in the image, we can divide the software into layers, each one relying on the previous one to behave properly.

The Bridges layer is responsible for the communication with external software, as well as handling all the errors that might occur in the process communicating, the observers are responsible for joining together all the information that we gathered from the vision and referee module, and the core module is the one responsible for creating the strategy per se.

A. Bridges

The interfaces layer (Vision, Referee and Command interfaces) is responsible for applying the necessary transformations into the data being received or exported. Currently, we receive and send all the data via the UDP protocol [13] using Protobuf [14], after the data is processed, it's transformed into ROS messages, actions and services.

1) Vision Interface: The vision interface is the module responsible for obtaining data from the simulator [15] or vision software [12]. We developed this module with the intent of being able to use the simulator or the cameras interchangeably, making the process of development and testing easier. After the data is received from the font, it's treated and sent to the observers where it will be processed.

2) Referee Interface: To get the information on the state of the game, we use the referee interface as a bridge between the Game Controller [1] and our strategy module.

3) Command Interface: After the data is processed, it must be sent to the communication module, where it will be sent to the robots via radio interface. As such, this module makes the inverse process the other two interfaces made, transforming from ROS messages into UDP packages, that will use Protobuf to communicate with the communication module.

B. Observers

The observer layer is responsible for joining together all the information received by the bridge layer and processing it, so the core layer can use it to make the best decision for the moment of the game. Currently, we only use a single observer module, the Game observer.

After collecting the data from the referee interface and simulator or camera software from the vision interface, we process it on the Game Observer module. Here, we also apply the Kalman Filter.

A Kalman Filter [11] is an processes a series of measurements observed over time, accounting for statistical noise and other inaccuracies. By estimating a joint probability distribution over the variables for each timeframe, the Kalman Filter produces more accurate estimates of unknown variables than those based on single measurements alone. In our context, we use the Kalman Filter to predict the positions of opposing players and obtain tracking information. This allows us to make more informed decisions for our team.

C. Core

After all the data has been prepared, it's sent to the core module, where it will be used to analyze the current context of the game and react accordingly. Currently, we use a behaviour tree strategy and keep a fixed robot as the keeper while the other two robots in field can switch between forward and backward.

1) Keeper: The keeper is the player that must defend the goal from enemy attacks while still being able to make the strategy for the offensive. For this, we decided to keep a fixed keeper that will change its behavior depending on the moment of the game.

2) Backward and Forward: For the other two members of the team, the position may change dynamically depending on the current situation of the field. We decided to share the same behavior tree for both of them, and as so, they are able to change their position seamlessly. And so, the forward is the player in the offensive side of the game, while the backward one is responsible for helping defend the goal and make the opposing team have some difficulty on their attack.

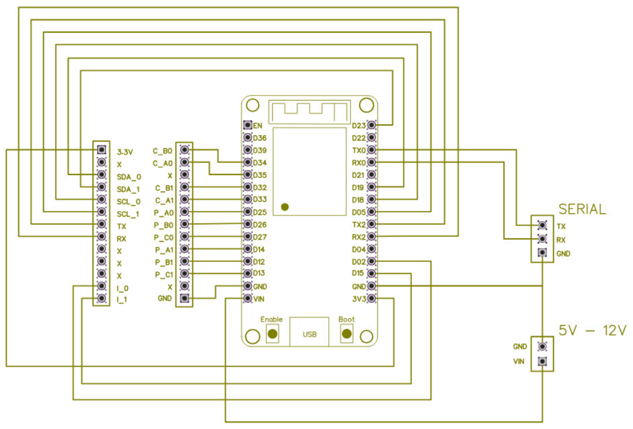

Fig. 2. Prototype of the connection between the drivers and the ESP32.

The robots change their function on the team by observing the current moment of the game. For example, if a robot is closer to the ball, it will assume a forward behavior, as we judge it to be the best behavior for the current moment. But, if the same robot sees that its teammate is closer to the ball and the situation is appropriate, it can assume a more defensive position.

III. Electronics

Currently, we are working on a Arduino Mega [8] based system. The RISC-based board from Arduino was chosen for its practicality and ease of use [9]. As it's the first year of the team, the huge amount of material and the great amount of IO ports made it the right choice for this kind of project. The PlatformIO framework [10] provides us a great advantage on the programming of the embedded code and makes it easy to implement good practices like CI/CD workflow and libraries like freeRTOS, making the process of working with real-time easier.

Earlier in the project development, there was also an attempt to use a the ESP-32 [6]. The Espressif board uses an ARM architecture and has the main advantage over the Arduino Mega being its price and size, as it's significantly cheaper and smaller than the Arduino counterpart. However, the lack of GPIO made it hard to work with the selected motors.

A. Motors

We have selected brushless DC electric motors (BLDC) for our robot components due to their availability, high precision, and efficiency, making them an ideal choice for this project. These motors require a specialized driver that utilizes a generator to operate effectively. For the control system, we are using the MPU-6050, a combined gyroscope and accelerometer, which provides accurate orientation and motion sensing. This sensor is crucial for maintaining stability and precise movement control in our robot.

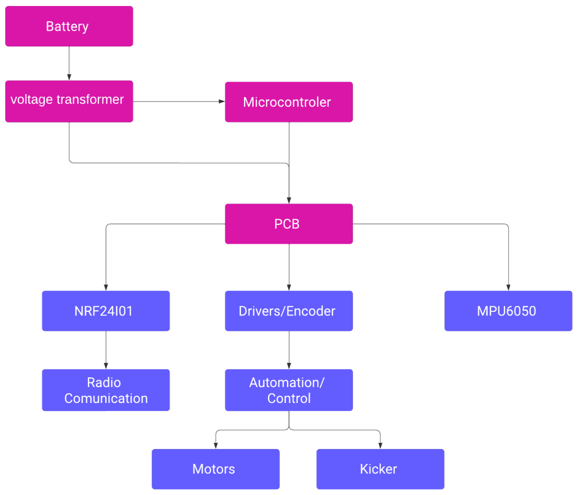

Fig. 3. Diagram of the electronics components.

B. Communication

For the communication, we decided to use radio frequency with the nRF24L01 module. With such, we use a SPI (Serial Peripheral Interface) to communicate with the microcontroller and the nRF24L01. For the setup, we went a single way of communication, i.e, only the computer via a “base station”, another ESP-32 with a nRF24L01 with a medium range antenna, sends information to the robots, while the latter are only able to receive information and not reply.

As such, the strategy module sends the information module via serial communication to the “base station” that will then send the data to the microcontroller. In each robot, then, the NRF24L01 will then decode the message and do the necessary actions, as shown in Figure 4.

C. Firmware

For the development of the firmware, we decided to use PlatformIO. This framework is a cross-platform, cross-architecture, multiple framework, professional tool used for professional development of embedded systems. As it integrates greatly with modern IDEs like VSCode and makes the process of change of components easier, it was a great choice for the current moment of the team.

IV. Mechanics

Mechanical design includes modeling of components and appropriate selection of motors. The modeling is carried out using CAD software, which allows the creation of digital prototypes of the components. This makes it easy to visualize and adjust designs before physical production and prototype development, and enables the use of 3D printing as a way to materialize some parts of the robot in any desired manner. We chose to model all components on Fusion360 for this task, as it was the most fit for our members.

The selection of motors along with the electronic design is crucial to ensure that the robot has the power and precision

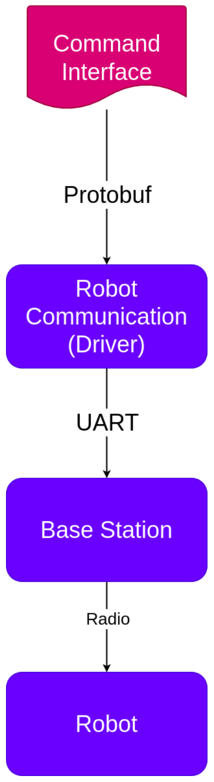

Fig. 4. Radio Communication between the base station and a robot.

required for its functions, for instance, movement. The decision is made based on requirements such as torque, speed and efficiency, taking into account the weight and load distribution of the robot.

For enhanced mobility, we paired those components to omnidirectional wheels, since they allow smooth maneuvering in any direction. To interact with the ball and the field, a solenoid-based kicking system was developed, enabling the robot to kick and accurately throw the ball after determining its timing.

A. 3d printing

3D printing is an important tool for the adequacy and cheapness of the project. PLA filament was used due to several advantages compared to other filaments, such as the ABS filament which requires greater printing complexity and higher costs. PLA is a user-friendly material and offers good print quality, with excellent dimensional accuracy and surface finish. The mechanical strength is adequate for many structural components, ensuring durability and efficient performance in robot applications.

B. Omnidirectional Wheels

Omnidirectional wheels are crucial components for the robot’s mobility. These wheels allow the robot to move in any direction without the need to rotate. Each wheel is made up of several parts arranged at an angle, allowing lateral, diagonal, and rotational movements. This design significantly improves the robot's maneuverability compared to other layouts. The wheels have been CAD modeled and 3D printed using PLA, ensuring that they meet design specifications so we can test out its resistance afterward.

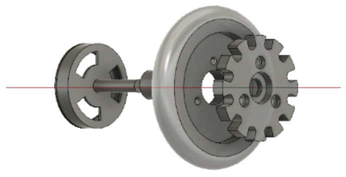

Fig. 5. Design for the wheel.

C. *Solenoid based Kicking system*

As for the robot's ability to shoot the ball, a kicking mechanism with two solenoids was developed so the system could adjust the angle of the shot. The idea is to use a driver system to activate each solenoid individually or simultaneously, allowing the front paddle to be angled and pushed in the desired direction. This provides a more precise method for kicking the ball according to its needs, hence enhancing the performance of the robot on the field. We have modeled the whole system in 3D, in order to have a general idea of its size and capabilities, and we need to perform circuit calculations to ensure we produce the correct voltage for the solenoids.

V. Conclusion

In this TDP, we described the progress made in our first year of development on the Oxebots team to participate in the SSL-EL.

As it's still the first year participating in the competition, there is still a lot of room for improvement in all the areas of the project. For future work, we aim to get better performance in the software development, being able to make the robots send information to the strategy module and use AI methods in the strategy module.

Acknowledgment

We would like to thank Prof. Luciano Oliveira, head of Ivisionlab, for providing us the space, technical advisory and necessary resources for developing this project.

References

[1] Game Controller, https://github.com/RoboCup-SSL/ssl-game-controller, last accessed 2024/07/17
[2] Ros Homepage, https://www.ros.org/, last accessed 2024/07/08
[3] Behavior Tree, https://en.wikipedia.org/wiki/Behavior_tree_(artificial_intelligence,_robotics_and_control), last accessed 2024/07/08
[4] Lacey, Tony. "Chapter 11 Tutorial: The Kalman Filter"
[5] Becker, Alex. "Kalman Filter from the ground up"
[6] Esp-32 datasheet, https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf, last accessed 2024/07/09
[7] FreeRTOS documenatation, https://www.freertos.org/FreeRTOS-quick-start-guide.html, last accessed 2024/07/09
[8] Arduino Mega datasheet, https://docs.arduino.cc/resources/datasheets/A000067-datasheet.pdf, last accessed 2024/07/17
[9] ATmega2560 datasheet, https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ProductDocuments/DataSheets/ATmega640-1280-1281-2560-2561-Datasheet-DS40002211A.pdf, last accessed 2024/07/17
[10] PlatformIO framework, https://platformio.org/, last accessed 2024/07/15
[11] Kalman Filter, https://en.wikipedia.org/wiki/Kalman_filter, last accessed 2024/07/22
[12] `Samp_Dist_Corr`, https://github.com/RoboCup-SSL/ssl-vision, last accessed 2024/07/10
[13] What is the User Datagram Protocol (UDP/IP)?, https://www.cloudflare.com/learning/ddos/glossary/user-datagram-protocol-udp/, last accessed 2024/20/08
[14] Protocol Buffers, https://protobuf.dev/, last accessed 2024/20/08
[15] GrSim, https://github.com/RoboCup-SSL/grSim, last accessed 2024/20/08