ROBOCUPJUNIOR SOCCER 2023

TEAM DESCRIPTION PAPER

LNX Robots

Abstract

Our robot's main chassis consist of two PCBs mounted horizontally, attached together with 3D printed covers made in Fusion 360. On-board Raspberry Pi 4 handles object detection, logging and main playing logic. Teensy 4.1 acts as an additional IO processing unit – controls all our 4 motors and reads from 16 line sensors.

1. Introduction

1. Team

We are a team of 4 high school students from Bratislava, Slovakia

Name	Role
Tomáš Kováč	hardware (PCB)
Mathias Suroviak	software (structure, playing strategy)
Matúš Mišiak	hardware (chassis), software (interfaces)
Matej Repa	hardware (movement)

2. Project Planning

1. Overall Project Plan

It all started as an idea of a robot that would have two PCBs:

- Bottom board
  - Line sensors
  - Motor drivers
  - DC-DC converter
  - Microcontroller (to send information to a Raspberry Pi on top board)
- Top board
  - Raspberry Pi (to process images from camera)
  - Compass

It was considered a good idea to use the whole potential of Raspberry Pi, so multiple processes and threads were going to be utilized.

The software needed to be clean and multiple features needed to be as separated as possible to prevent bugs.

Ideas for kicker and dribbler were abandoned since the development of these two parts would take a lot of time – it is preferred to have 100 % functional robot instead of hardware that is often unreliable. None of our team members had any experience with solenoids or designing mechanical things, such as dribbler.

2. Integration Plan

From the start of our robots development, a lot of different version for hardware were made. We started working on the hardware right after previous year's tournament, in June 2022.

We originally planned making only one robot for Slovakian RoboCup Junior, but things went well with the first one, so we decided to create another one.

Version 1

It was supposed to run on Arduino Mega and use MC33886DH as motor drivers. However, it could have never seen the light of day, since more experienced people from an electronics summer camp recommended us better alternatives: Teensy 4.1 and VNH5019.

Version 2

We decided to use QRE1113 as line sensors since we had success with them on line follower robots.

To power the robot, we chose 3s Li-Po batteries and XL4015 switching regulator.

We did not want to fully commit and buy everything needed for the bottom board out of fear of the design not being good enough. So, the first "working" version of a robot was a bottom board with:

- 2 motors
- 2x VNH5019
- 4x line sensors
- Teensy 4.1

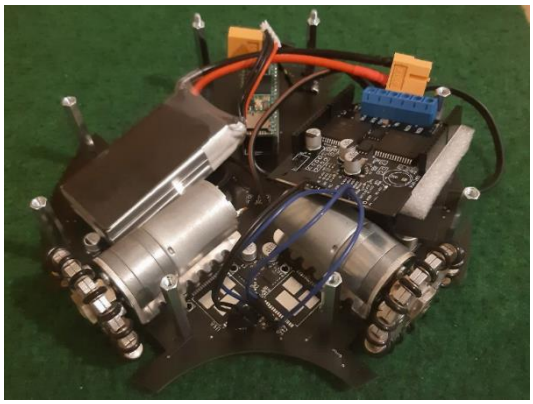

Figure 1: First robot prototype

We heard that wheels are often the most challenging part when it comes to designing a soccer robot, so we decided to buy omni wheels from GTF robots shop. For motors, we decided to use Pololu #3202, with the speed of 1000 RPM.

Results: QRE1113 line sensors have not been functioning well in detecting the line. So, we decided to opt for our own self-made line sensors from IR led and IR phototransistor. We made a few tests with different types of IR LEDs and phototransistors to see which two of them were the most compatible.

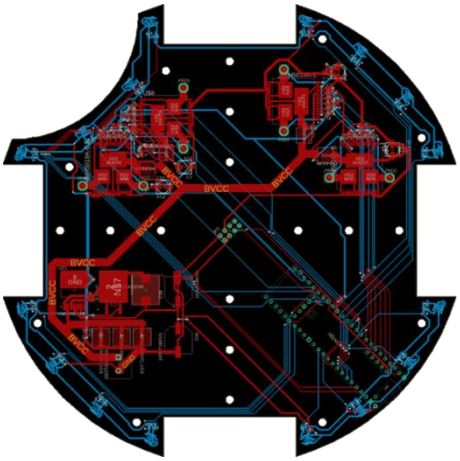

Figure 2: PCB layout of the first prototype

Version 3

Except of minor changes to weight distribution and a change of line sensors, it was the same as the second one.

We decided to use the circular pattern of the line sensors since the wedge starts only 2 cm behind the line according to the new rules. The robot could determine position of the line more precisely this way.

This is the final version of the robot.

Camera selection

Firstly, we decided to place the camera to the front of the robot, but that was later changed to camera being mounted on the handle of the robot, in the favor of larger field of view.

First tested camera – Raspberry Pi Camera Module 2 with IMX219 drop-in camera sensor

- unusual purple haze was visible all around the edges of the image
- customized the IMX219 tuning file (`/usr/share/libcamera/ipa/raspberrypi/imx219.json`)

    - removed values from *Automatic lens shading correction* parameter to revert it to the default state.
- result was a green haze in the center of the picture
    - better – it does not interfere very much with colors intended to detect

Second tested camera – Raspberry Pi Camera Module 3 Wide

- newer, supposed to be better in everything
- autofocus
    - turned out to be a problem
    - vibration caused by motion shook the lens
    - worse than the first one

Third tested camera – Arducam B0310

- same sensor as the second camera
- manual focus
- no problems

Second robot

Around a month before our Slovakian RoboCup, we decided to make the second robot since we had some time to spare. This second robot was an identical copy of the first one, except of the wheels, since we did not want to spend another 200 EUR on GTF robot wheels. So we made our own. They have less friction than the GTF robots ones, but they are useable.

Figure 3: self-made omni wheel

Also, we only had 2 more VNH5019 – we had to use the older model, VNH2SP30. They are mounted on their breakout boards on the top board, because we did not want to solder not fully compatible ICs on the board.

After a bit of programming, we realized that our motors (Pololu #3202) were way too weak to move the robot to any direction – we were only able to move in multiples of 45 degrees, otherwise some of the motors completely stopped moving. After the Slovakian RoboCup Junior we bought Pololu #3203, which are twice as strong and replaced the motors on the second robot with them.

3. Hardware

1. Mechanical Design and Manufacturing

The chassis design was designed using Fusion 360. The main structure of the body are the two PCBs, only the handle, mounting elements and features on the PCBs are 3D printed (white on *Figure 5*).

This allows the robot to have all the components directly on the surface of the body so it saves more space.

The bottom board is connected to the top board via 8 screws. We decided to 3D print a protection for our bottom board so no other robot could damage ours in any way.

Figure 5: Bottom board protection

Figure 4: Render of the robot's chassis

The top PCB has mounting holes for a battery holder, which stops the battery from moving around. The battery is tied with Velcro so it will not fall out.

The handle is also connected to the top board via 4 screws on the side of the robot.

Figure 6: Handle and battery holders

All the components were designed with integrity and robustness in mind. Anytime the component breaks, we make another, stronger one.

2. Electronic Design and Manufacturing

Bottom PCB

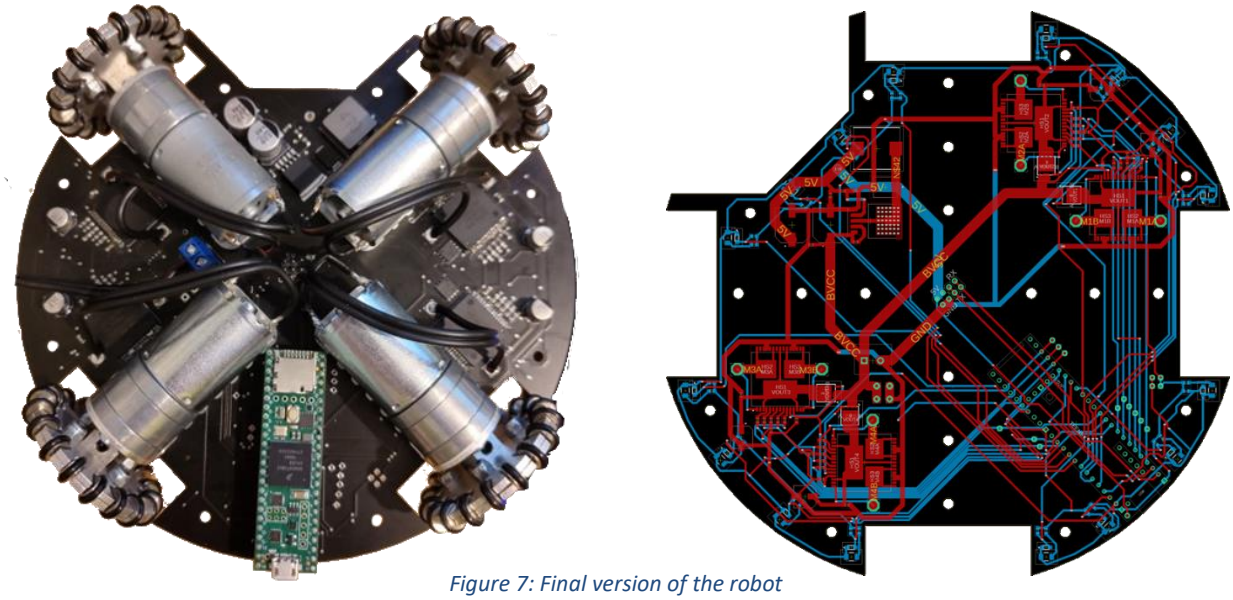

Figure 7: Final version of the robot

- 4x Pololu #3202
- 4x VNH5019
- 16x OSRAM SFH-4656 + Knightbright AP2012P3C
- 1x Teensy 4.1
- 1x XL4015

Top PCB

- 3 buttons (to help manipulating with the robot)
- 0,91“ SSD1306 128x32 I2C OLED display
- Adafruit BNO055 IMU
- Raspberry Pi 4
- Arducam B0310

We decided to use Raspberry Pi since we already have some experience with it and BNO055 since it is one of the best IMUs and we were never let down by it before.

4. Software

1. General software architecture

Low Level (multiproccesing)

Architecture of our software is based on multiprocessing. Program is divided into multiple processes which can utilize all 4 cores of Raspberry Pi and achieve significant performance boost in comparison with single core application.

Each IO component of the robot (for example: camera, compass, ...), which is later referred to as **interface**, runs in the separate process. This process is launched through wrapper which is called **module**. Module is the class where you can define function which will be executed on the separate process and shared variables.

Communication between modules is done through shared memory.

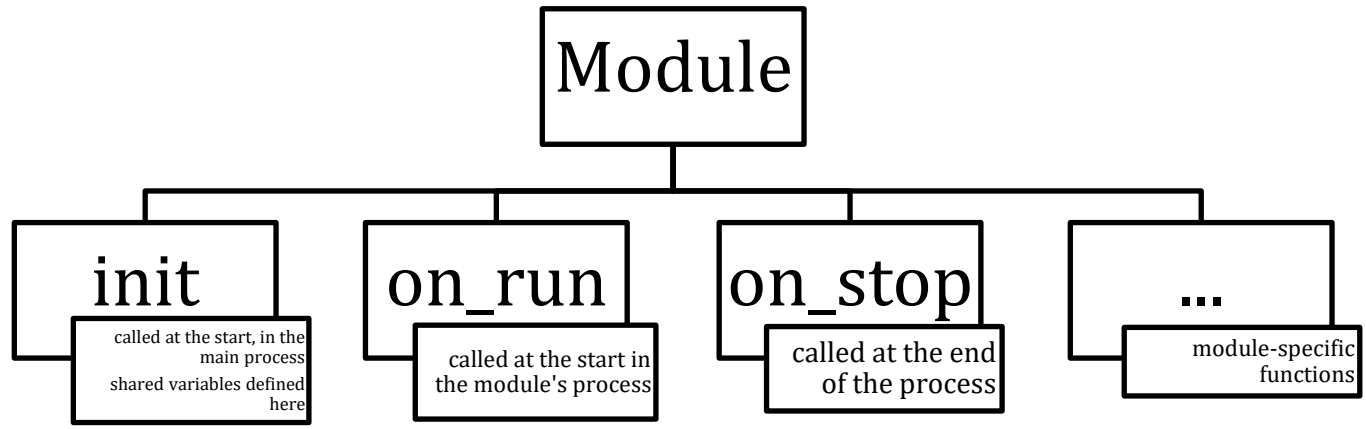

Figure 8: Module methods

Currently there are 7 main modules in our program:

Table 1: Module descriptions

Module	Main task	Output
Bluetooth Module *	Communication with another robot	Data from other robot (see ball, heatmap)
Camera Module	Camera image processing	Position and size of the ball and goal in the frame
Compass Module	Reading data from BNO055 compass module	Heading of the robot
Logger Module	Logging of the data from any module to file	-
UI Module	Communication with UI elements (buttons, display)	States of buttons
Undercarriage Module	Communication with bottom PCB	Values from line sensors
Visualizer Module	Hosting server for visualizer client on another computer	Calibration data

Camera Module takes care of processing of frames from camera. OpenCV is used to detect specific colors in the image. There are multiple steps in process of color detection.

1. Raw frame is read from Camera
2. Raw frame is cropped so only part of the image which contains the playfield is kept

3. Cropped frame is converted from RGB to HSV
4. OpenCV is used to find contours in the HSV frame
5. Bounding box is created around the longest contour

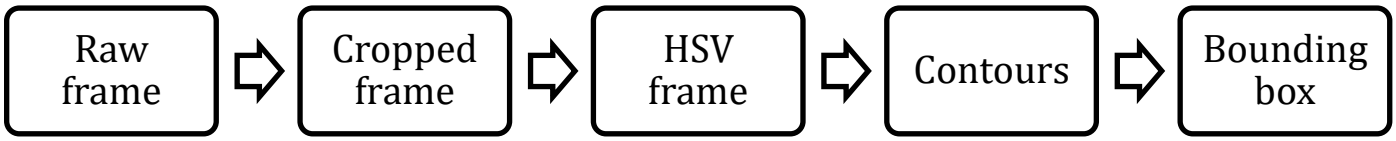

Figure 9: Object detection process

Logger Module is built upon python logging library. Multiprocessing queue is used to log data from any module. Logs are also formatted for better debugging.

```
[12:11:23] [soccer_robot.py/CRITICAL]: A critical error has occured:
[12:11:23] [soccer_robot.py/CRITICAL]: Traceback (most recent call last):
  File "/home/themathy/Dev/robocup-open/RoboCupOpen/soccer_robot/soccer_robot.py", line 67, in run
    self.on_update()
  File "/home/themathy/Dev/robocup-open/RoboCupOpen/robot.py", line 29, in on_update
    if self.line is not None:
AttributeError: 'Robot' object has no attribute 'line'
```

Figure 10: Example log

The biggest advantage of this architecture is that the end user (someone who would program strategy) does not have to deal with anything connected with multiprocessing. If any data is needed, getter on the module can be called to obtain them.

Bluetooth Module is based on Blue Dot library. Robots communicate via Bluetooth to ensure more efficient gameplay. If one of the robots is closer to the ball, the other retreats to our goal to defend

High level code (Strategy)

Main strategy of our robot is based on behaviors, which are executed when some conditions are met. In the diagram above, visual representation of this can be seen.

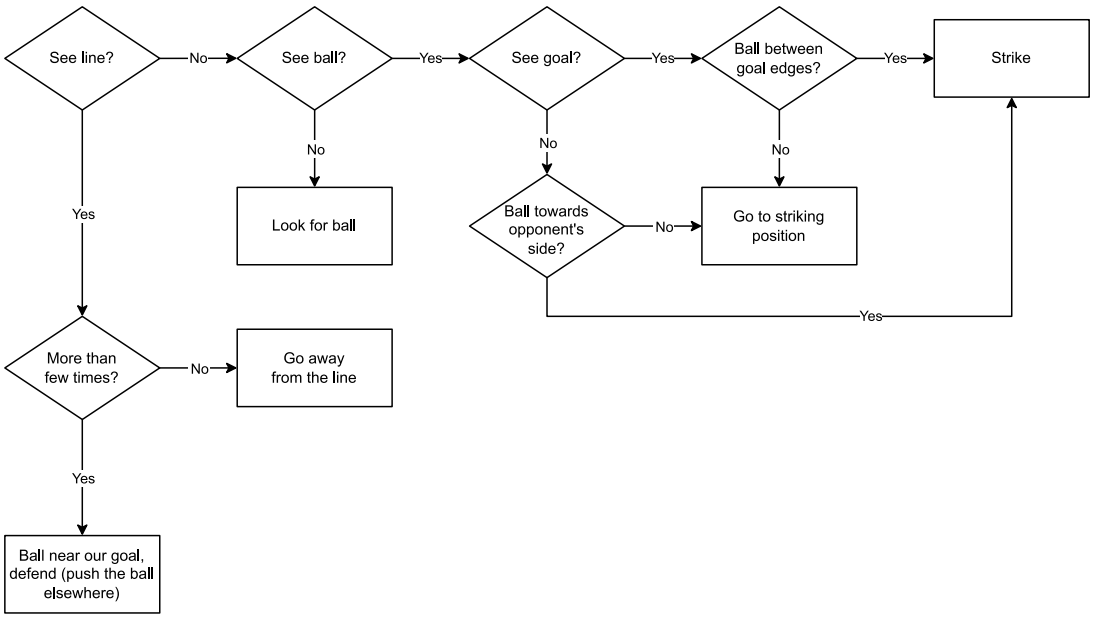

Figure 11: Playing strategy flowchart

Main idea of the robot is to get to the position where the ball angle is between angles of the goal. From this position goal can be scored. To get in front of the ball, angle which has to be set to motors is required. Calculation of this angle can be done using trigonometry.

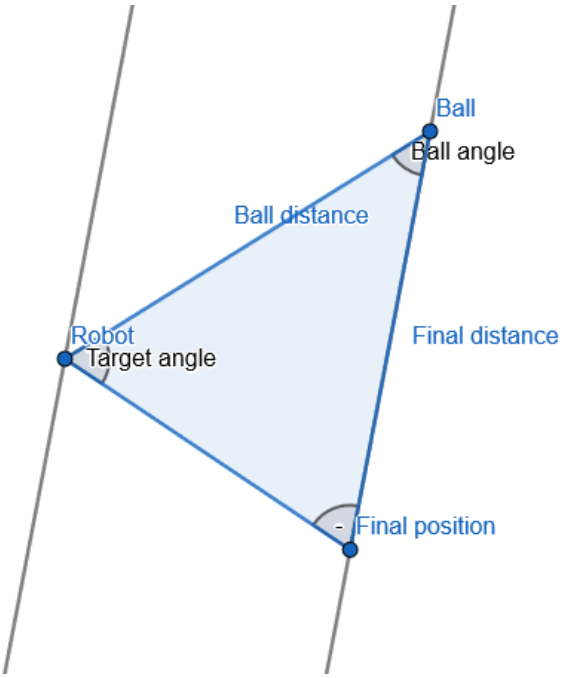

Figure 12: Calculation of the target position

If we define:
bd – ball distance, ba – ball angle, fd – final distance

$$c = \sqrt{bd^2 + fd^2 - 2 \cdot bd \cdot fd \cdot \cos(ba)}$$

$$\text{target angle} = \cos^{-1}\left(\frac{c^2 + bd^2 - fd^2}{2 \cdot c \cdot bd}\right)$$

If the goal is not seen, the robot uses compass, to shoot the ball to opponent's goal direction. This approach is not as precise as goal shooting.

If the ball is not seen, the robot performs a ball finding algorithm, which consists of going back for a fixed amount of time and then rotating.

Line detection has the highest priority in deciding – meaning that if robot sees ball, line avoiding is triggered no matter what other interfaces says.

The robot calculates the vector to the supposed line in multiple steps:

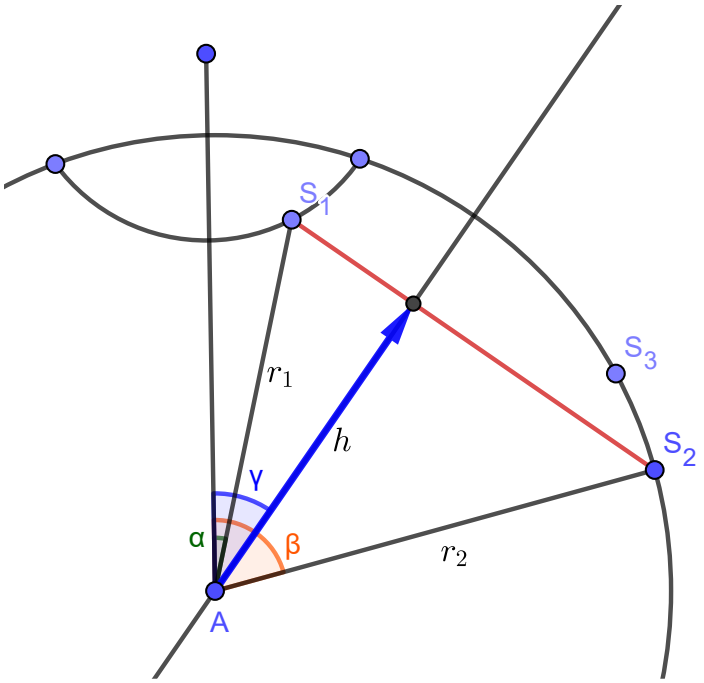

Figure 13: Calculating line position

1. Get the array of sensors that see the line (S_1, S_2, S_3)
2. Find the marginal ones (largest angle between them – S_1, S_2)
3. Calculate the height of the S_1S_2A triangle and angle of the height from the front:

$$h = \frac{r_1 r_2 \cdot \sin(\beta - \alpha)}{\sqrt{{r_1}^2 + {r_2}^2 - 2 r_1 r_2 \cdot \cos(\beta - \alpha)}}$$
$$\gamma = \alpha + \cos^{-1}\left(\frac{h}{r_1}\right)$$

$$r_1, r_2 - Distances\ of\ the\ sensors\ to\ the\ center$$
$$\alpha, \beta - angles\ of\ the\ sensor\ vectors\ from\ the\ north$$

5. Innovative solutions

Multiprocessing

Building our code around multiprocessing was the idea which makes things a lot faster. This approach is suitable for all projects where multicore computer is used. Thanks to our simple API, usage of multiple processes was not that hard as it might looks like. This is the thing we would recommend to other teams if their robot struggles with performance.

6. Performance evaluation

Visualizer

To evaluate performance, we made a visualizer in C++, which is communicating with our program through websockets. Data from robot are displayed there and there is also an option to calibrate camera detection,

line detection and motor speeds. Having this type of debugging tool helped us tweaking our robots to make them play just right.

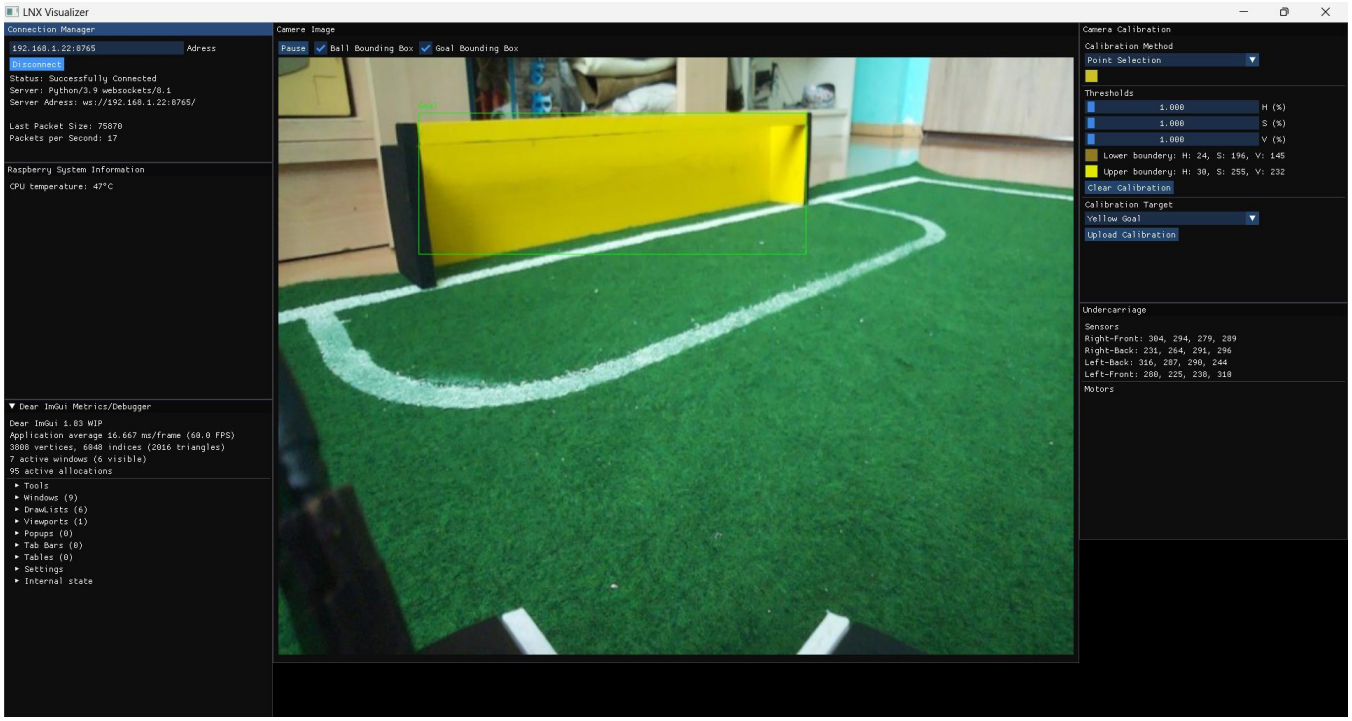

Figure 14: UI of the visualizer

Line calibration

To help solve line detection issues we made a program that saves values from line sensors into an huge array and with the help of excel we can transform this data into graphs, from which we can easily deduct the current issue or can be used to set the line detection value. We can also input custom movements and while these movements take place, the program will save the values from line sensors (we are able to read line sensors about 4 times per millisecond).

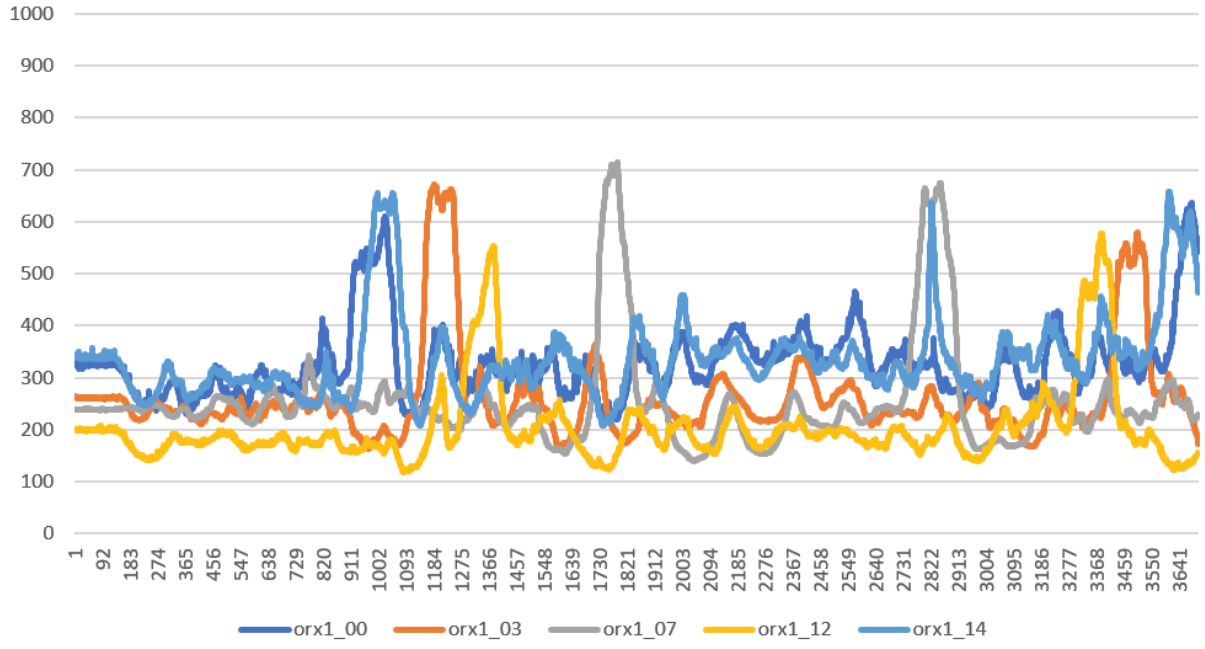

Figure 15: Graph of sensor values in time

7. Conclusion

Overall, we think that we learned a lot during the development of our robots. Although we did not manage to complete all our goals, we created our best robots yet. We designed our own PCBs, with 16 line sensors, 4 motors, wide angle camera and advanced software. This all combined created robots we are very proud of. We hope that we will be able to improve it even more in the future.

Appendix

- PCB schematics are available here
- Videos from development
    - https://youtu.be/nhbiG_MRE0U
    - https://youtu.be/kWrHDDubLJQ
    - https://youtu.be/ZqXnQKk4cL0
    - https://youtu.be/EfiDgmioZJE

References

Libraries used

- **Main program**
    - OpenCV – image processing
    - Multiprocessing – management of processes
    - Websockets – sending and receiving data through websockets
    - PiCamera2 – controlling and receiving frames from camera
    - Pillow – creating image for UI display
    - Bluedot – bluetooth communication
    - CircuitPython – display and compass communication
- **Undercarriage**
    - ArduinoJson – Parsing of JSON in Teensy
- **Visualizer**
    - ImGui – UI panels
    - GLFW – window creating
    - GLAD – OpenGL Loader
    - stb_image – image parser
    - Websocketpp – websocket communication