ZG24Robotics

Extended Team Description Paper

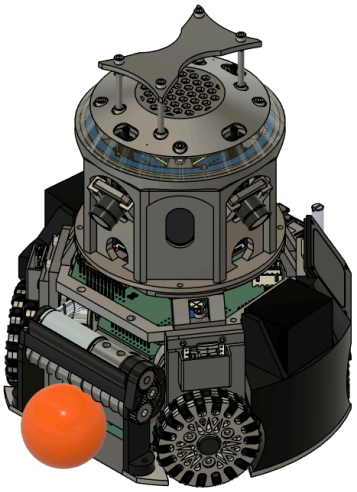
RoboCup Junior Soccer Open 2024./25.

Borut Patčev, Ivan Matošević, Ivan Perko, Jakov Džijan

1. Introduction

ZG24Robotics is a RoboCup Junior Soccer Open team from Zagreb, Croatia. It was established in 2024. and all members of the team came from different RoboCup Junior categories and have a lot of experience with robotics.

In RoboCup Junior Soccer Open autonomous robots play a 2v2 game of soccer using an orange golf ball. They play on a green carpet surrounded by black walls with a blue and a yellow goal on each side of the field. The team that scores the most goals wins. The robots are limited in size, weight, voltage and kicker strength.



Picture 1. *3D model*

2. Hardware

2.1. Robot design

2.1.1. Parts list

Table 1. *Parts list*

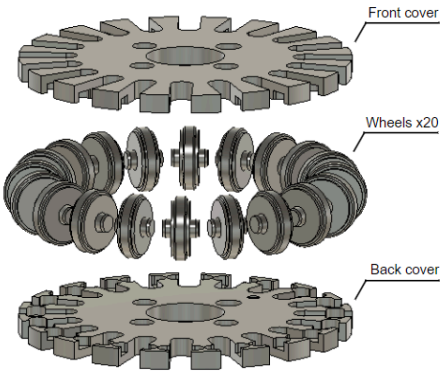
Battery x2	CNHL 3s 1300 mAh
Bluetooth module	HC-05
Camera x4	Sipeed Maix bit
Dribbler motor	Maxon 16 8W
Driving motors x4	Maxon ec45 30W
Multiplexer	CD74HC4067
IMU x3	BNO055
Lidars x8	VL53L1X
Line sensors x32	ALS-PT19-315C/L177
Microcontroller	Teensy 4.1, 4.0
Voltage regulator 5V / 3.3V	Pololu D36V50F5 / D36V50F3
Kicker motor	Solanoid ali

2.1.2. 3D model

The robots were designed in Fusion 360 software. The robot is divided into 4 layers that are connected to each other using M3 aluminium spacers. We removed the spacers between the bottom two layers to save on weight, because the four Joinmax motors are already connecting them together.

2.1.3. Omni-wheels

We are using 3d printed double-layer omni wheels with 20 small rubber wheels. We had an idea to make the wheels out of aluminium to increase durability but opted to 3D print them with PETG filament because the aluminium wheels would be too heavy and expensive. We also tried 3d printing the small wheels ourselves but they were not durable and would break frequently, so we replaced them with ones that we bought.

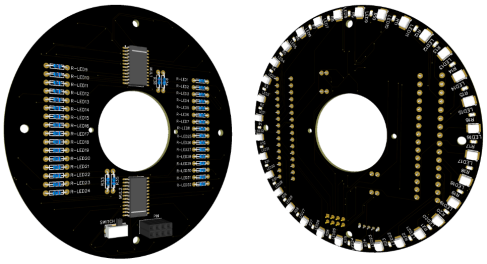


Picture 2. *Exploded view of the wheel*

2.1.4. Printed circuit boards

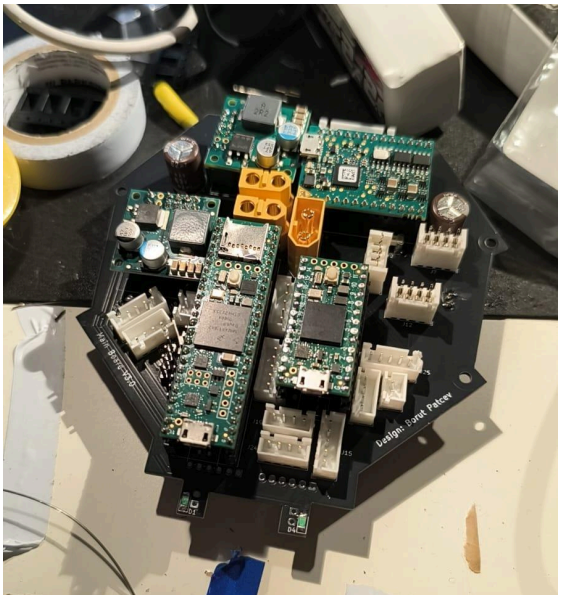
On the robot there are 2 PCB-s that are divided into 2 layers. They were designed in EasyEDA and KiCAD software.

The bottom PCB is positioned 5 milimetres above the ground and is used for detecting the white line. It has 32 line sensors, alongside 32 LED-s and 2 multiplexer switch IC-s which are used to reduce the number of ports used to read the line sensors.



Picture 3. *Bottom PCB*

The mainPCB is positioned above the motors and has all electronics on it. The main controller Teensy 4.1 is on it, together with the IMU, multiplexer, Bluetooth and Camera modules and some buttons and switches. We used to have a Raspberry Pi 4 on the PCB, but since we don't use it anymore there is a lot of space left where it used to be.



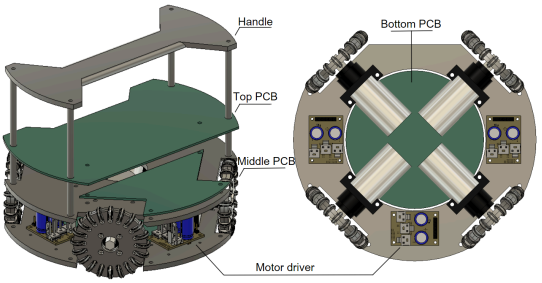
Picture 4. *MainPCB*

2.1.5. Cameras

The robot has four Sipeed Maix Bit cameras. The cameras give the robot full 360 vision around itself, without the use of mirrors or fisheye lenses. Unlike the years prior, this year it is allowed to use more than one camera on the robot, so it is much easier to track the ball around you. The cameras are mounted to a 3D printed holder which is attached above the top PCB.

2.2. Construction process

After we designed and 3D printed all the parts it came assembly time. Since the robot is divided into layers it was simple to put together. On the bottom plate there are four motors positioned in a cross and three motor drivers in between the motors. The fourth driver is placed on the second layer. The PCB with the line sensors is positioned in the hole in the bottom plate. The top plate is screwed on top of the motors which makes it sturdy and ensures the stability of the components on top of it.



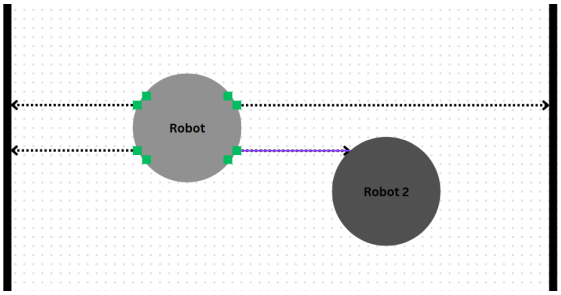
Picture 7. *PCB placement on the robot*

The main PCB is screwed on the top plate in the back of the robot and in front of it is the battery, which can be removed from the robot by taking off the side panel. The dribbler and the ESC are positioned in the front of the robot and are screwed to the top plate.

2.3. Innovations

2.3.1. Lidar positioning

There are a total of 8 lidar range finders on the robot, each on is 45° apart. If one side is blocked by the opposing robot, he ignores that side and aligns using the other lidars. This way the robot can always find his way back to the goal. In addition, if one of the lidars fails to setup or isn't working, the robot can still navigate using the other working lidars.



Picture 8. *Lidar range finder positioning*

2.3.2. Steel plates

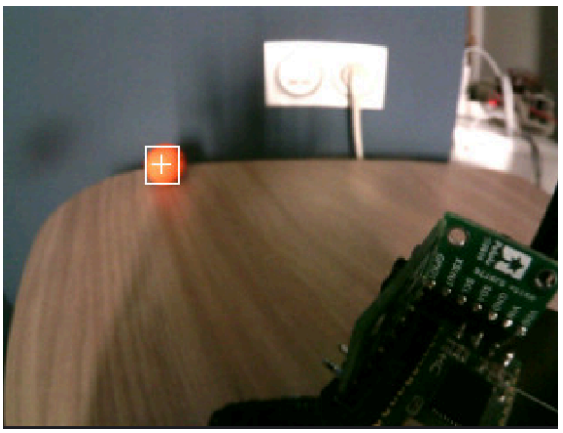
The first and second layer were initially 3D printed out of PETG. We decided that it was better to replace them with steel plates for several reasons. Although the PETG plates were firm, they would bend a little which would lift some motors off the ground and make them lose grip. Then the robot wouldn't go straight and would move in a weird way. The PETG plates were also very lightweight and since the whole robot was very light we decided to make it heavier to have more force on the ground, making it more powerful.

Picture 9. *Steel plates*

3. Software

3.1. Computer vision

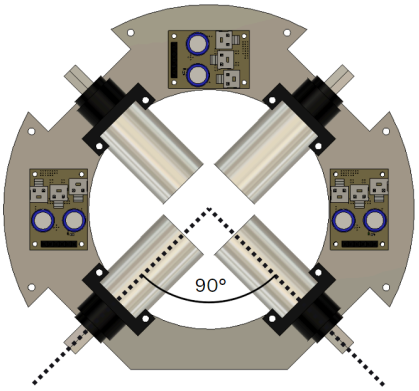
The two OpenMV cameras are programmed in the OpenMV IDE using MicroPython. They communicate with the main controller using UART. It is an unidirectional communication because they only need to send information and do not need to receive anything from the controller. We use a simple blob detection algorithm to find the ball and the goals. The camera just looks for blobs that are in a certain colour range and return the centre X and Y coordinates of the ball and goals.



Picture 10. *Camera view and ball tracking*

3.2. Movement control

Since the robot's driving motors are positioned in a cross and have omni-directional wheels, controlling them is different to controlling robots with standard wheels. We positioned the motors to have a 90° angle between them for easier control and better weight distribution.



Picture 11. *Motor alignment*

Firstly, we have to control each motor individually. There are 4 motor drivers on the robot, one for every motor. The drivers have 3 inputs:

- SLP - used for enabling the driver (digital input)
- DIR - used for changing the direction of rotation (digital input)
- PWM - used for changing the speed of rotation (analog input)

Table 2. *Motor driver states*

State	SLP	DIR	PWM	Output
1.	LOW	LOW	Speed	Motor off
2.	LOW	HIGH	Speed	Motor off
3.	HIGH	LOW	Speed	Clockwise direction
4.	HIGH	HIGH	Speed	Counter-clockwise direction

We made two simple functions for turning on and off all motors at the same time (*Picture 12.*). These functions are only called once to enable or disable the motors for further use in the program.

```
void motorsOn() {
  digitalWrite(24,HIGH);
  digitalWrite(25,HIGH);
  digitalWrite(26,HIGH);
  digitalWrite(27,HIGH);
}

void motorsOff() {
  digitalWrite(24,LOW);
  digitalWrite(25,LOW);
  digitalWrite(26,LOW);
  digitalWrite(27,LOW);
}
```

Picture 12. *Motor enabling functions*

We also made a function for controlling the speed and direction of each motor to simplify the function for general robot movement. It takes two arguments: the motor number (0 - 3) and the motor speed (-100 - 100). If the speed is negative the motor will reverse the direction. We also added a simple way to reverse the direction of each motor using an array of 4 zeroes, representing each of the four motors. While we are setting up the robot, if one of the motors spins in the wrong direction we can change the corresponding zero in the array to a one and the motor direction will reverse. In the same way we can add or remove power from each of the four motors to level the speed of all motors. This way we can ensure that the robot doesn't move unpredictably.

Finally, we need to combine all these functions to control the robot's movement (*Picture 13.*). Our function takes 4 arguments:

- *speed* - general movement speed
- *angle* - angle at which the robot moves
- *rotation* - additional rotation around the Z axis
- *speedLimit* - maximum speed of each motor

We firstly need to calculate the sine and cosine of the *angle* which are multiplied by the *speed* to calculate the speed of each motor. We also add the *rotation* to each motor, after which we check if any of the 4 motor speeds exceed the *speedLimit*. After that we set each motor to its corresponding speed that was calculated before.

```
angle += 135;
angle *= -1;
angle = toRad(angle);
float si = sin(angle);
float co = cos(angle);
float motorSpeed[4];

motorSpeed[0] = -speedMotor * si - rotation;
motorSpeed[1] =  speedMotor * co - rotation;
motorSpeed[2] =  speedMotor * si - rotation;
motorSpeed[3] = -speedMotor * co - rotation;

float maxMotorSpeed = abs(motorSpeed[0]);
for (int i = 0; i < 4; i++) {
  if (abs(motorSpeed[i]) > maxMotorSpeed)
    maxMotorSpeed = abs(motorSpeed[i]);
}
```

Picture 13. *Motor movement function*

3.3. Main program structure

3.3.1. Navigation

When the robot does not see the ball, he navigates back to the goal using lidar sensors. He finds the mean value of the lidars on each side and positions himself in the middle of the goal.

```
if (back)
  go(0,0);
else {
  directionBack = 180 + int(diffLR / backConst);
  if (directionBack > 180) directionBack -= 360;
  go(speedBack, directionBack);
}
```

Picture 15. *Robot centering code*

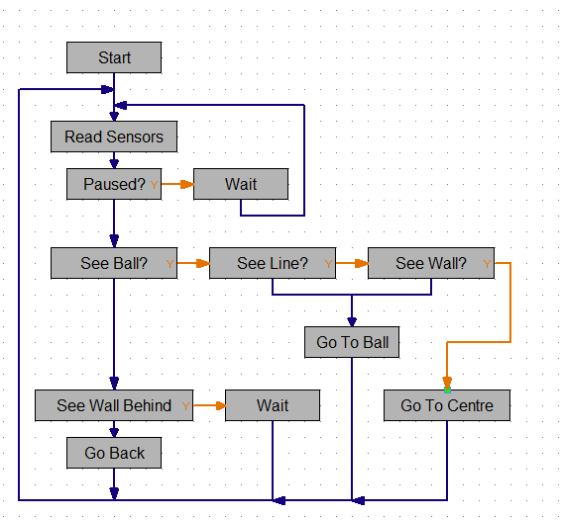
3.3.2. Multiplexing

We used two CD74HC4067 multiplexer IC-s to reduce the number of ports we use for reading the line sensors. Instead of using 32 analog inputs, we use 2 analog inputs and 4 digital outputs. We use the 4 digital outputs to select which sensor we want to use, and the two analog inputs to read them.

```
for (int i = 0; i < 2; i++) {
  for (int j = 0; j < 2; j++) {
    for (int k = 0; k < 2; k++) {
      for (int l = 0; l < 2; l++) {

        digitalWrite(32,i);
        digitalWrite(33,j);
        digitalWrite(30,k);
        digitalWrite(31,l);

        delay(1);

        lin[counter] = analogRead(A16);
        lin[counter + 16] = analogRead(A17);
```

Picture 16. *Line sensor multiplexing function*

3.4. Main program structure

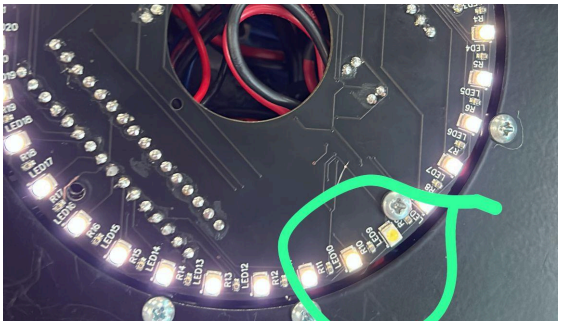


Picture 14. *Block diagram of the main program structure*

4. Problems

We had two major problems:

- Raspberry Pi. We could not figure out how to establish communication between Teensy 4.1 and Raspberry Pi. In the end, we replaced the Raspberry Pi with OpenMV cameras
- Dribbler height. We were limited with the height of the top plate so we had to find parts that were the perfect size: a smaller motor, belt and bearings.
- Line sensors not working. This was an easy problem to solve because we just needed to unsolder the broken sensors and replace them with new ones
-



Picture 17. *Burnt LED and line sensor*

5. Future implementations

We plan on widening the dribbler to better catch the ball. Additionally, we plan on adding a kicker with a capacitor charger module. The plan is to add a kicker and dribbler on both sides of the robot to facilitate picking up the ball. Currently, we don't have enough space on our robot because the motors take up too much space, but we plan on solving this problem by orienting the motors vertically on the lower plate, which would give us room for two dribblers and two kickers. Finally, we plan on installing a 360º lidar to enable better positioning on the field.