Wrocław University
of Science and Technology

Advanced Topics in Robotics Project

Remotely Controlled Mobile Platform

Students: Marcin Kochalski 275514
Tuğçe Avcu 276817

Date: 24.01.2025

1. Introduction

We started this project with the goal of building a remotely controlled mobile platform using Ackermann steering geometry. Our intention was to create a system that could mimic the steering behavior of real-world vehicles, while being controlled remotely through a graphical user interface (GUI). Initially, our plan was to use advanced technologies like ESP32, ROS, and Zephyr for real-time communication and motor control. However, in the process, major integration challenges arose, which turned us toward a simpler yet effective solution that included Arduino, the L298N motor driver, and the HC-05 Bluetooth module. This change allowed us to focus more on the functional aspects of the platform without getting bogged down by complex software compatibility issues.

This project was born from the desire to gain practical experience in robotics and control systems. Further, we needed to go a step further than theoretical knowledge into the application of the acquired knowledge on some hands-on, real-world problem. Interesting was the Ackermann steering geometry, for its wide application in automotive engineering; that was an opportunity to learn about its implementation in a small-scale, controlled environment.

We divided the work among team members based on our strengths during the beginning period. Some of us put more focus on the hardware aspect, like the setup of the motor and steering mechanism, whereas others worked on software aspects, which included developing the GUI and programming the microcontroller. The first try at integrating ESP32 and ROS resulted in compatibility issues related to wireless communication and real-time control. After discussing it thoroughly, we decided collectively to make the project simpler by using Arduino, which would provide us with a more stable and reliable platform.

In our final solution, Bluetooth communication was achieved by using the HC-05 module. The Python-based GUI was used as the main interface to send commands to the platform for movement, such as forward, backward, left, and right. In our design, the L298N was used for power and direction control of DC motors, and a servo motor for the steering angle based on Ackermann geometry. The platform was tested on a series of trials to ensure it was performing reliably and meeting all functional requirements.

Apart from the development of the fully functional mobile platform, the project taught us many lessons: problem-solving, teamwork, and adaptability. Transitioning from a complex to a simpler system proved to be a turning point to reinforce the importance of flexibility in engineering projects. At the end, we achieved a functional, remotely controlled Ackermann steering platform, and we are proud of the results.

2. Assumptions

Originally, the main assumption for the project was using Zephyr for the node control. After few weeks, we decided to switch to ROS only software part. In the end, considering we ended up being only two people in the group, ROS was abandoned as well for simple Arduino-based OS. Another assumption was using Waveshare board with ESP32 MCU, but since we had to go with a simpler version of OS, we decided to use Arduino. Also, considered we had reduced number of people, we decided to use PC based app instead of iOS based app.

For the milestones, most assumptions have been met. The original ones were as follow:

- Milestone no. 1: Hardware tests - First half of November
- Milestone no. 2: Mobile app integrated with ROS and Zephyr - Second half of December
- Milestone no. 2: Testing, refinement, document and presentation development - Second half of January
- Milestone no. 4: Final presentation - week 14 and 15

For hardware tests, we were able to program the Waveshare board so that the wheels are moving early November. Since our plans changed, the second milestone was not completed sucesfully, although the app was complete by then. After receiving Arduino board from our instructor, we were able to integrate it with the system in one evening – the robot was controllable early January.

3. Hardware part

For the main PCB, we used the Arduino Uno board.

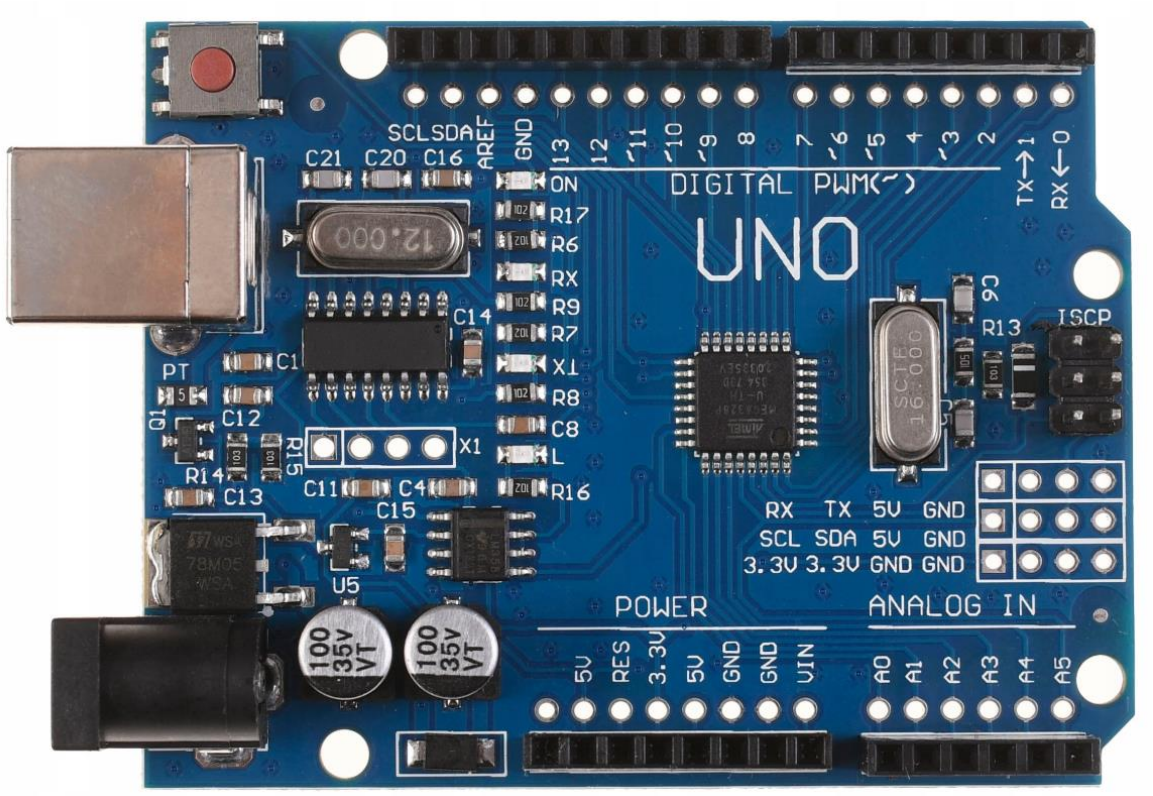

Fig1. Arduino Uno board

With the board, we are able to control the servos using PWM. We were able to power it with 5V by using L298N board.

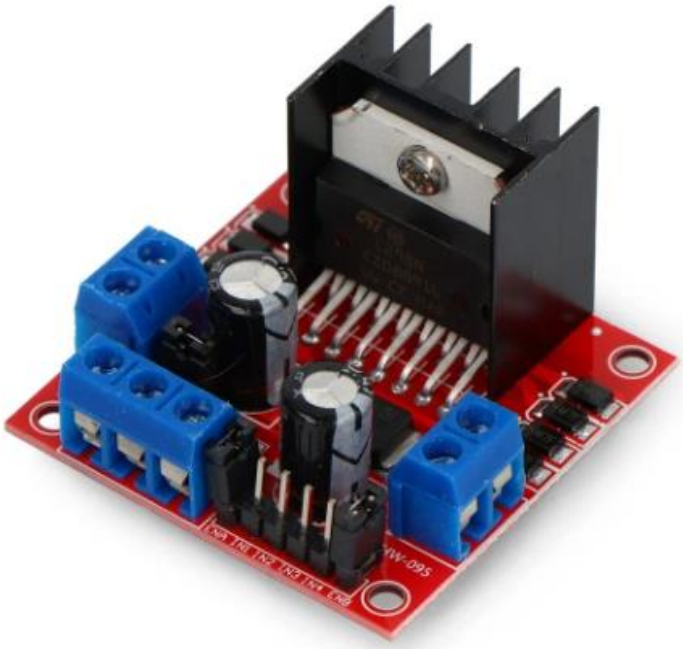

Fig2. L298N board

By connecting 12V batteries, we were able to power the Arduino board and the servos. The bluetooth module was powered from the Arduino board.

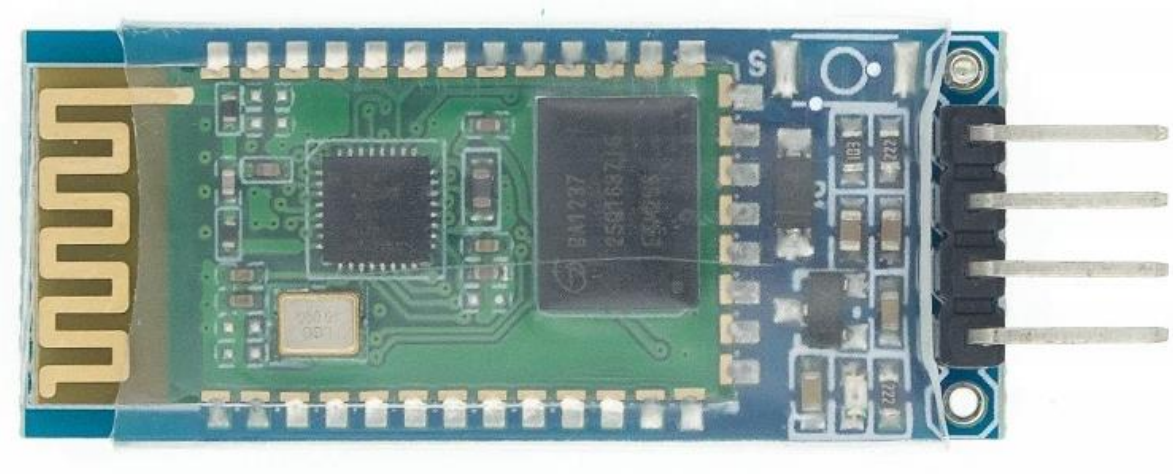

Fig3. HC-05 Bluetooth module

The data pin from the module was connected to RX pin on Arduino.

For PWM control of the servos, we connected the Arduino PWM ports with enable pin on L298N board, allowing us to control the speed and angle from the software level.

All the peripherals were grounded with Arduino's common ground.

For the robot control, it is required to use a computer capable of running the Python program.

The front wheels are steerable using MG996R servo. PWM pin is connected to L298N output and powered through the 5V supply.

Fig4. MG996 servo

Final look of the robot:

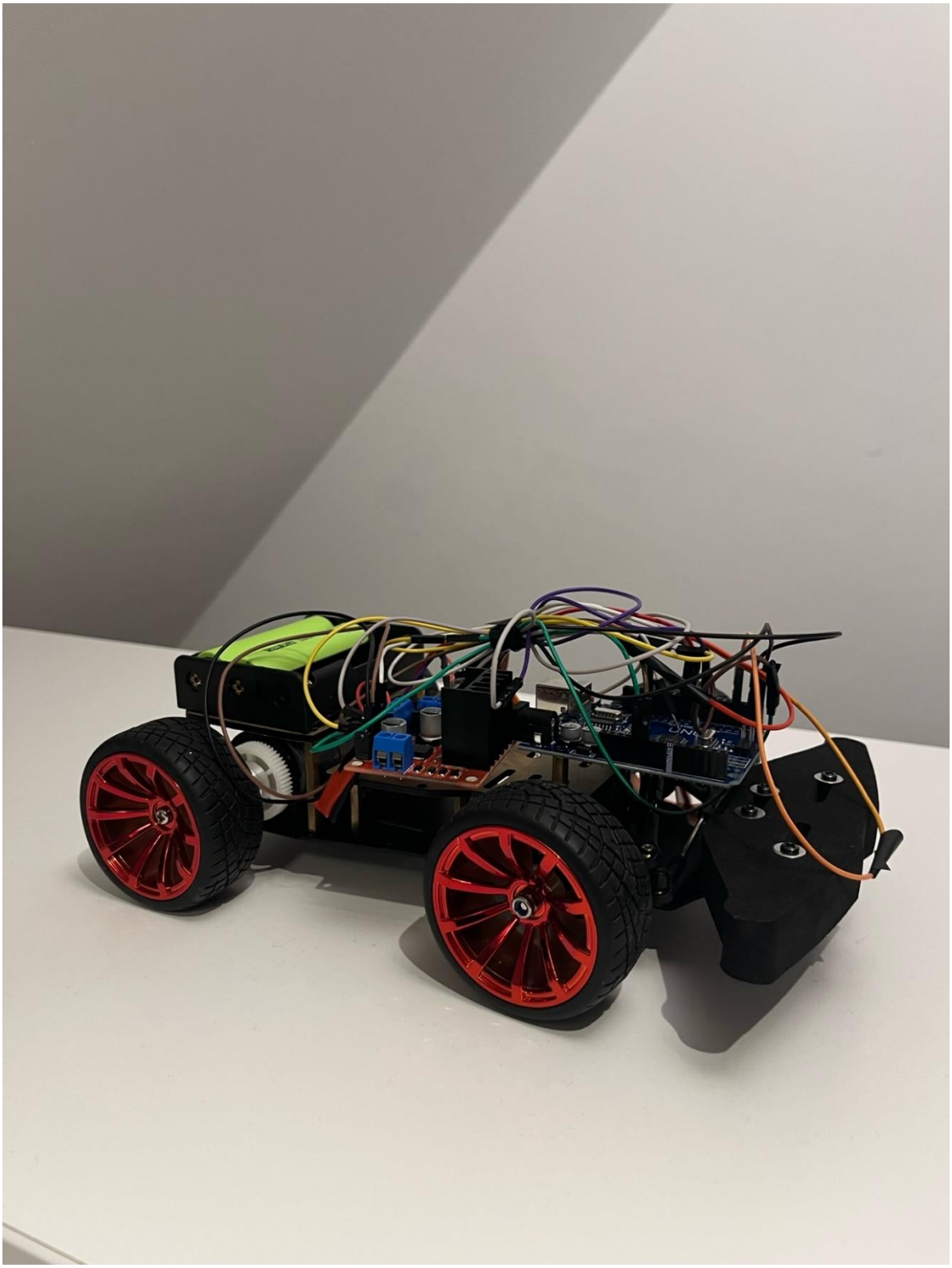

Fig.4 Left view of the robot

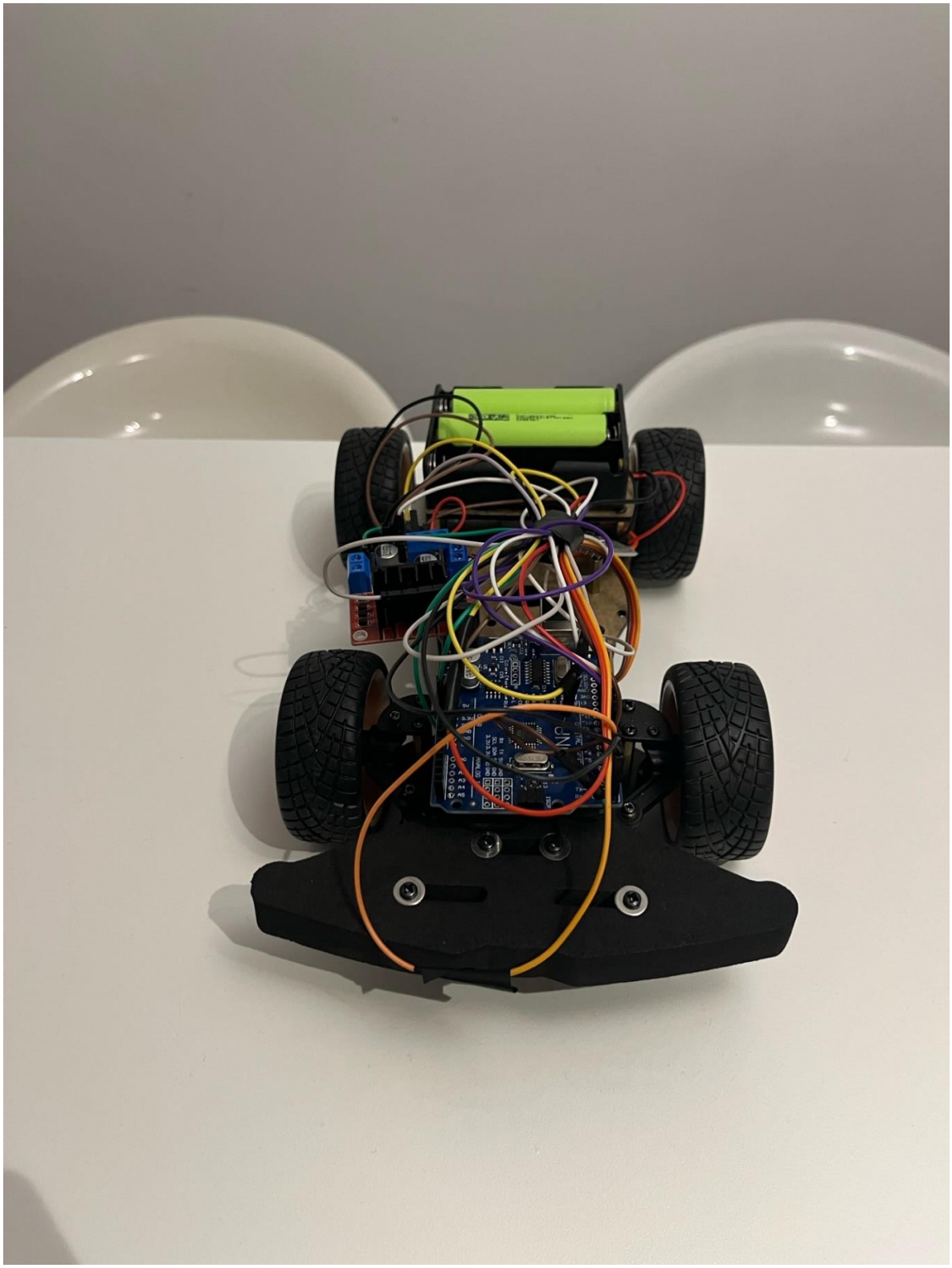

Fig.5 Front view of the robot

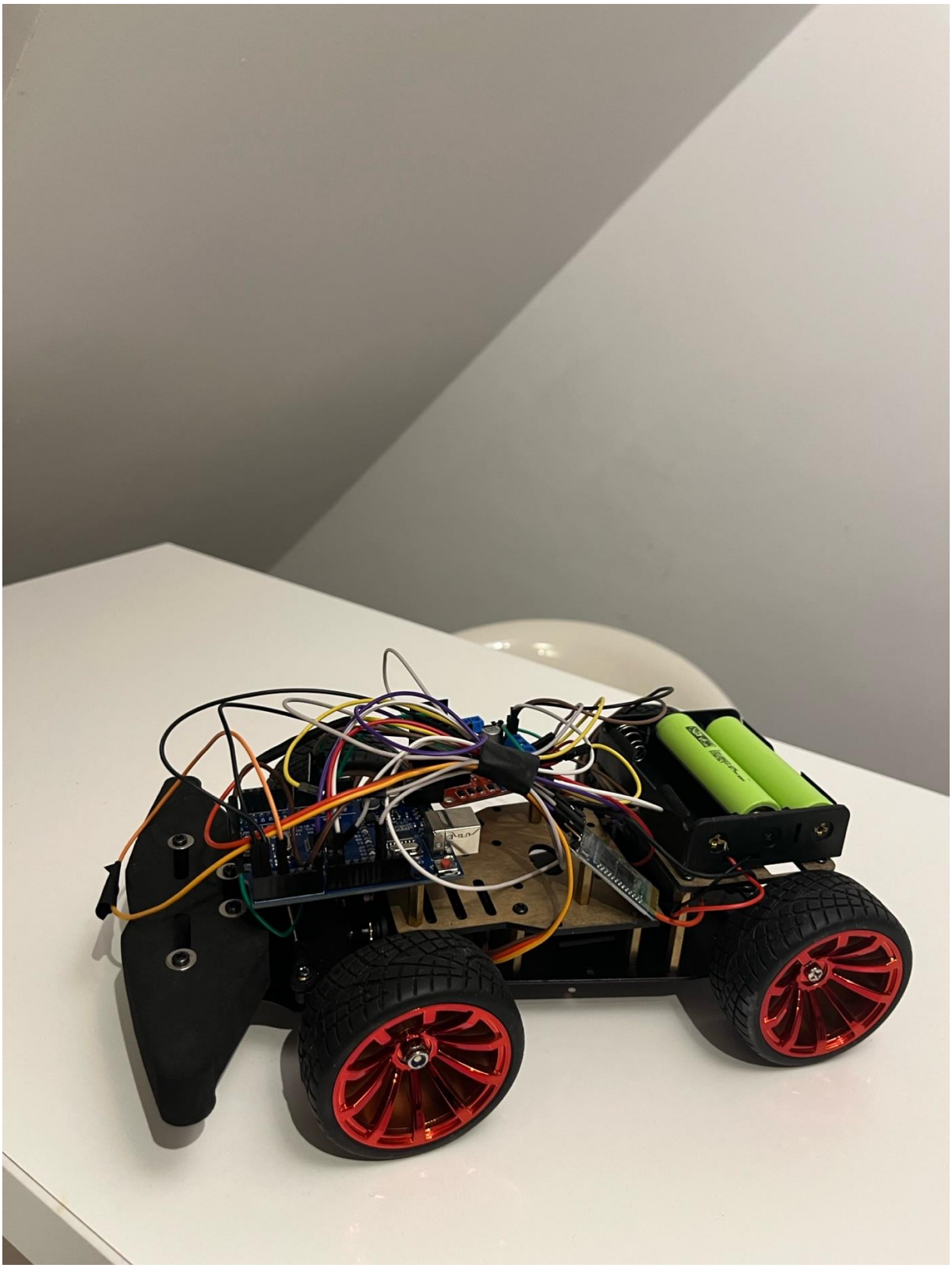

Fig.6 Right view of the robot

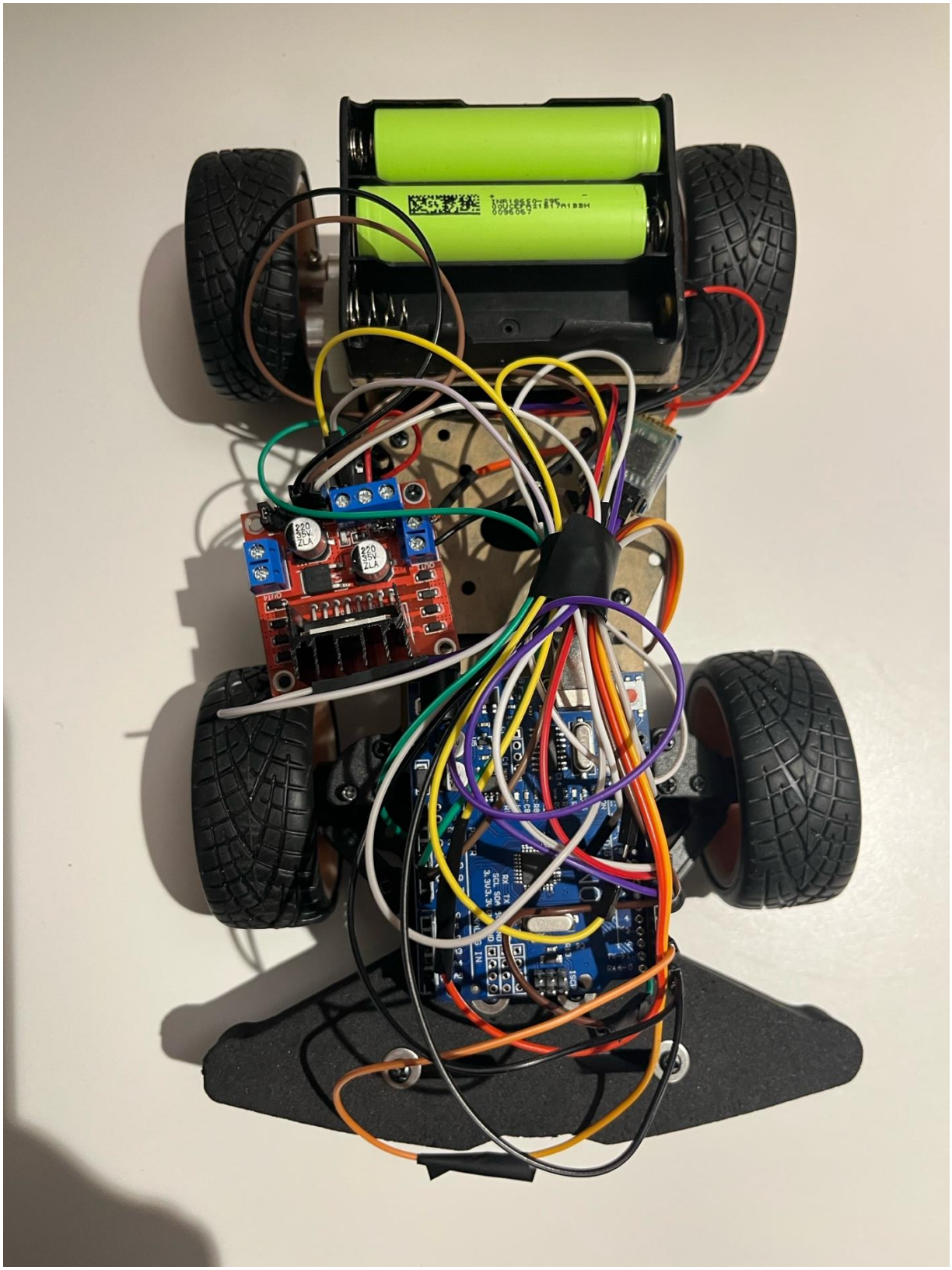

Fig.7 Upper view of the robot

4. Software part

The Arduino code was designed to process commands received via Bluetooth and control the motors accordingly.

First, we had to define the Arduino pins that are connected to peripherals.

```
const int ENA = 9;  // PWM pin for speed control
const int IN1 = 7;  // Direction control 1
const int IN2 = 8;  // Direction control 2
Servo myservo;      // Create servo object to control a servo
int pos = 30;        // Variable to store the servo position

void setup() {
  Serial.begin(9600); // Initialize Bluetooth communication
  pinMode(ENA, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  myservo.attach(3); // Attach the servo on pin 3
  myservo.write(30); // Center the servo initially
}
```

Bluetooth module sends messages via Serial that Arduino processes into signals, depending on what was sent.

```
void loop() {
  if (Serial.available()) {
    String command = Serial.readStringUntil('\n');
    command.trim();

    if (command == "FORWARD") {
      moveForward();
    } else if (command == "BACKWARD") {
      moveBackward();
    } else if (command == "LEFT") {
      turnLeft();
    } else if (command == "RIGHT") {
      turnRight();
    } else if (command == "STOP") {
      stopMotors();
    } else if (command == "CENTER") {
      centerWheel();  // New command to center the wheel
    }
  }
}
```

The commands are processed in following way:

```
void moveForward() {
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, 200);
}

void moveBackward() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  analogWrite(ENA, 230);
}

void turnLeft() {
  for (pos = 0; pos <= 60; pos += 5) {
    myservo.write(pos);
    delay(10);
  }
}

void turnRight() {
  for (pos = 60; pos >= 0; pos -= 5) {
    myservo.write(pos);
    delay(10);
  }
}

void stopMotors() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, 0);
}

void centerWheel() {
  // Move the servo to the center position
  myservo.write(30);  // Center position
  stopMotors();       // Stop the motors as well
}
```

Although normally the servo angles should be from 0 to 90, in our case the servo was either not calibrated properly or damaged. The center angle should be 45 degrees, but for us 45 degrees was slight to the right. The wheels could turn all the way to the right if we use 90 degrees, but considered we can't use negative values of the angle, it was able to turn left only 30 degrees, so we centered it at 30 degrees and allowed both wheels to turn by 30 degrees.

The Python GUI was created by using Tkinter library. It uses the serial library to establish a connection with the HC-05 Bluetooth module.

```
def connect_bluetooth():
    global bt_connection
    try:
        bt_connection = serial.Serial(BLUETOOTH_PORT, BAUD_RATE)
        time.sleep(2)
        messagebox.showinfo( title: "Connected",  message: f"HC-05'e {BLUETOOTH_PORT} connected!")
    except Exception as e:
        messagebox.showerror( title: "Error of Connection",  message: f"Bluetooth is not connected: {e}")
```

The user interacts with buttons in the GUI to send commands.

```
def send_command(command):
    global bt_connection
    if bt_connection and bt_connection.is_open:
        try:
            bt_connection.write((command + "\n").encode())
            print(f"Command sent: {command}")
        except Exception as e:
            messagebox.showerror( title: "error",  message: f"Cannot sent the message: {e}")
    else:
        messagebox.showerror( title: "no connection",  message: "Connect the bluetooth first")
```

Each button triggers a function that sends the corresponding command to the Arduino.

```
btn_forward = tk.Button(root, text="↑", font=button_font, bg="yellow", fg="black", width=5, height=2,
                        command=lambda: send_command("FORWARD"))
btn_forward.place(x=270, y=300)
```

The Arduino processes these commands to control the motors and servo, executing the desired movement.

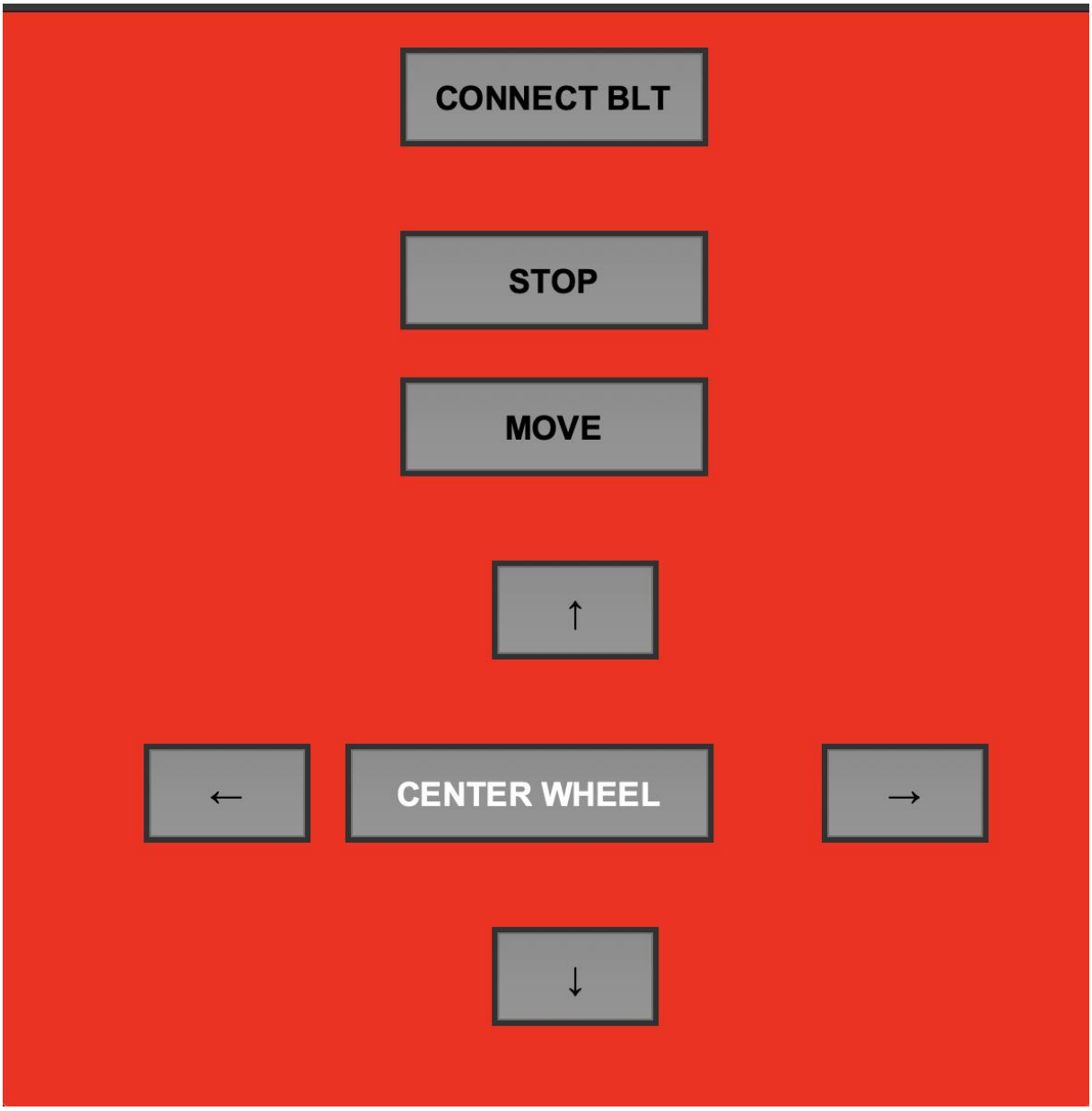

Fig.8 Python GUI

5. Startup and calibration

To program the board, it is required to open circuit the batteries, so no power is delivered to the board, as 12V supplied by the batteries and 5V supplied by the USB can damage the board. It is also required to unplug the power from HC-05, as it creates additional COM ports that interfere with programming of the board, blocking the main COM port.

Another important point is having Bluetooth devies discovery selected to Advanced, as HC-05 might not appear without it enabled. If correctly paired, it should appear in the devices as HC-05. The device pin is 1234.

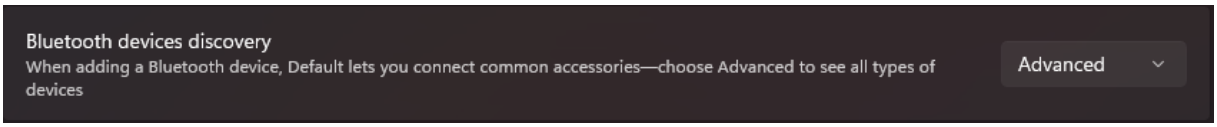

Fig.9 Windows options

We run the Python program, connect the bluetooth and the robot is ready to go.

6. Tests and results

Functionality Tests: Verified movements.

Bluetooth Range: Stable communication within a 10-meter range.

Stability Tests: Continuous operation for over an hour without failure.

GUI Usability: Commands executed seamlessly with no noticeable delay.

7. Tasks and distribution

Hardware design: Marcin Kochalski
Python app design: Tuğçe Avcu
Arduino design: Marcin Kochalski
Testing and debugging: Collaborative effort
Report: Collaborative effort

8. Summary

Project evaluation

The project successfully demonstrated the development of a remotely controlled mobile platform using Ackermann steering geometry. Despite initial challenges, the final system performed reliably and met the functional requirements.

Possible Enhancements

Better functionalities about moving the robot left-right from keyboard.
Extended Bluetooth range for broader usability (maybe with more advanced bluetooth module/board)
Adding sensors for obstacle detection and navigatoin.
Enhanced GUI features for better control.

9. Appendix

https://github.com/kcir-roberto/projects-ro-redcar/tree/main - Main directory of the GitHub repo
https://github.com/kcir-roberto/projects-ro-redcar/blob/main/project_plan/project_tasks_n_schedules.md - Initial task and milestones schedule
https://github.com/kcir-roberto/projects-ro-redcar/blob/main/project_plan/project_charter.md - Project charter
https://github.com/kcir-roberto/projects-ro-redcar/blob/main/report/appcode.py - Python app code
https://github.com/kcir-roberto/projects-ro-redcar/blob/main/report/arduinocode.ino - Arduino code