IITM Robot Car Report

Name - Arnav Jaiswal
School - Titan's School Amravati
Project - Pi Pico Bot with Mecanum Wheels
Course - Introduction to Electronics

Parts List -

Robot Car -

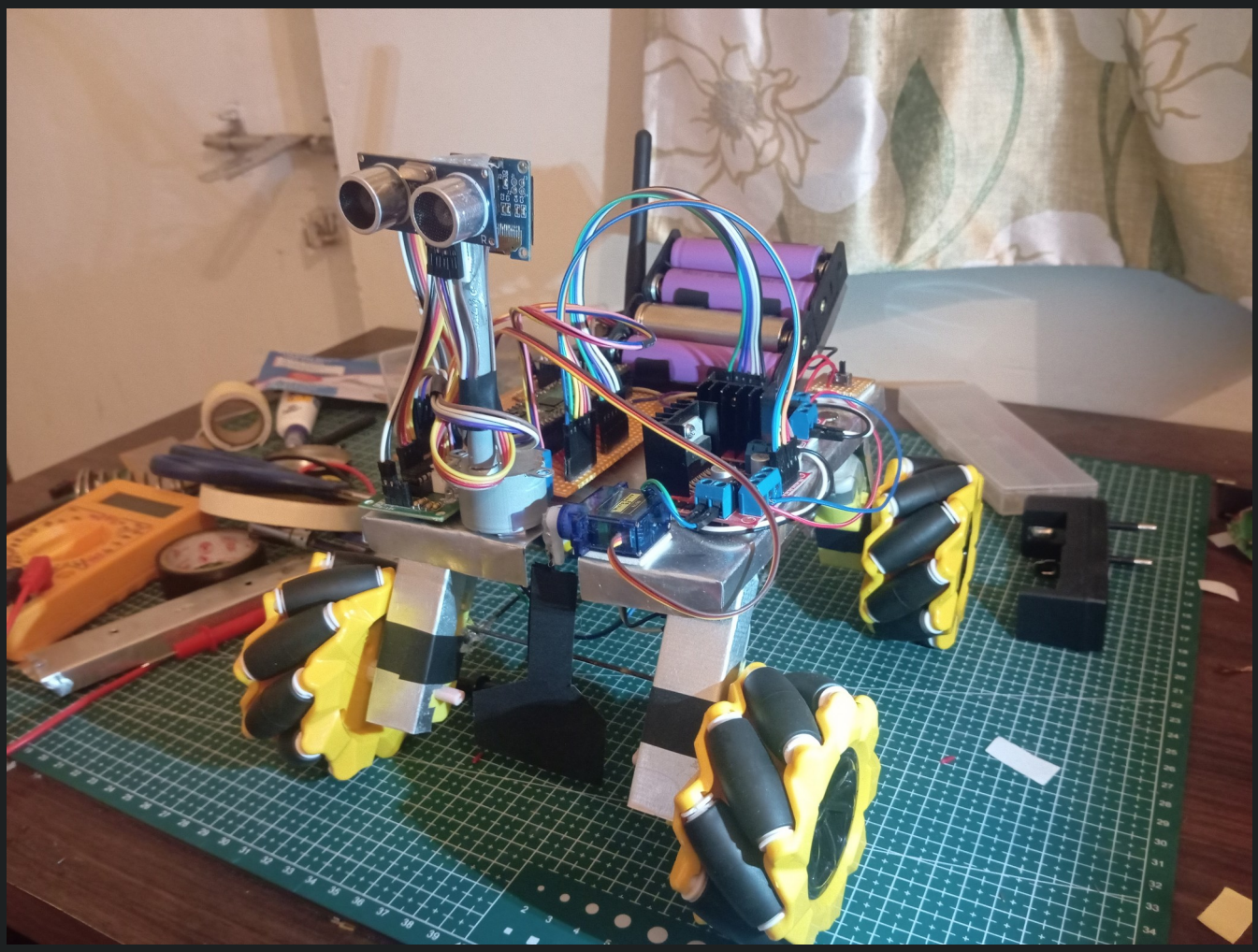
- Pi pico
- L298N Motor Drivers
- BO Motors
- Mecanum Wheels
- NRF PA LNA Module
- 28-BYJ Stepper Motor (ULN2003 Driver)
- SG90 Servo
- HC-SR04 Ultrasonic Sensor
- 128x64 I2C OLED Display

Transmitter -

- ESP32 Devkit V1
- 128x64 TFT Screen (SPI)
- NRF Module
- PS2 Joystick Module
- Touch Sensors

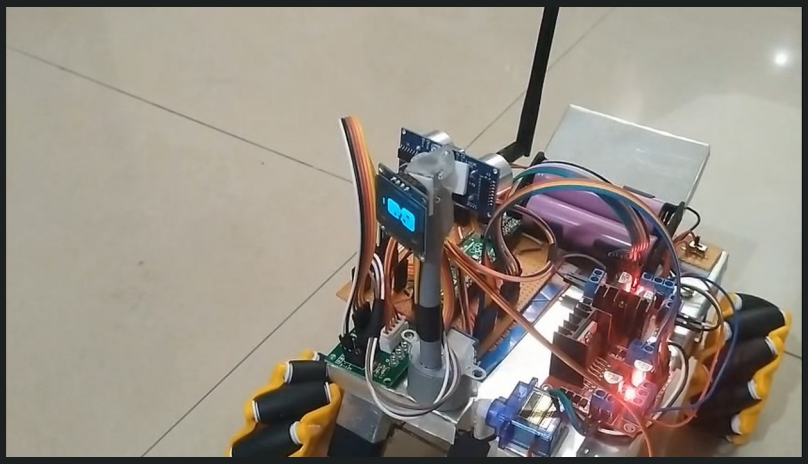
Project Descirption -

The Robot Car is controlled by a Rasberry Pi Pico connected to the L298N Motor Drivers, a NRF Module, and a head assembly consisting of a stepper motor, an OLED screen module and a ultrasonic distance sensor.



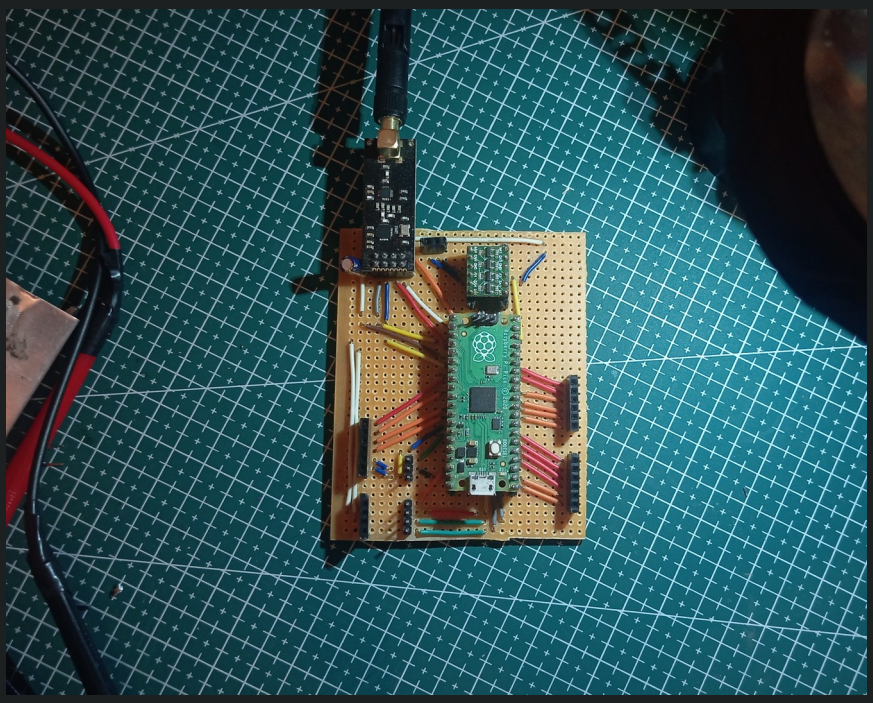
The chasis is made out of cut pieces of aluminium sheet, bent to make the chasis, with hard wire added for support along the bottom. It has a paddle at the front controlled by a servo to kick things in front of it. The head assembly has an ultrasonic distance sensor used so that the robot doesn’t accidentally

slam into a wall or a nearby obstacle, it can sweep around scanning around it too. We can also switch it to have the OLED screen on the front to show animated eyes to give the bot a bit more personality.

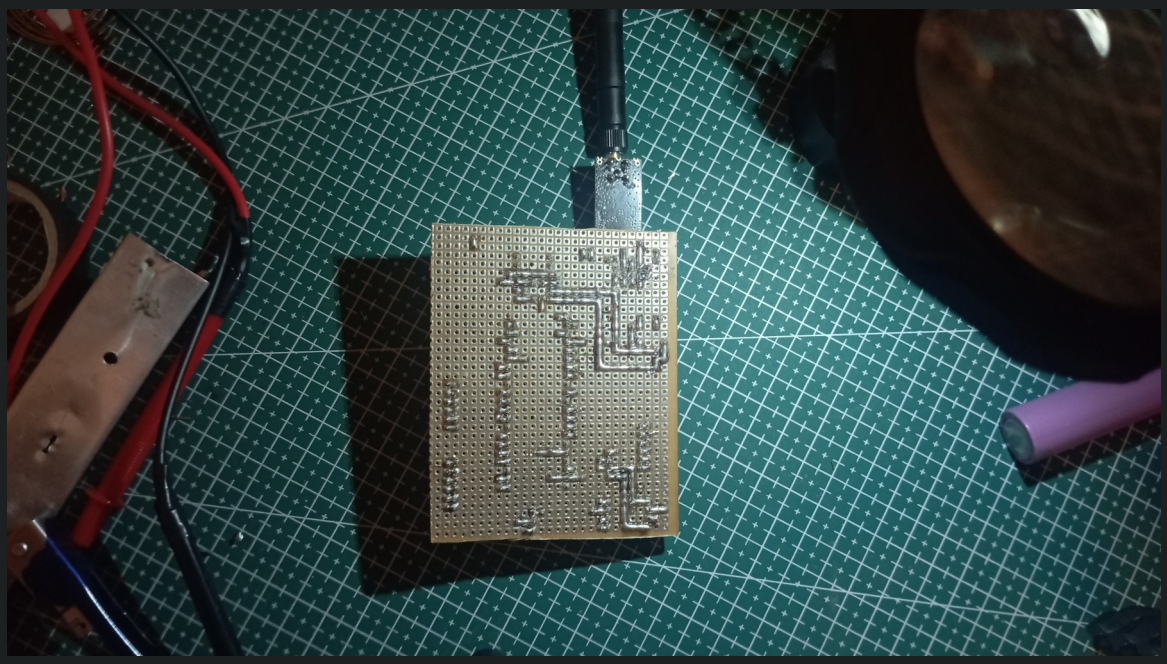


Of course I can’t proceed before adressing the elephant in the room - the huge mess of wires on the top of the robot. Unfortunately due to the amoun of stuff I added, the wires just kept growing until I gave up and embraced them. The wires are now at full display showing my efforts to make this bot more featureful.

I also want to talk about the PCB I made for the bot using a perfboard, and pre-cut wires I bought online. I soldered pin headers and the components, and although it may not be good to look at, it works, and I have convinced my self that that’s all that matters.

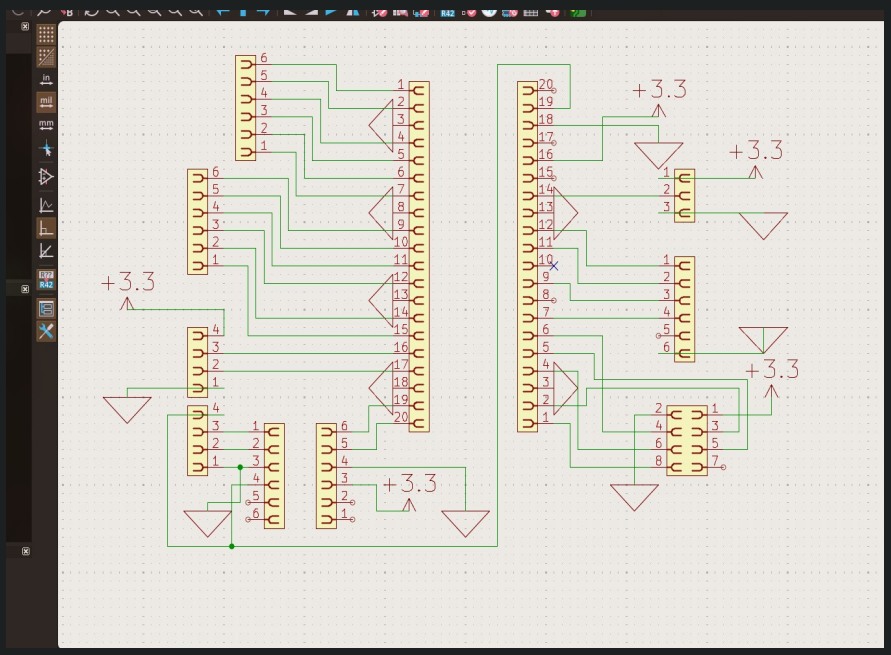


From the back, I had to go under the board and connect up some components because I didn’t want to overlap wires on the top to stop it from looking worse.

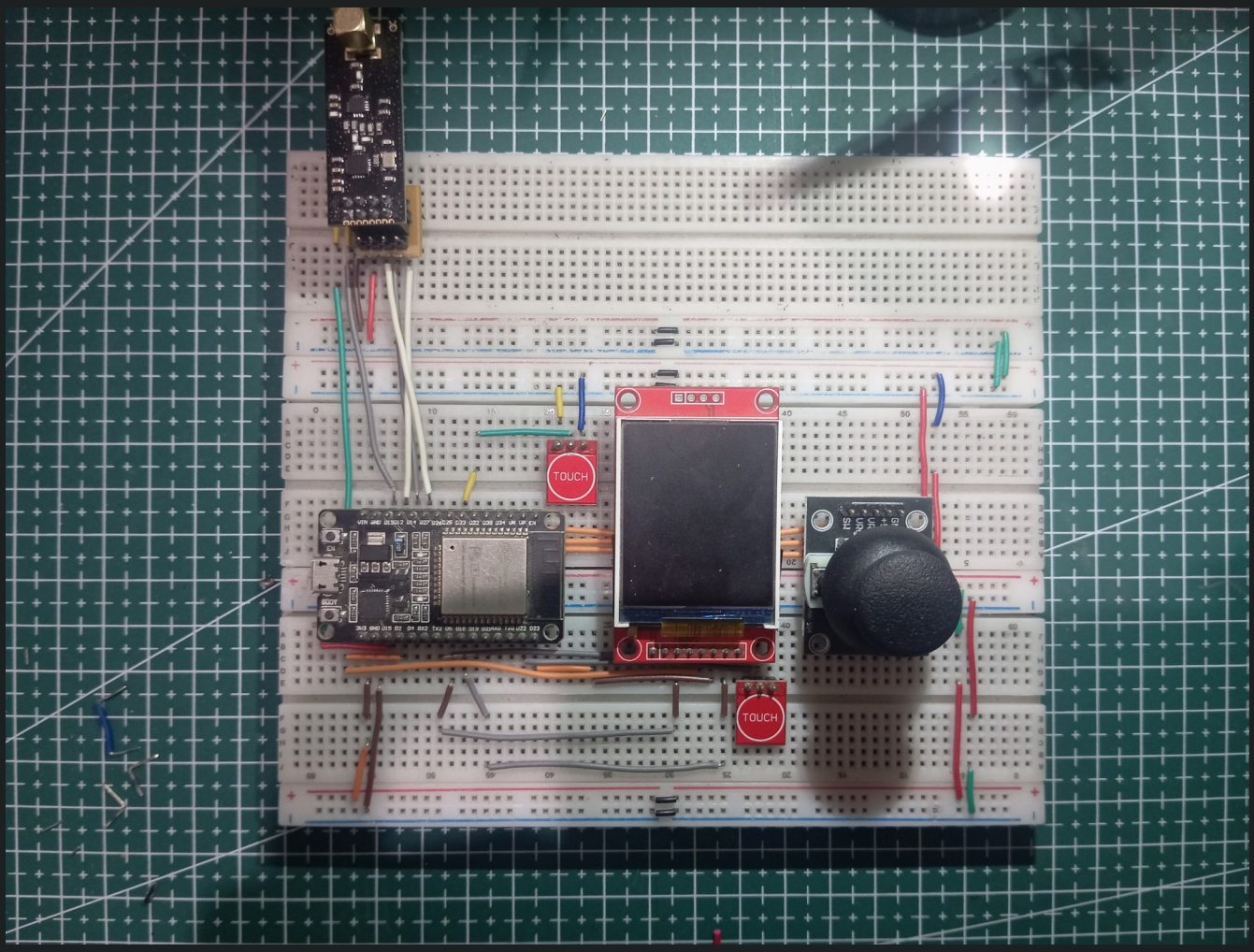


Before making this, I had to design and test this circuitboard to make sure I have enough pins and they are connected correctly. For that I used KiCAD’s

schematic editor tool to whip up a quick rought schematic for the board.



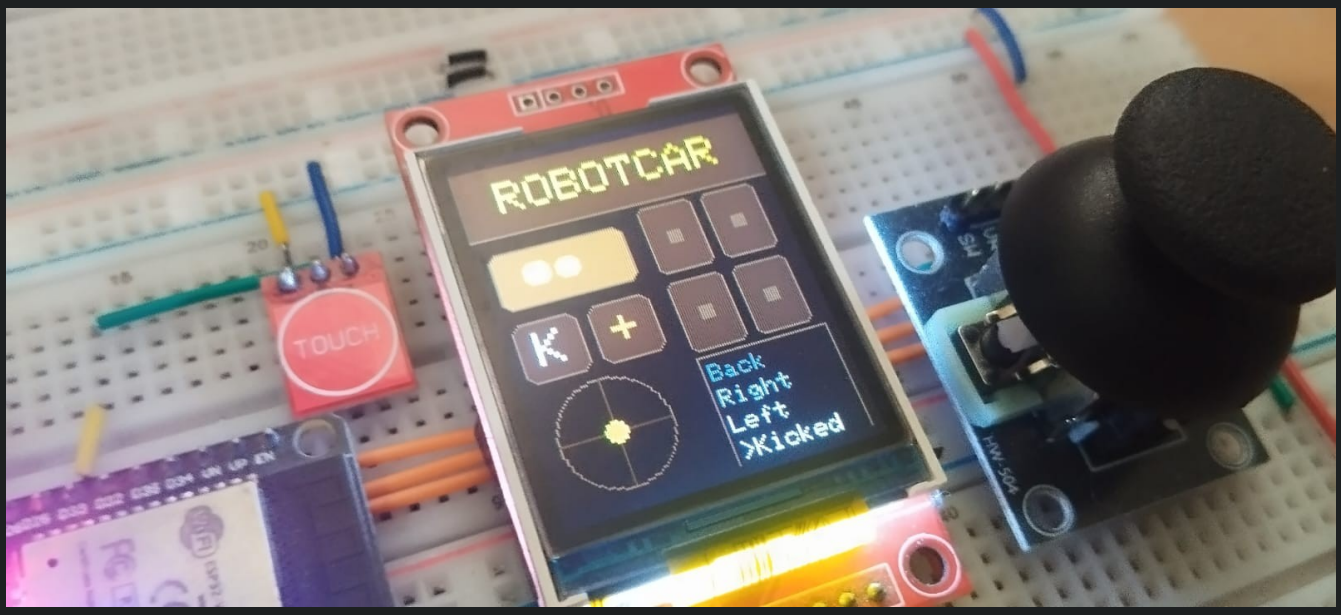
Now coming to the transmitter. I needed to connect a 128x160 TFT Screen that I had and the NRF Module both to an MCU. And honestly even after a lot of time learning electronics, the one thing I havn’t gotten to work is shared SPI lines. I have never been able to use 2 SPI devices in the same bus. And the only other MCU that I had other than the Pi Pico that had more than one SPI bus was the ESP32, so I used the HSPI bus of the ESP to connect to the TFT screen, and the VSPI bus of the ESP to connect to the NRF module, I then connected the Joystick module to a couple of ADC pins on the ESP, and all I had to do was write some code.



Code Breakdown -

I feel like I should start with the transmitter in terms of the code. I used the Arduino IDE for the ESP32, and I used the well known TFT_eSPI library for the screen anf the RF24 library for the antenna. After everything, I ended up with 5 seperate files for the code, 4 header files (to make the main code look clean) and one main.ino file. The 4 headers that I needed were: Colours.h, DrawUI.h, Log.h and Wheels.h

The colour header file stores the gruvbox dark colour scheme to be used for the TFT screen. You will notice that I am a stickler for making things look aesthetically pleasing, which is why I invested a lot of time into making sure my control dashboard on the screen looks good. I was inspired for this style by my desktop setup on which I use Arch Linux with Hyprland, and the gruvbox dark colour scheme. You may see some similarities between hyprland and the UI I made for the screen.



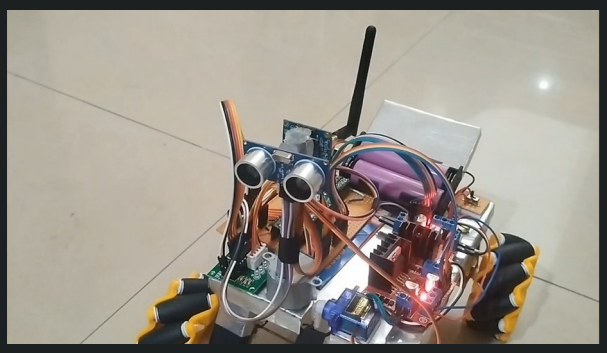
Here’s a snippet of my main loop code:

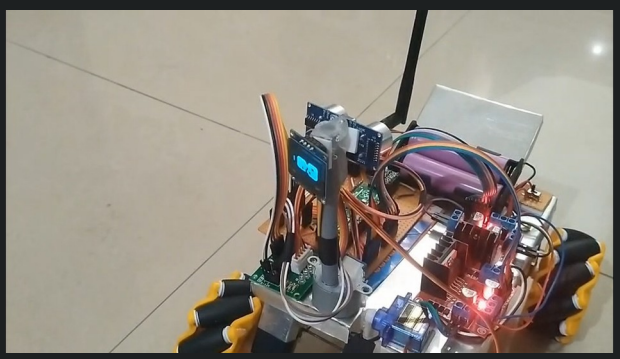
```
    updateJoystickDot(JOY_X_CENTER, JOY_Y_CENTER);
}

void loop() {
    int rawX = analogRead(JOY_X);
    int rawY = 4095 - analogRead(JOY_Y);  // joystick ke input

    bool kickBtn = digitalRead(BTN_KICK);
    if (kickBtn == LOW && lastKickBtn == HIGH) {  //kick waala button
        kickActive = true;
        kickTime   = millis();
        drawKickButton(true);
        addLog("Kicked");
        Serial.println("KICK");
    }
    if (kickActive && millis() - kickTime > 300) {
        kickActive = false;
        drawKickButton(false);
    }
    lastKickBtn = kickBtn;

    bool modeBtn = digitalRead(BTN_MODE);
    if (modeBtn == LOW && lastModeBtn == HIGH) {   //change mode from eyes to sensor
        eyesMode = !eyesMode;
        Serial.print("eyesMode is now: ");
        Serial.println(eyesMode);
        drawHeader();
        drawModeButton();
        addLog(eyesMode ? "Eyes" : "Sensor");
    }
    lastModeBtn = modeBtn;

    updateJoystickDot(rawX, rawY);

    // NRF stubbed — uncomment when module arrives
    // ControlData data = { rawX, rawY, kickActive, eyesMode };
```

As for the Pi Pico, I tried using micropython at first, but because I am not very good at micropython, I tried switching to the C++ that I'm more comfortable with. The setup that I could get working was that I used Vscode on my Linux machine and then installed arduino-cli package seperately, and then I installed the Community Maintained fork of the Arduino extension for Vscode, and finally that worked. I know it's probably not the best method, but again, it works!

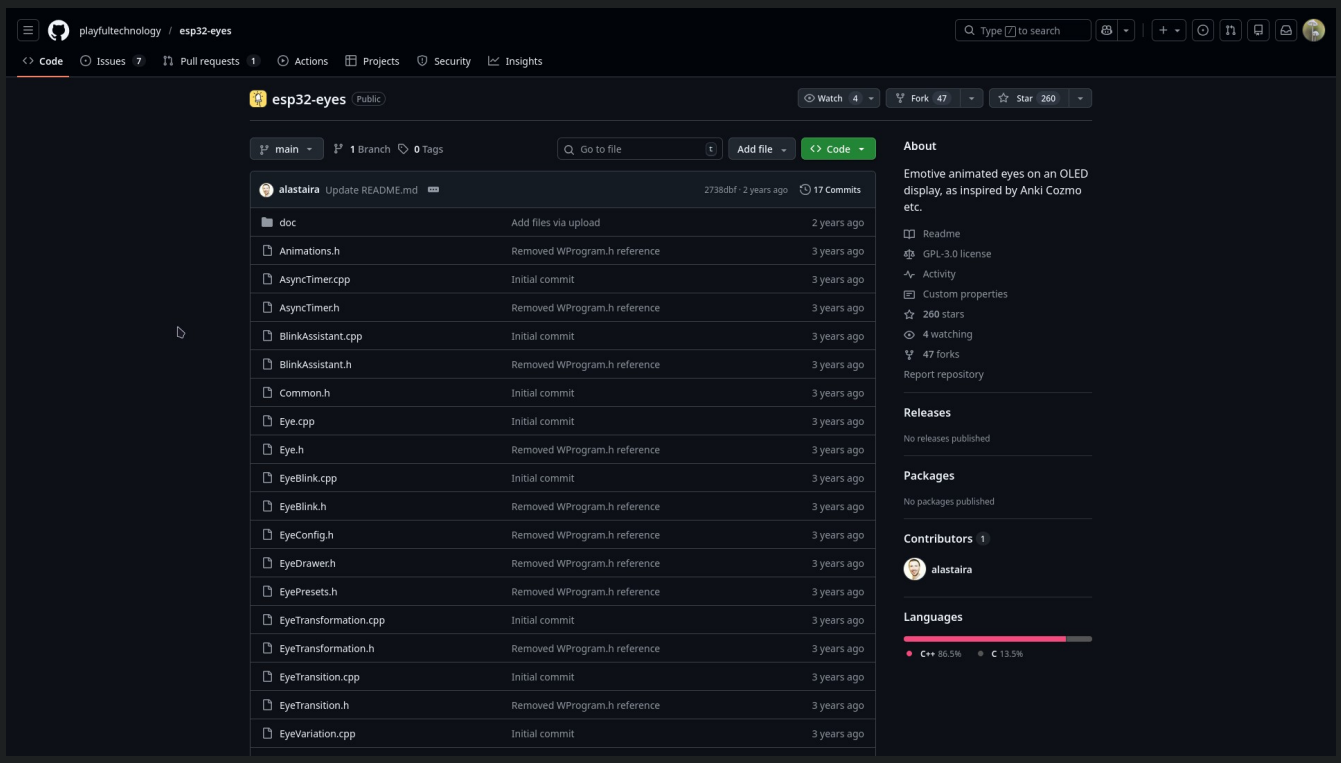
My code has different header files for all the different things I have added, like one for the stepper, where I stored all of the pins I need to power in order to make the stepper move in that direction. I used full stepping instead of half stepping to increase the speed.

The head part of the bot has 2 modes, one is the sensor mode, and the other is the screen mode.





In the screen mode it displays animated eyes onto the screen using a library I found made by the user 'playfultechnology' over on github.



I do not take credit for that part of the code as I simply just used it. I hope that is fine. The sensor part stops the bot from crashing into walls by stopping it if it detects that it is within 20cm of an obstacle. I also planned on having the distance data sent to the dashboard on the transmitter, but these NRF modules cant only act exclusively as a transmitter or a receiver, not both at the same time. Maybe there was some way I could've done it, but I didn't figure it out in time.

Summary -

So yeah… This is my project for you to see. I hope it is to your guys' liking. If possible I hope you'll conact me with feedback on this project, and if I;m lucky, maybe I'll even get a chance to visit the open house event🤞. That's all I got!