

**REPORT
PERSONAL PROJECT**

SELF-BALANCING ROBOT

**AUTHOR: NGUYEN TUAN VINH
GITHUB: SELF-BALANCING ROBOT ON GITHUB**

HO CHI MINH CITY, 08/2024

TABLE OF CONTENTS

TABLE OF CONTENTS	1
LIST OF TABLES	4
LIST OF IMAGES	5
Program	6
1 Overview	7
1.1 Introduction	8
1.2 Project Objectives	8
1.3 Scope	8
1.4 System Overview	8
1.5 Methodology	9
2 Components	10
2.1 Softwares	11
2.1.1 Integrated development environment (IDE)	11
2.1.2 Languages	11
2.1.3 Frameworks	11
2.1.4 Libraries	11
2.2 Hardwares	12
2.2.1 ESP32 WROOM 32	12
2.2.2 MPU 6050	14
2.2.3 I2C OLED 0.96 inches	15
2.2.4 Buttons	17
2.2.5 Li-on battery	17
2.2.6 2-pins pack	18
2.2.7 Ceramic capacitors 104 100nF	19
2.2.8 L298N motor driver	20
2.2.9 DC motors and wheels	21
2.2.10 Jump wires	21
3 Implement plan	23
3.1 Project's tasks	24
3.2 Project timeline	24
4 Schematic	25

5 Block diagrams	27
5.1 MPU Reader	28
5.1.1 Responsibilities	28
5.1.2 Data Received	28
5.1.3 Data Transmitted	28
5.1.4 Communication	28
5.2 PID Block	28
5.2.1 Responsibilities	29
5.2.2 Data Received:	29
5.2.3 Data Transmitted:	29
5.2.4 Communication:	29
5.3 Motor Controller	29
5.3.1 Responsibilities	29
5.3.2 Data Received:	29
5.3.3 Data Transmitted:	29
5.3.4 Communication:	29
5.4 Center Controller	30
5.4.1 Responsibilities	30
5.4.2 Data Received:	30
5.4.3 Data Transmitted:	30
5.4.4 Communication:	30
5.5 Button Reader	30
5.5.1 Responsibilities	30
5.5.2 Data Received:	30
5.5.3 Data Transmitted:	30
5.5.4 Communication:	30
5.6 Display Controller	30
5.6.1 Responsibility:	30
5.6.2 Data Received:	31
5.6.3 Data Transmitted:	31
5.6.4 Communication:	31
6 Scripts	32
6.1 MPU Reader	33
6.1.1 mpu_reader.hpp	33
6.1.2 mpu_reader.cpp	34
6.2 PID Block	38
6.2.1 PID_block.hpp	38
6.2.2 PID_block.cpp	39
6.3 Motor Controller	43
6.3.1 motor_controller.hpp	43
6.3.2 motor_controller.cpp	45
6.4 Center Controller	53
6.4.1 center_controller.hpp	53
6.4.2 center_controller.cpp	55
6.5 Button Reader	60
6.5.1 button_reader.hpp	60

6.5.2	button_reader.cpp	61
6.6	Display Controller	62
6.6.1	display_controller.hpp	62
6.6.2	display_controller.cpp	64
7	Demo		90

LIST OF TABLES

3.1 All project tasks	24
---------------------------------	----

LIST OF IMAGES

2.1	Visual Studio Code Logo	11
2.2	PLatformIO logo	11
2.3	ESP 32 WROOM 32	13
2.4	ESP 32 WROOM 32 pinouts	14
2.5	MPU 6050	15
2.6	I2C OLED 0.96 inches (128x64 pixels)	16
2.7	Button	17
2.8	Li-on battery 3.7V 1200mAh	18
2.9	2-pins pack	19
2.10	Ceramic capacitors 104 100nF	19
2.11	L298N motor driver	20
2.12	DC motor and wheels	21
2.13	Arduino UNO and Adapter	22
3.1	Project timeline	24
4.1	Self-balancing robot schematic	26
5.1	Block diagram of Robot	28
7.1	Self-Balancing Robot front view	91
7.2	Self-Balancing Robot up view	92
7.3	Self-Balancing Robot back view	93
7.4	Menu UI	94
7.5	PID parameters UI	95
7.6	Modify K _x UI	96
7.7	Start and plot UI	97

Listings

2.1	All library declaration in platformio.init	12
6.1	mpu_reader.hpp	33
6.2	mpu_reader.cpp	34
6.3	PID.block.hpp	38
6.4	PID.block.cpp	39
6.5	motor_controller.hpp	43
6.6	motor_controller.cpp	45
6.7	center_controller.hpp	54
6.8	center_controller.cpp	55
6.9	button_reader.hpp	60
6.10	button_reader.cpp	61
6.11	display_controller.hpp	62
6.12	display_controller.cpp	64

Chapter 1

Overview

Overview of the project

1.1 Introduction

This project presents the design, implementation, and control of a self-balancing two-wheeled robot. Unlike traditional self-balancing robots, this project incorporates a user interface (UI) for initialization, parameter observation, and tuning of the PID controller, including K_p, K_i, and K_d gains, as well as a setpoint. These parameters are stored in the ESP32's non-volatile memory to ensure data persistence even after power cycles. Self-balancing robots have gained significant attention in recent years due to their potential applications in various fields, such as transportation, robotics, and education.

1.2 Project Objectives

The primary objectives of this project are:

- To design and construct a stable self-balancing two-wheeled robot equipped with a user-friendly interface.
- To implement effective control algorithms, including a tunable PID controller, to maintain balance.
- To store control parameters in non-volatile memory for persistent configuration.
- To evaluate the robot's performance under various conditions and with different parameter settings.

1.3 Scope

This project encompasses the mechanical design, electrical system, control algorithm development, user interface design, and experimental testing of the self-balancing robot. The scope includes:

- Designing the physical structure of the robot.
- Selecting appropriate sensors (accelerometers, gyroscopes, encoders), actuators (motors), and microcontroller (ESP32).
- Developing control software using [Programming language, e.g., C++] and a user interface using [UI framework, e.g., Arduino IDE, ESP-IDF].
- Implementing a PID controller with parameters stored in non-volatile memory.
- Conducting experiments to evaluate the robot's stability, responsiveness, and the effectiveness of the user interface.

1.4 System Overview

The self-balancing robot comprises several key components:

- Mechanical structure: A lightweight frame, two wheels, and a platform for carrying electronics.
- Sensors: Accelerometers, gyroscopes, and encoders to measure the robot's orientation and wheel speed.
- Actuators: DC motors to drive the wheels.
- Microcontroller: An ESP32 microcontroller for processing sensor data, controlling the motors, and handling user interface interactions.

- Power supply: Batteries to power the system.
- User interface: A display or a mobile app to visualize sensor data, adjust PID parameters, and control the robot.

1.5 Methodology

The following methodology was adopted for this project:

- Literature review: A comprehensive review of self-balancing robot designs, control algorithms, and user interface development.
- System design: Creation of detailed design drawings and selection of components.
- Software development: Implementation of control algorithms, user interface, and data storage in non-volatile memory.
- Hardware assembly: Construction of the robot and integration of all components.
- Testing and evaluation: Conducting experiments to assess the robot's performance, the effectiveness of the user interface, and the stability of the stored parameters.

Chapter 2

Components

All softwares, library and components are used for implementing the project.

2.1 Softwares

2.1.1 Integrated development environment (IDE)

I used Visual Studio Code for development.



Figure 2.1: Visual Studio Code Logo

On VS Code, I installed PlatformIO for scripting and upload code to the Arduino board.



Figure 2.2: PlatformIO logo

2.1.2 Languages

- C++

2.1.3 Frameworks

Arduino framework

2.1.4 Libraries

In PlatformIO, I used:

- Adafruit SSD1306 - version:2.5.10
 - Author: Adafruit Industries

- *Purpose:* Drives an SSD1306 OLED display. In the context of a self-balancing robot, this library is likely used to display sensor data, control parameters, or debugging information on the OLED screen.

- **Adafruit GFX Library** - version:1.11.9

- *Author:* Adafruit Industries
- *Purpose:* Provides a graphics library for drawing shapes, text, and images on displays. It is used in conjunction with the SSD1306 library to create visual output on the OLED screen.

- **Adafruit Unified Sensor** - version:1.1.14

- *Author:* Adafruit Industries
- *Purpose:* Offers a unified interface for interacting with various sensors. In this case, it's likely used to provide a consistent way to access data from the MPU6050 sensor.

- **Adafruit MPU6050** - version:2.2.6

- *Author:* Adafruit Industries
- *Purpose:* Interacts with the MPU6050 6-axis accelerometer and gyroscope. This is a core component for obtaining sensor data for the self-balancing robot's control system.

- **Adafruit BusIO** - version:1.16.1

- *Author:* Adafruit Industries
- *Purpose:* Provides low-level I/O access for various bus protocols, including I2C. It's used to communicate with sensors like the MPU6050.

All library declarations in *platformio.ini*:

```

1 lib_deps =
2   adafruit/Adafruit SSD1306@^2.5.10
3   adafruit/Adafruit GFX Library@^1.11.9
4   adafruit/Adafruit Unified Sensor@^1.1.14
5   adafruit/Adafruit MPU6050@^2.2.6
6   adafruit/Adafruit BusIO@^1.16.1

```

Program 2.1: All library declaration in platformio.init

2.2 Hardwares

2.2.1 ESP32 WROOM 32

The ESP32 WROOM-32 is a powerful and versatile microcontroller module developed by Espressif Systems. It's built around the Xtensa LX6 microprocessor and incorporates Wi-Fi and Bluetooth connectivity. This combination makes it an ideal choice for IoT (Internet of Things) applications, robotics, and various embedded systems.



Figure 2.3: ESP 32 WROOM 32

Key features:

- **Dual-Core Processor:** The ESP32 features two Xtensa LX6 cores capable of running at up to 240 MHz, providing ample processing power for complex tasks.
- **Wi-Fi and Bluetooth Connectivity:** Built-in support for both Wi-Fi (802.11 b/g/n/e/i) and Bluetooth (Bluetooth Classic, Bluetooth Low Energy) enables seamless wireless communication.
- **Low Power Consumption:** The ESP32 offers multiple power-saving modes, making it suitable for battery-powered devices.
- **Rich Peripheral Set:** Includes ADC, DAC, PWM, I2C, SPI, UART, and other peripherals for interfacing with various sensors and actuators.
- **Secure Boot and Encryption:** Provides hardware-based security features for protecting your applications.
- **Extensive GPIOs:** Numerous General Purpose Input/Output pins offer flexibility for connecting external components.
- **Development Tools:** Well-supported by various development environments and frameworks, including Arduino, ESP-IDF, and MicroPython.

Pinout:

ESP32 DEVKIT V1 – DOIT

version with 36 GPIOs

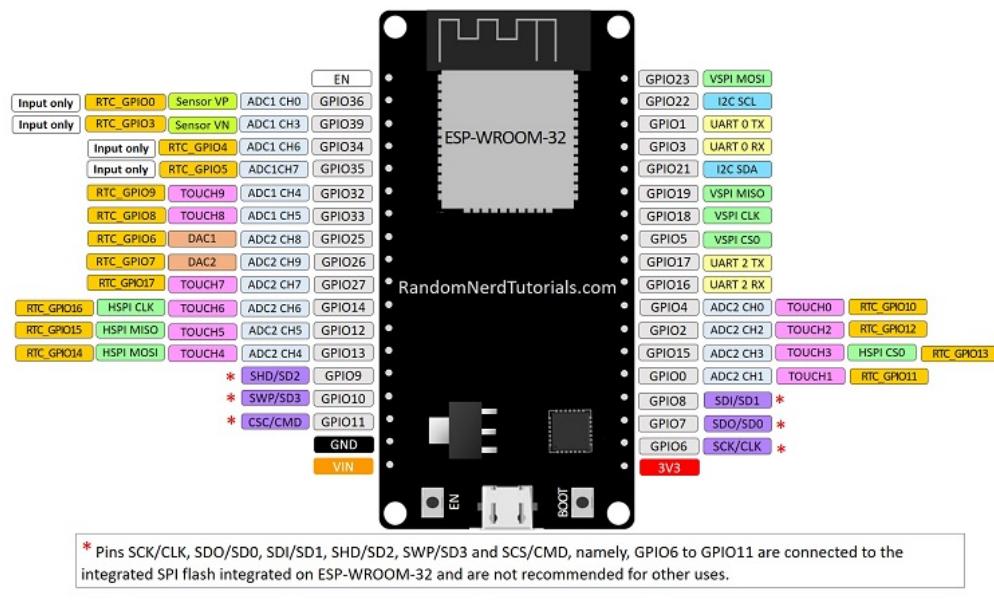


Figure 2.4: ESP 32 WROOM 32 pinouts

The ESP32 WROOM-32 module has a rich set of pins, including:

- GPIOs: General Purpose Input/Output pins for digital I/O.
- ADC: Analog-to-Digital Converter pins for reading analog signals.
- PWM: Pulse Width Modulation pins for controlling motor speeds, LED brightness, etc.
- I2C, SPI, UART: Serial communication interfaces for connecting to various sensors and modules.
- Power Supply Pins: VCC, GND for power supply.
- Reset Pin: For resetting the module.
- Boot Mode Pins: For selecting different boot modes (e.g., bootloader, flash, etc.).
- Antenna Pins: For connecting external antennas (optional).

2.2.2 MPU 6050

The MPU6050 is a crucial component in a self-balancing robot, serving as its "sense of balance." It's a 6-axis Inertial Measurement Unit (IMU) that provides real-time data on the robot's orientation and movement.

Its core responsibilities include:

- Measuring acceleration: Determines the robot's linear acceleration in three axes (X, Y, Z). This data is essential for calculating the robot's tilt or inclination.
- Measuring angular velocity: Provides information about the robot's rotational speed around the three axes (pitch, roll, and yaw). This data is vital for maintaining balance and making corrections.

By combining these measurements, the robot's control system can accurately determine its orientation and adjust motor speeds accordingly to maintain equilibrium.

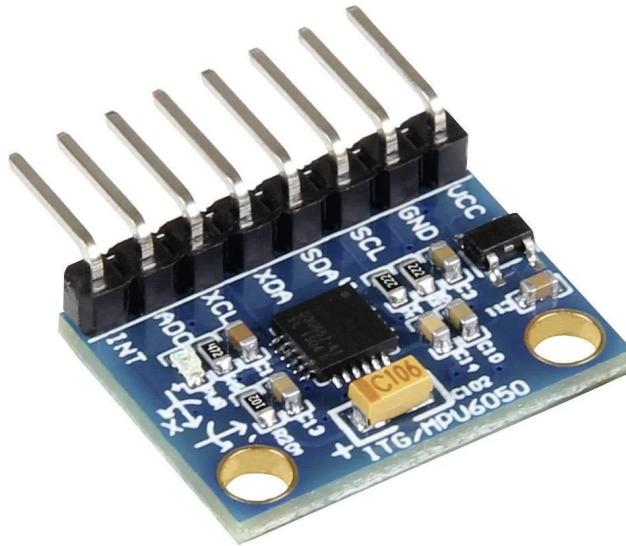


Figure 2.5: MPU 6050

Pinout and Wiring

The MPU6050 typically communicates via the I2C interface. The primary pins are:

- SCL (Serial Clock): Clock signal for I2C communication.
- SDA (Serial Data): Data line for I2C communication.
- VCC: Power supply (typically 3.3V or 5V).
- GND: Ground.

Specifications:

- 6-axis IMU: Combines a 3-axis accelerometer and a 3-axis gyroscope in a single chip.
- Accelerometer: Measures acceleration in g (gravity units).
- Gyroscope: Measures angular velocity in degrees per second (dps).
- I2C interface: For communication with a microcontroller.
- Operating voltage: Typically 3.3V or 5V.
- Low power consumption: Suitable for battery-powered applications.
- Small form factor: Easy to integrate into robot designs.

2.2.3 I2C OLED 0.96 inches

A 0.96-inch I2C OLED display is a valuable addition to a self-balancing robot, providing a visual interface for:

- Displaying real-time data: Show crucial information such as battery level, sensor readings (accelerometer, gyroscope), motor speeds, and control parameters.
- Debugging: Aid in troubleshooting by displaying error messages, sensor values, or control loop outputs.
- User interaction: Provide basic user input options like mode selection or parameter adjustment.

- Enhancements: Offer additional features like displaying robot status, balance indicators, or graphical representations of sensor data.



Figure 2.6: I2C OLED 0.96 inches (128x64 pixels)

Pinout: Typical pinouts for an I2C OLED display include:

- SCL (Serial Clock): For I2C clock signal.
- SDA (Serial Data): For I2C data transfer.
- VCC: Power supply (usually 3.3V or 5V).
- GND: Ground.

Specifications:

- Display size: 0.96 inches diagonal.
- Resolution: Typically 128x64 pixels.
- Interface: I2C communication.
- Power consumption: Low power consumption, suitable for battery-powered robots.
- Brightness control: Adjustable brightness for different lighting conditions.
- Controller: Often uses a SSD1306 or SH1106 controller.
- Library support: Ensure your microcontroller platform has a compatible library for driving the OLED display (e.g., Adafruit_SSD1306 for Arduino).
- Power supply: Provide a stable power supply to the OLED to prevent display issues.
- Level shifting: If your microcontroller operates at a different voltage than the OLED, you might need level shifting circuits.

2.2.4 Buttons

There are 4 buttons used to interact with the interface. These buttons correspond to the following tasks:

- ESC (escape button): responsible for exiting the current interface and returning to the previous interface.
- OK (confirm button): responsible for switching to the selected interface or confirming changes to the parameters.
- UP (up arrow button): responsible for moving up or increasing the parameter value.
- DOWN (down arrow button): responsible for moving down or decreasing the parameter value.



Figure 2.7: Button

2.2.5 Li-on battery

A 3.7V 1200mAh Li-ion battery is the primary power source for a self-balancing robot. In this project, I was using 2 batteries in series to create a 7.4V source.



Figure 2.8: Li-on battery 3.7V 1200mAh

Specifications

- Voltage: 3.7V nominal (typical operating voltage).
- Capacity: 1200mAh (milliampere-hours), indicating it can deliver 1.2 amperes for one hour.
- Chemistry: Lithium-ion (Li-ion), a type of rechargeable battery known for high energy density.
- Charging: Requires a specific Li-ion charger with appropriate voltage and current limits to prevent damage.
- Safety: Contains protective circuitry to prevent overcharging, over-discharging, and short circuits.
- Battery Management System (BMS): For larger battery packs or more complex applications, a BMS is recommended to monitor battery voltage, current, temperature, and prevent cell imbalance.
- Discharge Rate: The battery's C-rating indicates its maximum discharge current. Ensure it's sufficient for your robot's motor requirements.
- Weight: Consider the battery's weight, as it affects the robot's overall balance and maneuverability.
- Safety Precautions: Li-ion batteries can be hazardous if mishandled. Always follow safety guidelines for charging, storage, and handling.

2.2.6 2-pins pack

I used it to install 2 batteries to power the robot.



Figure 2.9: 2-pins pack

2.2.7 Ceramic capacitors 104 100nF

Ceramic capacitors, specifically those with a designation like "104", play a crucial role in a self-balancing robot by acting as noise filters. Their primary function is to suppress electrical noise generated by the robot's components, particularly the motors.



Figure 2.10: Ceramic capacitors 104 100nF

Specifications The designation "104" for a ceramic capacitor refers to its capacitance value. The code represents the capacitance in picofarads (pF). In this case, "104" translates to $10 \times 10^4 pF$, which equals 100,000 pF or 0.1 microfarads (uF).

Other key specifications for ceramic capacitors include:

- Voltage rating: The maximum voltage the capacitor can withstand.
- Temperature coefficient: How the capacitance changes with temperature.
- Capacitance tolerance: The allowable deviation from the specified capacitance value.
- Physical size: The dimensions of the capacitor.

2.2.8 L298N motor driver

The L298N module is a crucial component in a self-balancing robot, serving as the bridge between the microcontroller and the robot's motors. Its primary function is to control the direction and speed of the motors based on the signals from the microcontroller.

- Motor control: Amplifies the low-current signals from the microcontroller to drive higher-current motors.
- Direction control: Enables the reversal of motor direction for forward and backward movement.
- Current limiting: Provides basic overcurrent protection for the motors.

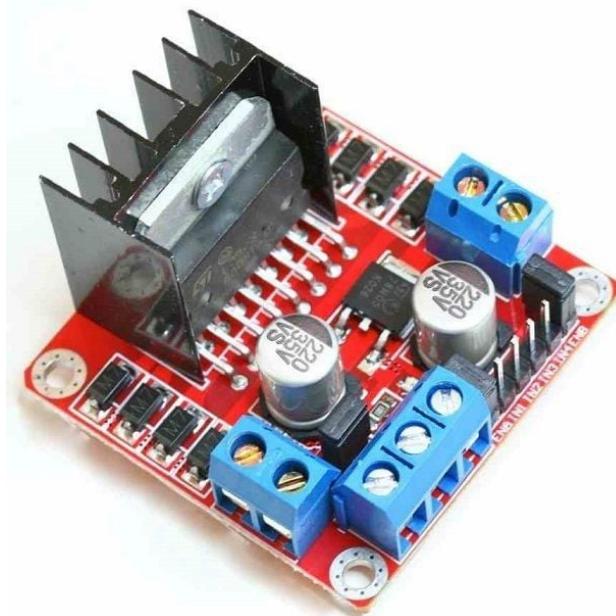


Figure 2.11: L298N motor driver

Pinout: The L298N module typically has the following pins:

- Input pins:
 - Enable pins (ENA, ENB): Control the enable state of each motor driver.
 - Input pins (IN1, IN2, IN3, IN4): Determine the direction of each motor.
- Output pins:
 - Output pins (OUT1, OUT2, OUT3, OUT4): Connect to the motor terminals.
- Power supply pins:
 - VCC: Power supply for the logic section.
 - GND: Ground.
 - VS: Power supply for the motor outputs.

Specifications

- Voltage: Typically operates on a supply voltage of 5V for the logic section and 12V or higher for the motor outputs.

- Current: Can handle a maximum current of around 2A to 3A per motor.
- Number of motors: Can control two DC motors simultaneously.
- Operating temperature: Typically operates within a temperature range of -25°C to +130°C.
- Package: Available in DIP or SMD packages.

2.2.9 DC motors and wheels

DC motors are the heart of a self-balancing robot, providing the necessary propulsion to maintain balance and movement.

- Propulsion: Convert electrical energy into mechanical energy to drive the robot.
- Balance: Adjust motor speeds to counteract imbalances and maintain upright position.
- Maneuverability: Allow for forward, backward, and turning movements.



Figure 2.12: DC motor and wheels

2.2.10 Jump wires

Jump wires, often referred to as jumper wires or connecting wires, serve as the electrical interconnections between various components in your self-balancing robot. They establish the pathways for data and power to flow between the microcontroller, sensors, motors, and other electronic modules.

- Component connectivity: Connect different parts of the robot's circuitry.
- Prototyping: Allow for rapid prototyping and testing of different configurations.
- Temporary connections: Provide flexible connections during assembly and troubleshooting.



Figure 2.13: Arduino UNO and Adapter

Kinds of Jump Wires are used:

- Male-to-Male Jump Wires:
 - Both ends have male pin connectors.
 - Commonly used for connecting male headers on components.
- Female-to-Female Jump Wires:
 - Both ends have female pin connectors.
 - Less common but can be useful in specific situations.
- Male-to-Female Jump Wires:
 - One end has a male pin connector, the other has a female pin connector.
 - The most versatile type, allowing connections between male and female headers.

Chapter 3

Implement plan

You can visit the project's plan on Jira by clicking [here](#). Be sure you have a Jira account and logged in already.

3.1 Project's tasks

With the goal of completing the project in 14 days, I have planned the implementation as shown in the table below.

ID	Task name	Duration (day)
SBR-1	Researching Information	2 (July 9, 2024 → July 10, 2024)
SBR-2	Component Selection	1 (July 11, 2024)
SBR-3	Designing Robot's 3D Model	7 (July 11, 2024 → July 17, 2024)
SBR-4	Electrical Circuit Design	3 (July 11, 2024 → July 13, 2024)
SBR-5	Component Assembly	2 (July 14, 2024 → July 15, 2024)
SBR-6	Robot Chassis Building	2 (July 19, 2024 → July 20, 2024)
SBR-7	Component Calibration	1 (July 18, 2024)
SBR-8	Balancing Algorithm Implementation	4 (July 21, 2024 → July 24, 2024)
SBR-9	Block Diagram Design	2 (July 25, 2024 → July 26, 2024)
SBR-10	Create Project and Configuration	4 (July 11, 2024 → July 14, 2024)
SBR-11	Initial Testing and Debugging	1 (July 27, 2024)
SBR-12	Balancing Algorithm Tuning	2 (July 28, 2024 → July 29, 2024)
SBR-13	Documentation	23 (July 9, 2024 → July 31, 2024)

Table 3.1: All project tasks

3.2 Project timeline

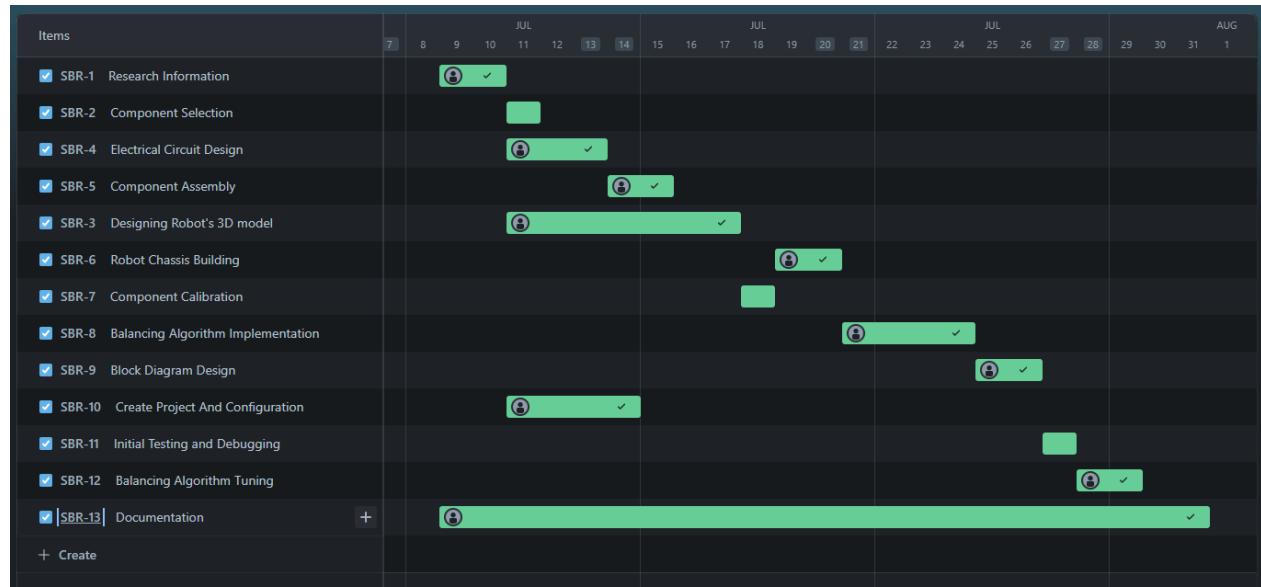


Figure 3.1: Project timeline

Chapter 4

Schematic

Include Robot Schematic

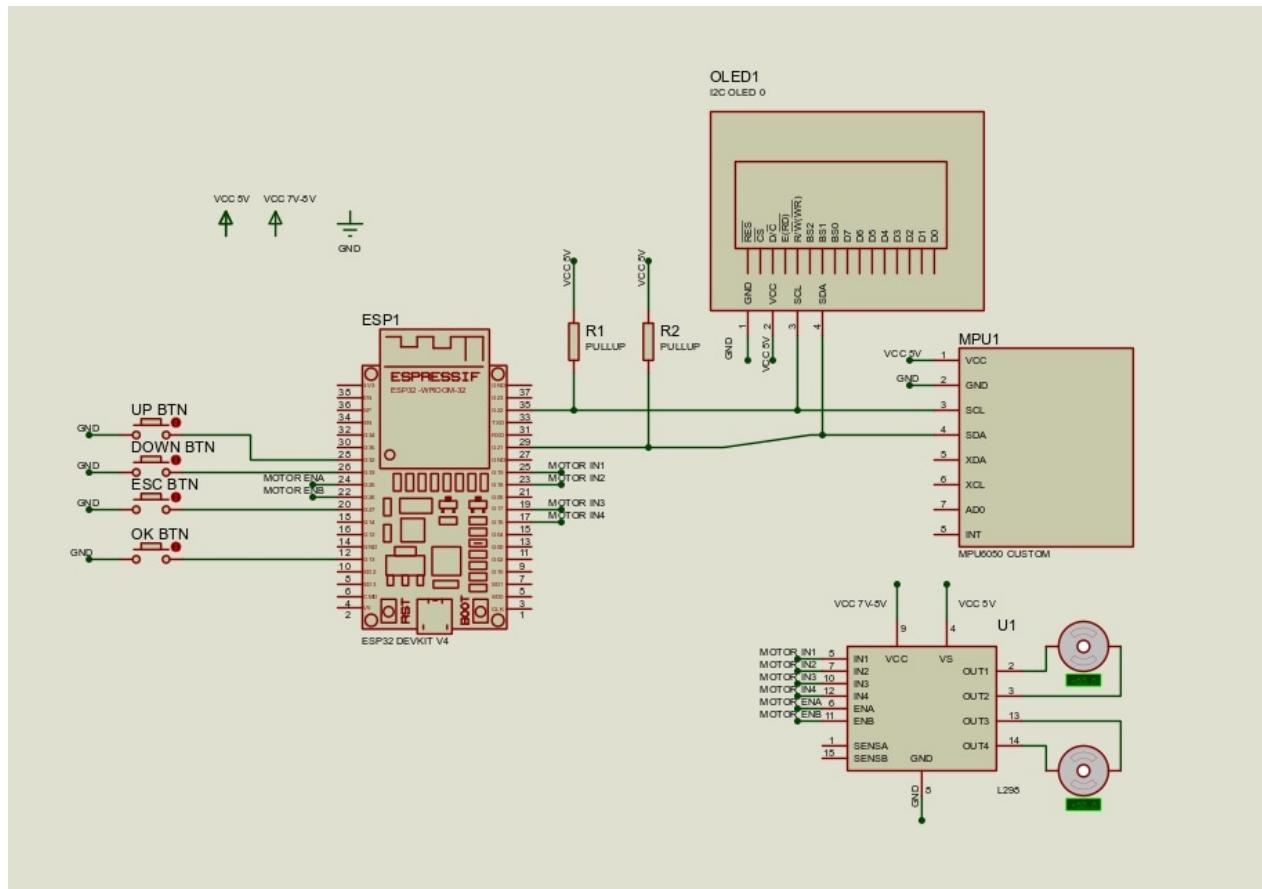


Figure 4.1: Self-balancing robot schematic

Chapter 5

Block diagrams

The block diagram section of the self-balancing robot implementation report provides a visual overview of the system's architecture, illustrating the interconnections between key components such as sensors, controllers, and actuators. It highlights how data flows from the sensors to the microcontroller, where algorithms process the information to maintain balance, and how the output signals are sent to the motors to adjust the robot's position accordingly

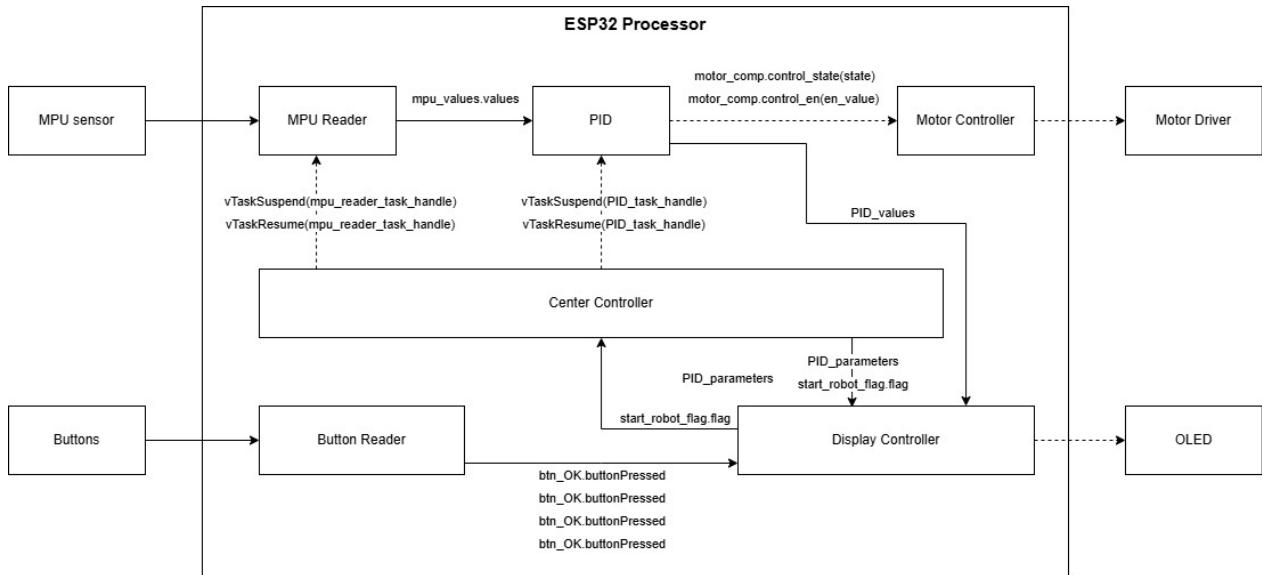


Figure 5.1: Block diagram of Robot

5.1 MPU Reader

The MPU Reader is responsible for acquiring motion data from the MPU6050 sensor, which is an Inertial Measurement Unit (IMU) that provides acceleration and gyroscope readings. This data is essential for understanding the orientation and movement of the system, typically used in balance control systems such as self-balancing robots.

5.1.1 Responsibilities

- Initialize and configure the MPU6050 sensor.
- Continuously read accelerometer and gyroscope data.
- Transmit the raw data to the PID Block for further analysis.

5.1.2 Data Received

Raw accelerometer and gyroscope data from the MPU6050 sensor.

5.1.3 Data Transmitted

Raw motion data (acceleration and angular velocity) sent to the Center Controller.

5.1.4 Communication

To PID Block: The MPU Reader communicates with the PID Block by transmitting the raw motion data, which is used to calculate the error in the system's current state compared to the desired state.

5.2 PID Block

The PID Block implements a Proportional-Integral-Derivative (PID) controller, which is used to compute the control signals needed to correct any deviations in the system's behavior from the desired setpoint. The PID Block is crucial in achieving stability and desired performance in control systems.

5.2.1 Responsibilities

- Implements a PID control algorithm to calculate control outputs based on sensor data and desired setpoint.
- Determines motor control signals based on PID outputs.
- Communicates control signals to the motor controller.

5.2.2 Data Received:

- Sensor data (acceleration and angular velocities) from the MPU reader.
- Desired Kp, Ki, Kd and setpoint from the center controller.

5.2.3 Data Transmitted:

- Control signals (motor speeds or directions) to the motor controller.
- PID parameters (Kp, Ki, Kd, setpoint) to the display controller (for UI).

5.2.4 Communication:

Receives sensor data from the MPU reader through a semaphore-based mechanism. Sends control signals to the motor controller. Sends filtered_pitch, gyro_angle_Y, output, error, integral, derivative to Display Controller. Communicates with the center controller for receiving Kp, Ki, Kd and setpoints.

5.3 Motor Controller

The Motor Controller is responsible for managing the operation of the motors based on the control signals it receives. It directly influences the physical movement and behavior of the system by adjusting the speed and direction of the motors.

5.3.1 Responsibilities

- Controls the movement of the robot by driving the motors.
- Receives control signals from the PID block and converts them into appropriate motor commands.

5.3.2 Data Received:

Control signals (motor speeds or directions) from the PID block.

5.3.3 Data Transmitted:

Motor status (e.g., current state, errors) to the center controller (optional).

5.3.4 Communication:

Directly controls motor driver pins using analogWrite() function.

5.4 Center Controller

The Center Controller acts as the heart of the system. It controls whether the components continue to work or stop temporarily.

5.4.1 Responsibilities

Orchestrates the overall system behavior.

5.4.2 Data Received:

Receiving star_robot_flag from Display Controller.

5.4.3 Data Transmitted:

Transmitting star_robot_flag to Display Controller

5.4.4 Communication:

Uses semaphores for synchronization between components.

5.5 Button Reader

The Button Reader is responsible for detecting user inputs through physical buttons on the system. These inputs are used to navigate the user interface, adjust settings, or trigger specific actions within the system.

5.5.1 Responsibilities

- Monitor the physical buttons for user interactions.
- Debounce the button inputs to ensure reliable readings.
- Translate button presses into specific commands or actions.
- Communicate these commands to the Display Controller or directly to the Center Controller if necessary.

5.5.2 Data Received:

Button press/release events from hardware interrupts.

5.5.3 Data Transmitted:

Button press/release events to the center controller.

5.5.4 Communication:

Uses interrupts for real-time button detection. Communicates button states to the center controller.

5.6 Display Controller

5.6.1 Responsibility:

Provides a user interface for displaying system information and allowing user interaction. Receives data from other components to display. Handles user input from buttons.

5.6.2 Data Received:

Sensor data (indirectly through the PID block) PID parameters from the center controller Button states from the button reader

5.6.3 Data Transmitted:

User input (button presses) to the center controller Updated PID parameters to the center controller

5.6.4 Communication:

Receives data from other components through shared memory or message passing (depending on implementation). Directly interacts with the display hardware. Uses semaphores for synchronization when accessing shared data.

Chapter 6

Scripts

Source Code for this project. You can visit project's repository on github to see detail. In this repository, I created 4 branch:

- **main:** This contains main script for robot,
- **PID_block_component:** This contains script for PID implementation.
- **display_controller_feature:** This contains script for displaying UI on OLED.
- **mpu_reader_feature:** This contains script for reading acceleration and gyroscope values.
- **cleaning_code:** Change code style to OOP and cleaning all script.

6.1 MPU Reader

6.1.1 mpu_reader.hpp

```
1 #pragma once
2 #ifndef MPU_READER_HPP
3 #define MPU_READER_HPP
4
5 #include <Adafruit_MPU6050.h>
6 #include <Adafruit_Sensor.h>
7 #include <Wire.h>
8 #include <Arduino.h>
9 #include <global.hpp>
10
11 class mpu_reader
12 {
13 private:
14
15     const char* TAG = "MPU READER";
16     const int8_t sample_time;
17     const uint8_t priority;
18
19     Adafruit_MPU6050* mpu;
20     TaskHandle_t mpu_reader_task_handle;
21
22     struct{
23         // QueueHandle_t queue;
24         SemaphoreHandle_t bi_semaphore;
25         struct_mpu_reader values;
26     } mpu_values;
27
28     Adafruit_Sensor *mpu_accel;
29     Adafruit_Sensor *mpu_gyro;
30
31     void sensors_setup();
32
33     void mpu_reading();
34
35     static void mpu_reading_wrapper(void* arg);
36
37 public:
38
39     TaskHandle_t get_task_handle();
40
41     // QueueHandle_t get_mpu_values_queue();
42
43     void run();
```

```

44
45     SemaphoreHandle_t get_semaphore_mpu();
46
47     struct_mpu_reader get_mpu_values();
48
49     mpu_reader(int8_t sample_time = 20, uint8_t priority = 6);
50
51     ~mpu_reader();
52 };
53
54 extern mpu_reader* mpu_reader_component;
55
56 #endif

```

Program 6.1: mpu_reader.hpp

6.1.2 mpu_reader.cpp

```

1 #include "mpu_reader.hpp"
2
3 void mpu_reader::sensors_setup(){
4     // this->mpu->setAccelerometerRange(MPU6050_RANGE_2_G);
5     // Serial.print("Accelerometer range set to: ");
6     // switch (this->mpu->getAccelerometerRange()) {
7     // case MPU6050_RANGE_2_G:
8     //     Serial.println("+-2G");
9     //     break;
10    // case MPU6050_RANGE_4_G:
11    //     Serial.println("+-4G");
12    //     break;
13    // case MPU6050_RANGE_8_G:
14    //     Serial.println("+-8G");
15    //     break;
16    // case MPU6050_RANGE_16_G:
17    //     Serial.println("+-16G");
18    //     break;
19    // }
20    // this->mpu->setGyroRange(MPU6050_RANGE_250_DEG);
21    // Serial.print("Gyro range set to: ");
22    // switch (this->mpu->getGyroRange()) {
23    // case MPU6050_RANGE_250_DEG:
24    //     Serial.println("+- 250 deg/s");
25    //     break;
26    // case MPU6050_RANGE_500_DEG:
27    //     Serial.println("+- 500 deg/s");
28    //     break;
29    // case MPU6050_RANGE_1000_DEG:

```

```
30     //      Serial.println("+- 1000 deg/s");
31     //      break;
32     // case MPU6050_RANGE_2000_DEG:
33     //      Serial.println("+- 2000 deg/s");
34     //      break;
35     //}
36
37     // this->mpu->setFilterBandwidth(MPU6050_BAND_21_HZ);
38     // Serial.print("Filter bandwidth set to: ");
39     // switch (this->mpu->getFilterBandwidth()) {
40     // case MPU6050_BAND_260_HZ:
41     //     Serial.println("260 Hz");
42     //     break;
43     // case MPU6050_BAND_184_HZ:
44     //     Serial.println("184 Hz");
45     //     break;
46     // case MPU6050_BAND_94_HZ:
47     //     Serial.println("94 Hz");
48     //     break;
49     // case MPU6050_BAND_44_HZ:
50     //     Serial.println("44 Hz");
51     //     break;
52     // case MPU6050_BAND_21_HZ:
53     //     Serial.println("21 Hz");
54     //     break;
55     // case MPU6050_BAND_10_HZ:
56     //     Serial.println("10 Hz");
57     //     break;
58     // case MPU6050_BAND_5_HZ:
59     //     Serial.println("5 Hz");
60     //     break;
61     //}
62
63     this->mpu_accel = this->mpu->getAccelerometerSensor();
64     this->mpu_accel->printSensorDetails();
65
66     this->mpu_gyro = this->mpu->getGyroSensor();
67     this->mpu_gyro->printSensorDetails();
68 }
69
70 void mpu_reader::mpu_reading(){
71     sensors_event_t accel, gyro;
72
73     vTaskSuspend(this->mpu_reader_task_handle);
74
75     while (true)
```

```
76     {
77
78         if(this->mpu_accel->getEvent(&accel)){
79             this->mpu_values.values.accel = accel.acceleration;
80         }
81
82         if(this->mpu_gyro->getEvent(&gyro)){
83             this->mpu_values.values.gyro = gyro.gyro;
84         }
85
86         xSemaphoreGive(this->mpu_values.bi_semaphore);
87         ESP_LOGI(this->TAG, "Gave semaphore!");
88
89         vTaskDelay((this->sample_time)/portTICK_PERIOD_MS);
90     }
91 }
92
93 TaskHandle_t mpu_reader::get_task_handle(){
94     return this->mpu_reader_task_handle;
95 }
96
97 // QueueHandle_t mpu_reader::get_mpu_values_queue(){
98
99 //     return this->mpu_values.queue;
100 // }
101
102 void mpu_reader::mpu_reading_wrapper(void* arg){
103     mpu_reader* mpu_instance = static_cast<mpu_reader*>(arg);
104
105     mpu_instance->mpu_reading();
106 }
107
108 void mpu_reader::run(){
109
110     if(xTaskCreatePinnedToCore(mpu_reading_wrapper,
111                                     "mpu_reading",
112                                     4096,
113                                     this,
114                                     this->priority,
115                                     &mpu_reader_task_handle,
116                                     0) == pdPASS){
117         // Serial.println("Created mpu_reading task successfully");
118
119         ESP_LOGI(this->TAG, "Created mpu_reading task successfully\n");
120
121     }else{
```

```
122     // Serial.println("Created mpu_reading task failed!");
123
124     ESP_LOGE(this->TAG, "mpu_reading task creation failed\n");
125     while (true){}
126 }
127
128 }
129
130 SemaphoreHandle_t mpu_reader::get_semaphore_mpu(){
131     return this->mpu_values.bi_semaphore;
132 }
133
134 struct_mpu_reader mpu_reader::get_mpu_values(){
135     return mpu_values.values;
136 }
137
138 mpu_reader::mpu_reader(int8_t sample_time, uint8_t priority)
139 :   sample_time(sample_time),
140   priority(priority),
141   mpu_reader_task_handle(nullptr)
142 {
143     ESP_LOGI(this->TAG, "MPU6050 is initialising...\n");
144
145     this->mpu = new Adafruit_MPU6050();
146
147     while (!this->mpu->begin(0x68, &Wire, 0)) {
148         // Serial.println("Failed to find MPU6050 chip");
149         ESP_LOGE(this->TAG, "Failed to find MPU6050 chip\n");
150         delay(1000);
151     }
152
153     // Serial.println("MPU6050 Found!");
154     ESP_LOGI(this->TAG, "MPU6050 Found!\n");
155
156     this->sensors_setup();
157     // q_mpu_values = xQueueCreate(20, sizeof(struct_mpu_reader));
158
159     // this->mpu_values.queue = xQueueCreate(20, sizeof(struct_mpu_reader));
160
161     this->mpu_values.bi_semaphore = xSemaphoreCreateBinary();
162
163     // Serial.println("MPU READER, Initialized successfully");
164     ESP_LOGI(this->TAG, "Initialized sensors successfully\n");
165 }
166
```

```

167 mpu_reader::~mpu_reader()
168 {
169 }
170
171 mpu_reader* mpu_reader_component;

```

Program 6.2: mpu_reader.cpp

6.2 PID Block

6.2.1 PID_block.hpp

```

1 #pragma once
2 #ifndef PID_BLOCK_HPP
3 #define PID_BLOCK_HPP
4
5 #include <Arduino.h>
6 #include "center_controller.hpp"
7 #include "motor_controller.hpp"
8 #include "mpu_reader.hpp"
9 #include "global.hpp"
10
11 class PID_block
12 {
13 private:
14
15     const char* TAG = "PID BLOCK";
16     const int delay_time;
17
18     unsigned long prev_time;
19     const uint8_t priority;
20
21     const float alpha;
22
23     TaskHandle_t PID_task_handle;
24
25     struct{
26
27         QueueHandle_t queue;
28         struct_PID_block values;
29
30     } PID_values;
31
32     struct_PID_parameters PID_params;
33
34     void PID_compute();
35

```

```

36     static void PID_compute_wrapper(void* arg);
37
38 public:
39     PID_block(int delay_time = 10, uint8_t priority = 7, float alpha =
40               0.9934);
41     ~PID_block();
42
43     QueueHandle_t get_PID_values_queue();
44     TaskHandle_t get_task_handle();
45     void set_PID_parameters(struct_PID_parameters params);
46     void run();
47
48     void set_prev_time();
49
50 extern PID_block* PID_block_component;
51
52
53 #endif

```

Program 6.3: PID_block.hpp

6.2.2 PID_block.cpp

```

1 #include "PID_block.hpp"
2
3 PID_block::PID_block(int delay_time, uint8_t priority, float alpha)
4 :   delay_time(delay_time),
5   priority(priority),
6   alpha(alpha),
7   PID_task_handle(nullptr)
8 {
9
10    ESP_LOGI(this->TAG, "PID is initialising\n");
11
12    this->PID_params = {0,0,0,0};
13    this->PID_values = {
14
15        .queue = xQueueCreate(20, sizeof(struct_PID_block)),
16        .values = {0,0,0,0,0,0}
17
18    };
19
20    ESP_LOGI(this->TAG, "Initialized PID successfully\n");
21
22 }
23

```

```
24 PID_block::~PID_block()
25 {
26 }
27
28 void PID_block::PID_compute(){
29
30     struct_mpu_reader mpu_values;
31
32     // QueueHandle_t q_mpu_values = mpu_reader_component->
33     // get_mpu_values_queue();
34
35     SemaphoreHandle_t sem_mpu = mpu_reader_component->get_semaphore_mpu();
36
37     float currentAngle = 0;
38     float duration_time = 0;
39
40     float pitch = 0;
41     float error = 0;
42     float previousError = 0;
43     float integral = 0;
44     float derivative = 0;
45     float output = 0;
46
47     static int motorSpeed = 0;
48
49     vTaskSuspend(PID_task_handle);
50     this->prev_time = millis();
51
52     while (true)
53     {
54         xSemaphoreTake(sem_mpu, portMAX_DELAY);
55
56         if(duration_time < (this->delay_time/1000)) continue;
57
58         mpu_values = mpu_reader_component->get_mpu_values();
59
60         duration_time = (millis() - this->prev_time) * 0.001;
61
62         pitch = atan2(mpu_values.accel.z, mpu_values.accel.x);
63         currentAngle = this->alpha*(currentAngle + mpu_values.gyro.y *
64         duration_time) + (1 - this->alpha)*(pitch);
65
66         error = this->PID_params.setpoint - (currentAngle * RAD_TO_DEG);
67
68         integral += error * duration_time;
```

```
68         derivative = (error - previousError) / duration_time;
69
70         output = this->PID_params.Kp * error + this->PID_params.Ki *
71         integral + this->PID_params.Kd * derivative;
72
73         motorSpeed = constrain(output, -255, 255);
74
75         if (motorSpeed > 0) {
76
77             motor_controller_component->control_state(MOVE_FORWARD);
78             motor_controller_component->control_both_EN(motorSpeed);
79
80         } else {
81
82             motor_controller_component->control_state(MOVE_BACKWARD);
83             motor_controller_component->control_both_EN(abs(motorSpeed));
84
85         }
86
87         ESP_LOGI(this->TAG,
88                 "Current Angle = %.2f,\terror = %.2f,\tintegral = %.2f,\t
89         derivative = %.2f,\toutput = %.2f,\tduration_time = %.2f",
90                 currentAngle*RAD_TO_DEG,
91                 error,
92                 integral,
93                 derivative,
94                 output,
95                 duration_time);
96
97         this->PID_values.values = {
98
99             .filtered_pitch = currentAngle,
100            .gyro_angle_Y = mpu_values.gyro.y,
101            .output = output,
102            .error = error,
103            .integral = integral,
104            .derivative = derivative,
105
106        };
107
108        if(xQueueSend(this->PID_values.queue, &(this->PID_values.values),
109        this->delay_time/portTICK_PERIOD_MS) == pdTRUE){
110
111            ESP_LOGI(this->TAG, "Sent PID values successfully");
112
113        } else{
```

```
111
112         ESP_LOGE(this->TAG, "Queue is full");
113
114     }
115
116     previousError = error;
117
118     this->prev_time = millis();
119
120 }
121
122 }
123
124 void PID_block::PID_compute_wrapper(void* arg){
125
126     PID_block* PID_block_instance = static_cast<PID_block*>(arg);
127
128     PID_block_instance->PID_compute();
129 }
130
131 QueueHandle_t PID_block::get_PID_values_queue(){
132     return this->PID_values.queue;
133 }
134
135 TaskHandle_t PID_block::get_task_handle(){
136     return this->PID_task_handle;
137 }
138
139 void PID_block::set_PID_parameters(struct_PID_parameters params){
140     this->PID_params = params;
141 }
142
143 void PID_block::run(){
144
145     if( xTaskCreatePinnedToCore(PID_compute_wrapper,
146                                 "PID_task",
147                                 4096,
148                                 this,
149                                 this->priority,
150                                 &PID_task_handle,
151                                 0) == pdPASS){
152         // ESP_LOGI("PID_BLOCK", "Created task successfully!\n");
153         ESP_LOGI(this->TAG, "Created task successfully!");
154
155     }else{
156         // ESP_LOGE("PID_BLOCK", "Create PID_task() failed!\n");
157 }
```

```

157
158     ESP_LOGE(this->TAG, "Create PID_task failed!");
159
160     while (true){}
161
162 }
163 }
164
165 void PID_block::set_prev_time(){
166     this->prev_time = millis();
167 }
168
169 PID_block* PID_block_component;

```

Program 6.4: PID_block.cpp

6.3 Motor Controller

6.3.1 motor_controller.hpp

```

1 #pragma once
2 #ifndef MOTOR_CONTROLLER_HPP
3 #define MOTOR_CONTROLLER_HPP
4
5 #include <Arduino.h>
6
7 #define MOTOR_IN1_PIN      19
8 #define MOTOR_IN2_PIN      18
9 #define MOTOR_IN3_PIN      17
10 #define MOTOR_IN4_PIN     16
11
12 #define MOTOR_ENA_PIN     25
13 #define MOTOR_ENB_PIN     26
14
15 #define MOVE_FORWARD      0
16 #define MOVE_BACKWARD     1
17 #define MOVE_LEFT         2
18 #define MOVE_RIGHT        3
19 #define MOVE_STOP         4
20
21 #define MIN_MOVE_STATE_VALUE 0
22 #define MAX_MOVE_STATE_VALUE 4
23
24 class motor_controller
25 {
26 private:
27

```

```
28     const char* TAG = "MOTOR CONTROLLER";
29     const uint8_t motor_in_state[5][4];
30     const int delay_time;
31     const uint8_t priority;
32
33     TaskHandle_t motor_controller_task_handle;
34
35     struct{
36
37         uint8_t state;
38         SemaphoreHandle_t xMutex_state;
39
40     }current_movement_state;
41
42     struct{
43
44         int value;
45         SemaphoreHandle_t xMutex_value;
46
47     }ENA_value;
48
49     struct{
50
51         int value;
52         SemaphoreHandle_t xMutex_value;
53
54     }ENB_value;
55
56     void motor_work();
57
58     static void motor_work_wrapper(void* arg);
59
60 public:
61     motor_controller(int delay_time = 10, uint8_t priority = 7);
62     ~motor_controller();
63
64     void run(void);
65
66     TaskHandle_t get_task_hanlde();
67
68     void write_state(uint8_t state);
69     void write_ENA(int value);
70     void write_ENB(int value);
71     void write_both_EN(int value);
72
73     void control_state(uint8_t state);
```

```

74     void control_both_EN(int value);
75     void control_stop();
76
77     void stop();
78 };
79
80 extern motor_controller* motor_controller_component;
81
82 #endif

```

Program 6.5: motor_controller.hpp

6.3.2 motor_controller.cpp

```

1 #include "motor_controller.hpp"
2
3 #define IN1_SIGNAL_INDEX 0
4 #define IN2_SIGNAL_INDEX 1
5 #define IN3_SIGNAL_INDEX 2
6 #define IN4_SIGNAL_INDEX 3
7
8 #define ENABLE_MIN_VALUE 0
9 #define ENABLE_MAX_VALUE 255
10
11 #define MOTOR_MAX_STATE 4
12 #define MOTOR_MIN_STATE 0
13
14 motor_controller::motor_controller(int delay_time, uint8_t priority)
15     : motor_in_state{
16         {1, 0, 0, 1},
17         {0, 1, 1, 0},
18         {0, 1, 0, 1},
19         {1, 0, 1, 0},
20         {0, 0, 0, 0}
21     },
22     delay_time(delay_time),
23     priority(priority),
24     motor_controller_task_handle(nullptr)
25 {
26     ESP_LOGI(this->TAG, "PID is initialising\n");
27
28     pinMode(MOTOR_IN1_PIN, OUTPUT);
29     pinMode(MOTOR_IN2_PIN, OUTPUT);
30     pinMode(MOTOR_IN3_PIN, OUTPUT);
31     pinMode(MOTOR_IN4_PIN, OUTPUT);
32
33     pinMode(MOTOR_ENA_PIN, OUTPUT);

```

```
34     pinMode(MOTOR_ENB_PIN, OUTPUT);  
35  
36     this->current_movement_state = {  
37         .state = MOVE_STOP,  
38         .xMutex_state = xSemaphoreCreateMutex()  
39     };  
40  
41     this->ENA_value = {  
42         .value = 0,  
43         .xMutex_value = xSemaphoreCreateMutex()  
44     };  
45  
46     this->ENB_value = {  
47         .value = 0,  
48         .xMutex_value = xSemaphoreCreateMutex()  
49     };  
50  
51  
52     this->write_state(MOVE_STOP);  
53     this->write_both_EN(0);  
54  
55     ESP_LOGI(this->TAG, "Initialized motors successfully!");  
56  
57 }  
58  
59 motor_controller::~motor_controller()  
60 {  
61 }  
62  
63 void motor_controller::motor_work(){  
64  
65     vTaskSuspend(this->motor_controller_task_handle);  
66  
67     while (true)  
68     {  
69         // motor_write_both_EN(80);  
70  
71         if(xSemaphoreTake(this->current_movement_state.xMutex_state,  
72             portMAX_DELAY) == pdTRUE){  
73             // current_movement_state++;  
74             // if(current_movement_state > 4) current_movement_state = 0;  
75  
76             // Serial.printf("current_movement_state = %d\n",this->  
77             current_movement_state);  
78             ESP_LOGI(this->TAG,  
79                     "current_movement_state = %d\n",  
80                     current_movement_state);  
81         }  
82     }  
83 }
```

```
78         this->current_movement_state.state);  
79  
80         digitalWrite(MOTOR_IN1_PIN ,  
81                         this->motor_in_state[this->current_movement_state.  
state][IN1_SIGNAL_INDEX]);  
82  
83         digitalWrite(MOTOR_IN2_PIN ,  
84                         this->motor_in_state[this->current_movement_state.  
state][IN2_SIGNAL_INDEX]);  
85  
86         digitalWrite(MOTOR_IN3_PIN ,  
87                         this->motor_in_state[this->current_movement_state.  
state][IN3_SIGNAL_INDEX]);  
88  
89         digitalWrite(MOTOR_IN4_PIN ,  
90                         this->motor_in_state[this->current_movement_state.  
state][IN4_SIGNAL_INDEX]);  
91  
92         xSemaphoreGive(this->current_movement_state.xMutex_state);  
93  
94     }  
95  
96     if(xSemaphoreTake(this->ENA_value.xMutex_value , portMAX_DELAY) ==  
97         pdTRUE){  
98  
99         analogWrite(MOTOR_ENA_PIN , this->ENA_value.value);  
100  
101         xSemaphoreGive(this->ENA_value.xMutex_value);  
102     }  
103  
104     if(xSemaphoreTake(this->ENB_value.xMutex_value , portMAX_DELAY) ==  
105         pdTRUE){  
106  
107         analogWrite(MOTOR_ENB_PIN , this->ENB_value.value);  
108     }  
109     vTaskDelay(this->delay_time/portTICK_PERIOD_MS);  
110 // vTaskDelay((3000)/portTICK_PERIOD_MS);  
111  
112 // Serial.printf("\n");  
113 }  
114  
115 }  
116  
117 void motor_controller::motor_work_wrapper(void* arg){
```

```
118     motor_controller* motor_controller_instance = static_cast<
119     motor_controller*>(arg);
120
121 }
122
123 void motor_controller::run(void){
124
125     if( xTaskCreatePinnedToCore(motor_work_wrapper,
126                                 "motor_work",
127                                 2048,
128                                 this,
129                                 this->priority,
130                                 &motor_controller_task_handle,
131                                 0)==pdPASS){
132
133         ESP_LOGI(this->TAG, "Created motor_work task successfully\n");
134
135
136         // ESP_LOGI(this->TAG, "Task state = %d\n", eTaskGetState(this->
137         motor_controller_task_handle));
138
139     }else{
140
141         // Serial.println("Create motor_controller_task() task failed!");
142         ESP_LOGE(this->TAG, "Create motor_controller_task task failed!\n")
143         ;
144
145     }
146
147 TaskHandle_t motor_controller::get_task_hanlde(){
148
149     return this->motor_controller_task_handle;
150
151 }
152
153 void motor_controller::write_state(uint8_t state){
154
155     if(xSemaphoreTake(this->current_movement_state.xMutex_state,
156                       portMAX_DELAY) == pdTRUE){
157
158         if(state < MOTOR_MIN_STATE){
159
160             this->current_movement_state.state = MOTOR_MIN_STATE;
161
162         }else if (state > MOTOR_MAX_STATE){
163
164             this->current_movement_state.state = MOTOR_MAX_STATE;
165
166         }
167
168     }
169
170 }
```

```
160
161         this->current_movement_state.state = MOTOR_MAX_STATE;
162
163     } else{
164         this->current_movement_state.state = state;
165     }
166
167     xSemaphoreGive(this->current_movement_state.xMutex_state);
168
169     ESP_LOGI(this->TAG, "Set moving state successfully\n");
170
171 } else{
172     ESP_LOGE(this->TAG, "Can not set moving state\n");
173 }
174
175 }
176
177 void motor_controller::write_ENA(int value){
178
179     value += 50;
180
181     if(xSemaphoreTake(this->ENA_value.xMutex_value, portMAX_DELAY) == pdTRUE){
182
183         if(value < ENABLE_MIN_VALUE){
184
185             ENA_value.value = ENABLE_MIN_VALUE;
186         } else if (value > ENABLE_MAX_VALUE)
187         {
188             ENA_value.value = ENABLE_MAX_VALUE;
189         } else{
190             ENA_value.value = value;
191         }
192         ESP_LOGI(this->TAG, "Set ENA successfully\n");
193         xSemaphoreGive(this->ENA_value.xMutex_value);
194     } else{
195
196         ESP_LOGE(this->TAG, "Can't set PWM for ENA!\n");
197
198     }
199
200 }
201
202 void motor_controller::write_ENB(int value){
203
204     if(xSemaphoreTake(this->ENB_value.xMutex_value, portMAX_DELAY) ==
```

```
pdTRUE){  
205  
206     if(value < ENABLE_MIN_VALUE){  
207  
208         this->ENB_value.value = ENABLE_MIN_VALUE;  
209  
210     }else if (value > ENABLE_MAX_VALUE)  
211     {  
212         this->ENB_value.value = ENABLE_MAX_VALUE;  
213     }  
214     }else{  
215  
216         this->ENB_value.value = value;  
217     }  
218  
219     ESP_LOGI(this->TAG , "Set ENB successfully\n");  
220  
221     xSemaphoreGive(this->ENB_value.xMutex_value);  
222 }else{  
223  
224     ESP_LOGE(this->TAG , "Can't set PWM for ENB!\n");  
225  
226 }  
227  
228 }  
229  
230 void motor_controller::write_both_EN(int value){  
231  
232     this->write_ENA(value);  
233     this->write_ENB(value);  
234 }  
235  
236 void motor_controller::stop(){  
237  
238     if(xSemaphoreTake(this->current_movement_state.xMutex_state ,  
portMAX_DELAY) == pdTRUE){  
239  
240         this->current_movement_state.state = MOVE_STOP;  
241  
242         digitalWrite(MOTOR_IN1_PIN ,  
243                         this->motor_in_state[this->current_movement_state.  
state][IN1_SIGNAL_INDEX]);  
244  
245         digitalWrite(MOTOR_IN2_PIN ,  
246                         this->motor_in_state[this->current_movement_state.  
state][IN2_SIGNAL_INDEX]);  
247 }
```

```
247     digitalWrite(MOTOR_IN3_PIN ,
248                 this->motor_in_state[this->current_movement_state .
249 state][IN3_SIGNAL_INDEX]);
250
251     digitalWrite(MOTOR_IN4_PIN ,
252                 this->motor_in_state[this->current_movement_state .
253 state][IN4_SIGNAL_INDEX]);
254
255     xSemaphoreGive(this->current_movement_state.xMutex_state);
256 }
257
258 if(xSemaphoreTake(this->ENA_value.xMutex_value , portMAX_DELAY) ==
259 pdTRUE){
260
261     this->ENA_value.value = 0;
262
263     analogWrite(MOTOR_ENA_PIN , this->ENA_value.value);
264
265     xSemaphoreGive(this->ENA_value.xMutex_value);
266 }
267
268 if(xSemaphoreTake(this->ENB_value.xMutex_value , portMAX_DELAY) ==
269 pdTRUE){
270
271     this->ENB_value.value = 0;
272
273     analogWrite(MOTOR_ENB_PIN , this->ENB_value.value);
274
275 }
276
277 void motor_controller::control_state(uint8_t state){
278
279     this->current_movement_state.state = state;
280
281     if(state < MIN_MOVE_STATE_VALUE) {
282
283         ESP_LOGI(this->TAG , "Input state is less than MIN_MOVE_STATE_VALUE
284 ");
285
286         this->current_movement_state.state = MIN_MOVE_STATE_VALUE;
287     }
288 }
```

```
288     if(state > MAX_MOVE_STATE_VALUE) {
289
290         ESP_LOGI(this->TAG, "Input state is greater than
291         MAX_MOVE_STATE_VALUE");
292
293     }
294
295     ESP_LOGI(this->TAG,
296             "current_movement_state = %d\n",
297             this->current_movement_state.state);
298
299     // digitalWrite(MOTOR_IN1_PIN,
300     //               this->motor_in_state[this->current_movement_state.state]
301     //               [IN1_SIGNAL_INDEX]);
302
303     // digitalWrite(MOTOR_IN2_PIN,
304     //               this->motor_in_state[this->current_movement_state.state]
305     //               [IN2_SIGNAL_INDEX]);
306
307     digitalWrite(MOTOR_IN3_PIN,
308                 this->motor_in_state[this->current_movement_state.state][
309                 IN3_SIGNAL_INDEX]);
310
311
312 }
313
314 void motor_controller::control_both_EN(int value){
315
316     // this->ENA_value.value = value;
317     this->ENB_value.value = value;
318
319     // ESP_LOGI(this->TAG, "value = %d,\tENA_value = %d,\tENB_value = %d",
320     //           value,
321     //           this->ENA_value.value,
322     //           this->ENB_value.value);
323
324     // ESP_LOGI(this->TAG, "value = %d,\tENB_value = %d",
325     //           value,
326     //           this->ENB_value.value);
327
328     // analogWrite(MOTOR_ENA_PIN, this->ENA_value.value);
```

```

329
330     analogWrite(MOTOR_ENB_PIN, this->ENB_value.value);
331
332 }
333
334 void motor_controller::control_stop(){
335
336
337     this->current_movement_state.state = MOVE_STOP;
338
339     // digitalWrite(MOTOR_IN1_PIN,
340     //               this->motor_in_state[this->current_movement_state.state]
341     //               [IN1_SIGNAL_INDEX]);
342
343     // digitalWrite(MOTOR_IN2_PIN,
344     //               this->motor_in_state[this->current_movement_state.state]
345     //               [IN2_SIGNAL_INDEX]);
346
347     digitalWrite(MOTOR_IN3_PIN,
348                 this->motor_in_state[this->current_movement_state.state][
349                 IN3_SIGNAL_INDEX]);
350
351     digitalWrite(MOTOR_IN4_PIN,
352                 this->motor_in_state[this->current_movement_state.state][
353                 IN4_SIGNAL_INDEX]);
354
355     // this->ENA_value.value = 0;
356
357     // analogWrite(MOTOR_ENA_PIN, this->ENA_value.value);
358
359     this->ENB_value.value = 0;
360
361
362
363
364 motor_controller* motor_controller_component;

```

Program 6.6: motor_controller.cpp

6.4 Center Controller

6.4.1 center_controller.hpp

```
1 #pragma once
2 #ifndef CENTER_CONTROLLER_HPP
3 #define CENTER_CONTROLLER_HPP
4
5 #include <Arduino.h>
6 #include "EEPROM.h"
7 #include "mpu_reader.hpp"
8 #include "motor_controller.hpp"
9 #include "PID_block.hpp"
10 #include "display_controller.hpp"
11 #include "global.hpp"
12
13 class center_controller
14 {
15 private:
16
17     const char* TAG = "CENTER CONTROLLER";
18     const int delay_time;
19
20     struct{
21
22         bool flag;
23         SemaphoreHandle_t xMutex_flag;
24
25     } start_robot_flag;
26
27     struct{
28
29         struct_PID_parameters params;
30         SemaphoreHandle_t xMutex_PID_parameters;
31
32     } PID_params;
33
34
35     struct_eeprom_address eeprom_adresses;
36
37     // QueueHandle_t q_angle_values;
38
39     // struct_angle_values angle_values;
40
41     void system_controller();
42
43     static void system_controller_wrapper(void* arg);
44
45
46 public:
```

```

47     center_controller(int delay_time = 200);
48     ~center_controller();
49
50     void set_PID_parameters(struct_PID_parameters params);
51
52     struct_PID_parameters get_PID_parameters();
53
54     SemaphoreHandle_t get_mutex_PID_params_handle();
55
56     struct_eeprom_address get_eeprom_adresses();
57
58     void set_start_robot_flag(bool value);
59
60     void run();
61 };
62
63 extern center_controller* center_controller_component;
64
65 #endif

```

Program 6.7: center_controller.hpp

6.4.2 center_controller.cpp

```

1 #include "center_controller.hpp"
2
3 center_controller::center_controller(int delay_time)
4 : delay_time(delay_time)
5 {
6     ESP_LOGI(this->TAG, "Center controller is initialising\n");
7
8     EEPROM.begin(sizeof(struct_PID_parameters));
9
10    this->start_robot_flag = {
11        .flag = false,
12        .xMutex_flag = xSemaphoreCreateMutex()
13    };
14
15    this->eeprom_adresses = {
16
17        .eeprom_Kp_address      = 0,
18        .eeprom_Ki_address      = sizeof(float),
19        .eeprom_Kd_address      = 2*sizeof(float),
20        .eeprom_setpoint_address = 3*sizeof(float)
21
22    };
23

```

```

24     this->PID_params = {
25
26         .params = {
27
28             .Kp      = EEPROM.readFloat(eeprom_adresses.eeprom_Kp_address
29             ),
30             .Ki      = EEPROM.readFloat(eeprom_adresses.eeprom_Ki_address
31             ),
32             .Kd      = EEPROM.readFloat(eeprom_adresses.eeprom_Kd_address
33             ),
34             .setpoint = EEPROM.readFloat(eeprom_adresses.
35                 eeprom_setpoint_address)
36         },
37
38         .xMutex_PID_parameters = xSemaphoreCreateMutex()
39     };
40
41
42     ESP_LOGI(this->TAG,
43             "eeprom's size = %d\n",
44             EEPROM.length());
45
46     ESP_LOGI(this->TAG,
47             "Kp = %.2f at address = %d\n",
48             this->PID_params.params.Kp,
49             eeprom_adresses.eeprom_Kp_address);
50
51
52     ESP_LOGI(this->TAG,
53             "Ki = %.2f at address = %d\n",
54             this->PID_params.params.Ki,
55             eeprom_adresses.eeprom_Ki_address);
56
57     ESP_LOGI(this->TAG,
58             "Kd = %.2f at address = %d\n",
59             this->PID_params.params.Kd,
60             eeprom_adresses.eeprom_Kd_address);
61
62     ESP_LOGI(this->TAG,
63             "Setpoint = %.2f at address = %d\n",
64             this->PID_params.params.setpoint,
65             eeprom_adresses.eeprom_setpoint_address);
66
67     ESP_LOGI(this->TAG, "Initialized Center controller successfully\n");
68 }
69
70 center_controller::~center_controller()

```

```
66 {
67 }
68
69 void center_controller::set_PID_parameters(struct_PID_parameters params){
70
71     if(xSemaphoreTake(this->PID_params.xMutex_PID_parameters,
72                      portMAX_DELAY) == pdTRUE){
73
74         this->PID_params.params = params;
75         xSemaphoreGive(this->PID_params.xMutex_PID_parameters);
76     }
77 }
78
79 struct_PID_parameters center_controller::get_PID_parameters(){
80     struct_PID_parameters temp = {0,0,0,0};
81
82     if(xSemaphoreTake(this->PID_params.xMutex_PID_parameters,
83                      portMAX_DELAY) == pdTRUE){
84
85         temp = this->PID_params.params;
86         xSemaphoreGive(this->PID_params.xMutex_PID_parameters);
87     }
88
89     return temp;
90 }
91 SemaphoreHandle_t center_controller::get_mutex_PID_params_handle(){
92     return this->PID_params.xMutex_PID_parameters;
93 }
94
95 void center_controller::set_start_robot_flag(bool value){
96
97     if(xSemaphoreTake(this->start_robot_flag.xMutex_flag, portMAX_DELAY)
98        == pdTRUE){
99
100         this->start_robot_flag.flag = value;
101         xSemaphoreGive(this->start_robot_flag.xMutex_flag);
102     }
103 }
104
105 struct_eeprom_address center_controller::get_eeprom_adresses(){
106     return this->eeprom_adresses;
107 }
108
```

```
109 void center_controller::system_controller(){
110
111     eTaskState mpu_task_state, motor_task_state, pid_task_state;
112
113     while (true){
114
115         vTaskDelay(this->delay_time/portTICK_PERIOD_MS);
116
117         if(xSemaphoreTake(this->start_robot_flag.xMutex_flag,
118                           portMAX_DELAY) == pdFALSE){
119             ESP_LOGE(this->TAG, "Can't get xMutex_start_robot_flag\n");
120             continue;
121         }
122
123         mpu_task_state = eTaskGetState(mpu_reader_component->
124                                         get_task_handle());
125
126         pid_task_state = eTaskGetState(PID_block_component->
127                                       get_task_handle());
128
129         if(this->start_robot_flag.flag){
130
131             if(mpu_task_state == eSuspended){
132
133                 vTaskResume(mpu_reader_component->get_task_handle());
134
135                 mpu_task_state = eTaskGetState(mpu_reader_component->
136                                               get_task_handle());
137
138                 if (mpu_task_state == eRunning)
139                 {
140                     ESP_LOGI(this->TAG, "MPU reader's task resumed
141 successfully\n");
142                 }
143
144             }
145
146             if(pid_task_state == eSuspended){
147
148                 PID_block_component->set_PID_parameters(this->PID_params.
149
150                 params);
151
152                 PID_block_component->set_prev_time();
153
154                 vTaskResume(PID_block_component->get_task_handle());
155
156                 pid_task_state = eTaskGetState(PID_block_component->
```

```
    get_task_handle());  
149  
150        if (pid_task_state == eRunning)  
151        {  
152            ESP_LOGI(this->TAG, "PID block's task resumed  
153 successfully\\n");  
154        }  
155    }  
156  
157    xSemaphoreGive(this->start_robot_flag.xMutex_flag);  
158  
159    continue;  
160}  
161  
162  
163    if (mpu_task_state == eRunning || mpu_task_state == eBlocked)  
164    {  
165        vTaskSuspend(mpu_reader_component->get_task_handle());  
166  
167        ESP_LOGI(this->TAG, "Suspended mpu_reader task!");  
168    }  
169  
170    if (pid_task_state == eRunning || pid_task_state == eBlocked)  
171    {  
172        vTaskSuspend(PID_block_component->get_task_handle());  
173        motor_controller_component->control_stop();  
174        ESP_LOGI(this->TAG, "Suspended pid_block task!");  
175    }  
176  
177    xSemaphoreGive(this->start_robot_flag.xMutex_flag);  
178  
179 }  
180}  
181  
182 void center_controller::system_controller_wrapper(void* arg){  
183  
184     center_controller* center_controller_instance = static_cast<  
185     center_controller*>(arg);  
186  
187     center_controller_instance->system_controller();  
188 }  
189  
190 void center_controller::run(){  
191     mpu_reader_component->run();
```

```

192
193     PID_block_component->run();
194
195     if(xTaskCreatePinnedToCore(system_controller_wrapper,"
196         system_controller", 2048, this, 9, nullptr, 1)== pdPASS){
197         ESP_LOGI(this->TAG, "Created system_controller task successfully")
198         ;
199     }else{
200         ESP_LOGE(this->TAG, "Created system_controller task failed!");
201         while (true){}
202     }
203
204
205 }
206
207 center_controller* center_controller_component;

```

Program 6.8: center_controller.cpp

6.5 Button Reader

6.5.1 button_reader.hpp

```

1 #pragma once
2 #ifndef BUTTON_READER_HPP
3 #define BUTTON_READER_HPP
4
5 #include <Arduino.h>
6
7 #define DEBOUNCE_THRESHOLD 220
8
9 #define BTN_OK      13
10 #define BTN_ESC    27
11 #define BTN_DOWN   33
12 #define BTN_UP    32
13
14 struct Button {
15     const uint8_t PIN;
16     // uint32_t numberKeyPresses;
17     volatile bool buttonPressed;
18     volatile unsigned long lastDebounceTime;
19     volatile int counter;
20 };
21
22 void init_ctrl_btn(Button btn);

```

```

23
24 void IRAM_ATTR handleButtonPress(void* arg);
25
26 void attach_btn_interrupt(Button* btn);
27
28 void button_init(void);
29
30 extern Button btn_OK;
31 extern Button btn_ESC;
32 extern Button btn_UP;
33 extern Button btn_DOWN;
34
35
36 #endif

```

Program 6.9: button_reader.hpp

6.5.2 button_reader.cpp

```

1 #include "button_reader.hpp"
2
3 Button btn_OK = {BTN_OK, false, 0, 0};
4 Button btn_ESC = {BTN_ESC, false, 0, 0};
5 Button btn_UP = {BTN_UP, false, 0, 0};
6 Button btn_DOWN = {BTN_DOWN, false, 0, 0};
7
8
9 void init_ctrl_btn(Button btn){
10
11     pinMode(btn.PIN, INPUT_PULLUP);
12 }
13
14 portMUX_TYPE mux = portMUX_INITIALIZER_UNLOCKED;
15
16 void IRAM_ATTR handleButtonPress(void* arg) {
17     portENTER_CRITICAL_ISR(&mux);
18
19     unsigned long currentTime = millis();
20
21     Button* temp = static_cast<Button*>(arg);
22
23     if(currentTime != temp->lastDebounceTime){
24
25         if ((currentTime - temp->lastDebounceTime) > DEBOUNCE_THRESHOLD) {
26             temp->buttonPressed = true;
27             temp->lastDebounceTime = currentTime;
28         }

```

```

29     }
30
31     portEXIT_CRITICAL_ISR(&mux);
32 }
33
34 void attach_btn_interrupt(Button* btn){
35
36     attachInterruptArg(btn->PIN, handleButtonPress, btn, FALLING);
37 }
38
39 void button_init(void){
40     // Initialize buttons and attach them to Interrupt
41     init_ctrl_btn(btn_OK);
42     init_ctrl_btn(btn_ESC);
43     init_ctrl_btn(btn_UP);
44     init_ctrl_btn(btn_DOWN);
45
46     attach_btn_interrupt(&btn_OK);
47     attach_btn_interrupt(&btn_ESC);
48     attach_btn_interrupt(&btn_UP);
49     attach_btn_interrupt(&btn_DOWN);
50 }
```

Program 6.10: button_reader.cpp

6.6 Display Controller

6.6.1 display_controller.hpp

```

1 #pragma once
2 #ifndef OLED_DISPLAY_HPP
3 #define OLED_DISPLAY_HPP
4
5 #include <SPI.h>
6 #include <Wire.h>
7 #include <Adafruit_GFX.h>
8 #include <Adafruit_SSD1306.h>
9 #include <Arduino.h>
10
11 #include "button_reader.hpp"
12 #include "global.hpp"
13 #include "UI_tree.hpp"
14 #include "center_controller.hpp"
15 #include "PID_block.hpp"
16
17 #define SCREEN_WIDTH 128
18 #define SCREEN_HEIGHT 64
```

```
19
20 #define OLED_RESET      -1 // Reset pin # (or -1 if sharing Arduino reset
21   pin)
22
23 class display_controller
24 {
25   private:
26     const char* TAG = "DISPLAY CONTROLLER";
27     const uint8_t fps;
28     const uint8_t priority; // draw_menu task's priority
29     const uint8_t scr_width;
30     const uint8_t scr_height;
31
32     Adafruit_SSD1306* display;
33     UI_tree* current_UI;
34     SemaphoreHandle_t xMutex_menu_curr_opt;
35     struct_PID_parameters temp_PID_params;
36
37     void create_all_ui(void);
38
39     void draw_menu();
40     static void draw_menu_wrapper(void* arg);
41
42     void get_btn_OK();
43     static void get_btn_OK_wrapper(void* arg);
44
45     void get_btn_ESC();
46     static void get_btn_ESC_wrapper(void* arg);
47
48     void get_btn_UP();
49     static void get_btn_UP_wrapper(void* arg);
50
51     void get_btn_DOWN();
52     static void get_btn_DOWN_wrapper(void* arg);
53
54 public:
55   display_controller( uint8_t fps = 30,
56                      uint8_t priority = 5,
57                      uint8_t width = SCREEN_WIDTH,
58                      uint8_t height = SCREEN_HEIGHT);
59   ~display_controller();
60
61   Adafruit_SSD1306* get_display();
62
63   UI_tree* get_current_ui();
64
65   uint8_t get_fps();
```

```

64
65     struct_PID_parameters get_temp_PID_params();
66
67     void set_temp_PID_params(struct_PID_parameters new_value);
68
69     void set_current_ui(UI_tree* ui);
70
71     void run();
72 };
73
74 extern display_controller* display_controller_component;
75
76 #endif

```

Program 6.11: display_controller.hpp

6.6.2 display_controller.cpp

```

1 #include "display_controller.hpp"
2
3 #pragma region ALL DEFINITION FOR MENU UI
4
5 #define MENU_START_N_PLOT_OPTION 0
6 #define MENU_PID_VALUES_OPTION 1
7
8 #define MENU_MAX_OPTION 2
9 #define MENU_MIN_OPTION 1
10
11 #pragma endregion
12
13 #pragma region ALL DEFINITION FOR PARAMETERS OF PID UI
14
15 #define PID_PARAMETERS MODIFY_KP 0
16 #define PID_PARAMETERS MODIFY_KI 1
17 #define PID_PARAMETERS MODIFY_KD 2
18 #define PID_PARAMETERS MODIFY_SETPOINT 3
19
20 #define PID_PARAMETERS_MAX_OPTION 4
21 #define PID_PARAMETERS_MIN_OPTION 1
22
23 #pragma endregion
24
25 #pragma region ALL ALL DEFINITION FOR MODIFY UIS
26
27 #define MODIFY_MIN_PARAMETERS_VALUE 0
28 #define MODIFY_MAX_PARAMETERS_VALUE 20
29

```

```
30 #pragma endregion
31
32 #pragma region UN-NAME STRUCT FOR ALL UIS
33 struct{
34     String title = "MENU";
35     uint8_t args[1] = {0};
36     uint8_t args_len = 1;
37     uint8_t nextUI_len = 2;
38     void display_func(){
39
40         if (display_controller_component->get_current_ui()->args[0] ==
41             MENU_START_N_PLOT_OPTION)
42         {
43             display_controller_component->get_display()->setTextColor(BLACK,
44             WHITE);
45             }else{
46                 display_controller_component->get_display()->setTextColor(WHITE,
47                 BLACK);
48             }
49             display_controller_component->get_display()->println(F("Start and
50             Plot"));
51
52         if (display_controller_component->get_current_ui()->args[0] ==
53             MENU_PID_VALUES_OPTION)
54         {
55             display_controller_component->get_display()->setTextColor(BLACK,
56             WHITE);
57             }else{
58                 display_controller_component->get_display()->setTextColor(WHITE,
59                 BLACK);
60             }
61             display_controller_component->get_display()->println(F("PID's values
62             "));
63         }
64
65         void btnUP_func(){
66
67             ESP_LOGI(title, "In menu_ui_btnUP_func, display_controller_component
68             ->get_current_ui()->args[0] = %d\n", display_controller_component->
69             get_current_ui()->args[0]);
70             if(display_controller_component->get_current_ui()->args[0] <=
71             MENU_MIN_OPTION - 1){
72                 display_controller_component->get_current_ui()->args[0] =
73                 MENU_MAX_OPTION - 1;
74             }else{
75                 --display_controller_component->get_current_ui()->args[0];
76             }
77         }
```

```
64     }
65
66     }
67
68     void btnDOWN_func(){
69
70         if(display_controller_component->get_current_ui()->args[0] >=
71             MENU_MAX_OPTION - 1) {
72             display_controller_component->get_current_ui()->args[0] =
73             MENU_MIN_OPTION - 1;
74         }else{
75             ++display_controller_component->get_current_ui()->args[0];
76         }
77
78     void btnOK_func(){
79
80         if(display_controller_component->get_current_ui()->next_UI[
81             display_controller_component->get_current_ui()->args[0]] != nullptr){
82             // Serial.println("display_controller_component->get_current_ui()-
83             //>next_UI[display_controller_component->get_current_ui()->args[0]] !=
84             //nullptr");
85             if(args[0] == MENU_START_N_PLOT_OPTION){
86
87                 center_controller_component->set_start_robot_flag(true);
88
89             }
90
91             UI_tree* next_ui = display_controller_component->get_current_ui()
92             ->next_UI[display_controller_component->get_current_ui()->args[0]];
93
94             display_controller_component->set_current_ui(next_ui);
95         }else{
96             // Serial.println("display_controller_component->get_current_ui()-
97             //>next_UI[display_controller_component->get_current_ui()->args[0]] ==
98             //nullptr");
99         }
100     }
```

```

101     UI_tree* previous_ui = display_controller_component->
102     get_current_ui()->prev_UI;
103
104     display_controller_component->set_current_ui(previous_ui);
105 }
106 }
107 } menu_ui;
108
109 struct
110 {
111     String title = "START AND PLOT";
112     uint8_t args_len = 0;
113     uint8_t nextUI_len = 0;
114
115     void display_func(){
116
117         static struct_PID_block PID_values;
118
119         if(xQueueReceive(PID_block_component->get_PID_values_queue(), &
120 PID_values, display_controller_component->get_fps()/portTICK_PERIOD_MS)
121 == pdTRUE){
122
123             display_controller_component->get_display()->print("Pitch: ");
124             display_controller_component->get_display()->print(PID_values.
125 filtered_pitch*RAD_TO_DEG);
126             display_controller_component->get_display()->println(" deg");
127             // display_controller_component->get_display()->print("Gyro Y: ");
128             // display_controller_component->get_display()->print(PID_values.
129             gyro_angle_Y);
130             // display_controller_component->get_display()->println(" deg/s");
131             display_controller_component->get_display()->print("Error: ");
132             display_controller_component->get_display()->println(PID_values.
133 error);
134             display_controller_component->get_display()->print("Integral: ");
135             display_controller_component->get_display()->println(PID_values.
136 integral);
137             display_controller_component->get_display()->print("Derivative: ")
138 ;
139             display_controller_component->get_display()->println(PID_values.
140 derivative);
141             display_controller_component->get_display()->print("Output: ");
142             display_controller_component->get_display()->println(PID_values.
143 output);
144
145     }else{

```

```

137         ESP_LOGE("DISPLAY CONTROLLER", "Can't receive angle values!\n");
138         display_controller_component->get_display()->print("Pitch: ");
139         display_controller_component->get_display()->print(PID_values.
140             filtered_pitch*RAD_TO_DEG);
141         display_controller_component->get_display()->println(" deg");
142         // display_controller_component->get_display()->print("Gyro Y: ");
143         // display_controller_component->get_display()->print(PID_values.
144         gyro_angle_Y);
145         // display_controller_component->get_display()->println(" deg/s");
146         display_controller_component->get_display()->print("Error: ");
147         display_controller_component->get_display()->println(PID_values.
148             error);
149         display_controller_component->get_display()->print("Integral: ");
150         display_controller_component->get_display()->println(PID_values.
151             integral);
152         display_controller_component->get_display()->print("Derivative: ");
153     }
154 }
155
156 void btnESC_func(){
157
158     if(display_controller_component->get_current_ui()->prev_UI !=
159         nullptr){
160
161         center_controller_component->set_start_robot_flag(false);
162
163         UI_tree* previous_ui = display_controller_component->
164             get_current_ui()->prev_UI;
165
166         display_controller_component->set_current_ui(previous_ui);
167     }
168
169 }
170 } start_n_plot_ui;
171
172 struct{
173

```

```
174     String title = "PID'S PARAMETERS";
175     uint8_t args[1] = {0};
176     uint8_t args_len = 1;
177     uint8_t nextUI_len = 4;
178
179     void display_func(){
180
181         if (display_controller_component->get_current_ui()->args[0] ==
182             PID_PARAMETERS MODIFY_KP)
183         {
184             display_controller_component->get_display()->setTextColor(BLACK,
185             WHITE);
186         } else{
187             display_controller_component->get_display()->setTextColor(WHITE,
188             BLACK);
189         }
190
191         display_controller_component->get_display()->printf("Kp = %.2f\n",
192             center_controller_component->get_PID_parameters().Kp);
193
194         if (display_controller_component->get_current_ui()->args[0] ==
195             PID_PARAMETERS MODIFY_KI)
196         {
197             display_controller_component->get_display()->setTextColor(BLACK,
198             WHITE);
199         } else{
200             display_controller_component->get_display()->setTextColor(WHITE,
201             BLACK);
202         }
203
204         display_controller_component->get_display()->printf("Ki = %.2f\n",
205             center_controller_component->get_PID_parameters().Ki);
206
207         if (display_controller_component->get_current_ui()->args[0] ==
208             PID_PARAMETERS MODIFY_KD)
209         {
210             display_controller_component->get_display()->setTextColor(BLACK,
211             WHITE);
212         } else{
213             display_controller_component->get_display()->setTextColor(WHITE,
214             BLACK);
215         }
216
217         display_controller_component->get_display()->printf("Kd = %.2f\n",
218             center_controller_component->get_PID_parameters().Kd);
219
220 }
```

```
208     if (display_controller_component->get_current_ui()->args[0] ==  
209         PID_PARAMETERS MODIFY_SETPOINT)  
210     {  
211         display_controller_component->get_display()->setTextColor(BLACK,  
212             WHITE);  
213     } else{  
214         display_controller_component->get_display()->setTextColor(WHITE,  
215             BLACK);  
216     }  
217  
218     display_controller_component->get_display()->printf("Setpoint = %.2f  
219     \n", center_controller_component->get_PID_parameters().setpoint);  
220  
221     }  
222  
223     void btnUP_func(){  
224         Serial.printf("In menu_ui_btnUP_func, display_controller_component->  
225             get_current_ui()->args[0] = %d\n", display_controller_component->  
226             get_current_ui()->args[0]);  
227         if(display_controller_component->get_current_ui()->args[0] <=  
228             PID_PARAMETERS_MIN_OPTION - 1) {  
229             display_controller_component->get_current_ui()->args[0] =  
230             PID_PARAMETERS_MAX_OPTION - 1;  
231         } else{  
232             --display_controller_component->get_current_ui()->args[0];  
233         }  
234     }  
235  
236     void btnDOWN_func(){  
237         if(display_controller_component->get_current_ui()->args[0] >=  
238             PID_PARAMETERS_MAX_OPTION - 1) {  
239             display_controller_component->get_current_ui()->args[0] =  
240             PID_PARAMETERS_MIN_OPTION - 1;  
241         } else{  
242             ++display_controller_component->get_current_ui()->args[0];  
243         }  
244     }  
245  
246     void btnOK_func(){  
247 }
```

```

244     if(display_controller_component->get_current_ui()->next_UI[  

245         display_controller_component->get_current_ui()->args[0]] != nullptr){  

246         Serial.printf("display_controller_component->get_current_ui()->  

247             next_UI[%d] != nullptr\n", args[0]);  

248         UI_tree* next_ui = display_controller_component->get_current_ui()  

249             ->next_UI[display_controller_component->get_current_ui()->args[0]];  

250         display_controller_component->set_current_ui(next_ui);  

251     }else{  

252         Serial.printf("display_controller_component->get_current_ui()->  

253             next_UI[%d] == nullptr\n", args[0]);  

254     }  

255 }  

256  

257 void btnESC_func(){  

258  

259     if(display_controller_component->get_current_ui()->prev_UI !=  

260         nullptr){  

261         UI_tree* previous_ui = display_controller_component->  

262             get_current_ui()->prev_UI;  

263         display_controller_component->set_current_ui(previous_ui);  

264     }  

265 } PID_parameters;  

266  

267 struct{  

268  

269     String title = "MODIFY_KP";  

270     uint8_t *args;  

271     uint8_t args_len = 0;  

272     uint8_t nextUI_len = 0;  

273  

274     void display_func(){  

275  

276         display_controller_component->get_display()->printf("Kp = %.2f\n",  

277             display_controller_component->get_temp_PID_params().Kp);  

278  

279         display_controller_component->get_display()->println("-UP button:  

280             increase value");  

281         display_controller_component->get_display()->println("-DOWN button:  

282             increase value");  

283         display_controller_component->get_display()->println("-OK button:  

284             set value");  

285     }  


```

```
280
281     void btnUP_func(){
282         struct_PID_parameters temp = display_controller_component->
283         get_temp_PID_params();
284
285         temp.Kp += 0.1F;
286
287         display_controller_component->set_temp_PID_params(temp);
288     }
289
290     void btnDOWN_func(){
291
292         // Serial.printf("In menu_ui_btnUP_func ,
293         display_controller_component->get_current_ui()->args[0] = %d\n",
294         display_controller_component->get_current_ui()->args[0]);
295         struct_PID_parameters temp = display_controller_component->
296         get_temp_PID_params();
297
298         temp.Kp -= 0.1F;
299
300         display_controller_component->set_temp_PID_params(temp);
301
302     }
303
304     void btnOK_func(){
305
306         center_controller_component->set_PID_parameters(
307         display_controller_component->get_temp_PID_params());
308
309         EEPROM.writeFloat(center_controller_component->get_eeprom_adresses()
310             .eeprom_Kp_address ,
311             display_controller_component->get_temp_PID_params
312             () .Kp);
313         EEPROM.commit();
314
315     }
316
317     void btnESC_func(){
318
319         if(display_controller_component->get_current_ui()->prev_UI !=
320             nullptr){
321
322             display_controller_component->set_temp_PID_params(
323             center_controller_component->get_PID_parameters());
324
325             UI_tree* previous_ui = display_controller_component->
```

```
get_current_ui() -> prev_UI;
    display_controller_component -> set_current_ui(previous_ui);
}
}

}

} modify_kp;

struct{
String title = "MODIFY KI";
uint8_t *args;
uint8_t args_len = 0;
uint8_t nextUI_len = 0;

void display_func(){

    display_controller_component -> get_display() -> printf("Ki = %.2f\n",
display_controller_component -> get_temp_PID_params().Ki);

    display_controller_component -> get_display() -> println("-UP button:
increase value");
    display_controller_component -> get_display() -> println("-DOWN button:
increase value");
    display_controller_component -> get_display() -> println("-OK button:
set value");

}

void btnUP_func(){
    struct_PID_parameters temp = display_controller_component ->
get_temp_PID_params();

    temp.Ki += 0.1F;

    display_controller_component -> set_temp_PID_params(temp);
}

void btnDOWN_func(){

    // Serial.printf("In menu_ui_btnUP_func,
display_controller_component -> get_current_ui() -> args[0] = %d\n",
display_controller_component -> get_current_ui() -> args[0]);
    struct_PID_parameters temp = display_controller_component ->
get_temp_PID_params();

    temp.Ki -= 0.1F;
```

```

354
355     display_controller_component->set_temp_PID_params(temp);
356
357 }
358
359 void btnOK_func(){
360
361     center_controller_component->set_PID_parameters(
362     display_controller_component->get_temp_PID_params());
363
364     EEPROM.writeFloat(center_controller_component->get_eeprom_adresses()
365     .eeprom_Ki_address,
366             display_controller_component->get_temp_PID_params()
367     ().Ki);
368     EEPROM.commit();
369 }
370
371 void btnESC_func(){
372
373     if(display_controller_component->get_current_ui()->prev_UI !=
374     nullptr){
375
376         display_controller_component->set_temp_PID_params(
377         center_controller_component->get_PID_parameters());
378
379         UI_tree* previous_ui = display_controller_component->
380         get_current_ui()->prev_UI;
381         display_controller_component->set_current_ui(previous_ui);
382     }
383
384 }
385
386 struct{
387
388     String title = "MODIFY_KD";
389     uint8_t *args;
390     uint8_t args_len = 0;
391     uint8_t nextUI_len = 0;
392
393     void display_func(){
394
395         display_controller_component->get_display()->printf("Kd = %.2f\n",
396         display_controller_component->get_temp_PID_params().Kd);
397
398 }
```

```
393     display_controller_component->get_display()->println("-UP button:  
394     increase value");  
395     display_controller_component->get_display()->println("-DOWN button:  
396     increase value");  
397     display_controller_component->get_display()->println("-OK button:  
398     set value");  
399 }  
400  
401 void btnUP_func(){  
402     struct_PID_parameters temp = display_controller_component->  
403     get_temp_PID_params();  
404  
405     temp.Kd += 0.1F;  
406  
407     display_controller_component->set_temp_PID_params(temp);  
408 }  
409  
410 void btnDOWN_func(){  
411  
412     // Serial.printf("In menu_ui_btnUP_func,  
413     display_controller_component->get_current_ui()->args[0] = %d\n",  
414     display_controller_component->get_current_ui()->args[0]);  
415     struct_PID_parameters temp = display_controller_component->  
416     get_temp_PID_params();  
417  
418     temp.Kd -= 0.1F;  
419  
420     display_controller_component->set_temp_PID_params(temp);  
421 }  
422  
423 void btnOK_func(){  
424  
425     center_controller_component->set_PID_parameters(  
426     display_controller_component->get_temp_PID_params());  
427  
428     EEPROM.writeFloat(center_controller_component->get_eeprom_adresses()  
429     .eeprom_Kd_address,  
430             display_controller_component->get_temp_PID_params  
431     () .Kd);  
432     EEPROM.commit();  
433 }  
434  
435 void btnESC_func(){
```

```
429
430     if(display_controller_component->get_current_ui()->prev_UI != nullptr){
431
432         display_controller_component->set_temp_PID_params(
433             center_controller_component->get_PID_parameters());
434
435         UI_tree* previous_ui = display_controller_component->
436             get_current_ui()->prev_UI;
437         display_controller_component->set_current_ui(previous_ui);
438     }
439 } modify_kd;
440
441 struct{
442
443     String title = "MODIFY SETPOINT";
444     uint8_t *args;
445     uint8_t args_len = 0;
446     uint8_t nextUI_len = 0;
447
448     void display_func(){
449
450         display_controller_component->get_display()->printf("Setpoint = %.2f
451 \n", display_controller_component->get_temp_PID_params().setpoint);
452
453         display_controller_component->get_display()->println("-UP button:
454 increase value");
455         display_controller_component->get_display()->println("-DOWN button:
456 increase value");
457         display_controller_component->get_display()->println("-OK button:
458 set value");
459
460     }
461
462     void btnUP_func(){
463         struct_PID_parameters temp = display_controller_component->
464             get_temp_PID_params();
465
466         temp.setpoint += 0.1F;
467
468         display_controller_component->set_temp_PID_params(temp);
469     }
470
471     void btnDOWN_func(){
```

```

467
468     // Serial.printf("In menu_ui_btnUP_func ,
469     display_controller_component->get_current_ui()->args[0] = %d\n",
470     display_controller_component->get_current_ui()->args[0]);
471     struct_PID_parameters temp = display_controller_component->
472     get_temp_PID_params();
473
474     temp.setpoint -= 0.1F;
475
476     display_controller_component->set_temp_PID_params(temp);
477
478 }
479
480 void btnOK_func(){
481
482     center_controller_component->set_PID_parameters(
483     display_controller_component->get_temp_PID_params());
484
485     EEPROM.writeFloat(center_controller_component->get_eeprom_adresses()
486     .eeprom_setpoint_address,
487             display_controller_component->get_temp_PID_params
488     ()->setpoint);
489     EEPROM.commit();
490
491 }
492
493 void btnESC_func(){
494
495     if(display_controller_component->get_current_ui()->prev_UI !=
496     nullptr){
497
498         display_controller_component->set_temp_PID_params(
499         center_controller_component->get_PID_parameters());
500
501         UI_tree* previous_ui = display_controller_component->
502         get_current_ui()->prev_UI;
503         display_controller_component->set_current_ui(previous_ui);
504     }
505
506 }
507
508 } modify_setpoint;
509 #pragma endregion
510
511 void display_controller::create_all_ui(void){
512     // Creat MENU UI
513     ESP_LOGI( this->TAG , "Creating Menu UI");

```

```

504     current_UI = new UI_tree( menu_ui.title, menu_ui.args,
505                               menu_ui.args_len, nullptr, menu_ui.nextUI_len,
506                               [](){menu_ui.display_func();},
507                               [](){menu_ui.btnUP_func();},
508                               [](){menu_ui.btnDOWN_func();},
509                               [](){menu_ui.btnOK_func();},
510                               [](){menu_ui.btnESC_func();});
511
512     if(current_UI != nullptr){
513         ESP_LOGI( this->TAG, "Creating Menu UI successfully");
514     }
515
516     UI_tree* temp_ui = current_UI;
517
518 // Create START AND PLOT UI at current_ui->nextUI[0]
519     if(current_UI->next_UI[0] == nullptr){
520
521         ESP_LOGI( this->TAG, "Creating Start and plot UI");
522
523         UI_tree* start_and_plot_ui_temp = new UI_tree(start_n_plot_ui.title,
524                                         nullptr, start_n_plot_ui.args_len,
525                                         current_UI,
526                                         start_n_plot_ui.
527                                         nextUI_len,
528                                         [](){start_n_plot_ui.
529                                         display_func();},
530                                         [](){},
531                                         [](){},
532                                         [](){},
533                                         [](){start_n_plot_ui.
534                                         btnESC_func();});
535
536         temp_ui->next_UI[0] = start_and_plot_ui_temp;
537     }
538
539 // Create PID'S PARAMETERS UI at current_ui->nextUI[1]
540     if(current_UI->next_UI[1] == nullptr){
541
542         ESP_LOGI( this->TAG, "Creating PID's parameters UI");
543
544         UI_tree* pid_parameters_ui_temp = new UI_tree(PID_parameters.title,
545                                         PID_parameters.args, PID_parameters.args_len,

```

```

545                               current_UI ,
546                               PID_parameters .
547                               nextUI_len ,
548                               [] () {PID_parameters .
549                               display_func ();} ,
550                               [] () {PID_parameters .
551                               btnUP_func ();} ,
552                               [] () {PID_parameters .
553                               btnDOWN_func ();} ,
554                               [] () {PID_parameters .
555                               btnOK_func ();} ,
556                               [] () {PID_parameters .
557                               btnESC_func ();});
558                               temp_ui->next_UI [1] = pid_parameters_ui_temp;
559 }
560
561 if (current_UI->next_UI [1] != nullptr){
562     ESP_LOGI( this->TAG, "Created PID's parameters UI successfully");
563 }
564
565 temp_ui = temp_ui->next_UI [1];
566
567 // Create MODIFY KP UI
568 if (temp_ui->next_UI [0] == nullptr){
569
570     ESP_LOGI( this->TAG, "Creating Modify Kp UI");
571
572     UI_tree* modify_kp_ui_temp = new UI_tree(modify_kp.title, modify_kp.
573     args, modify_kp.args_len,
574                               current_UI->next_UI [1],
575                               modify_kp.nextUI_len,
576                               [] () {modify_kp.display_func ()}
577                               ;},
578                               [] () {modify_kp.btnUP_func ();},
579                               [] () {modify_kp.btnDOWN_func ()}
580                               ;},
581                               [] () {modify_kp.btnOK_func ();},
582                               [] () {modify_kp.btnESC_func ()}
583                               ;);
584
585     temp_ui->next_UI [0] = modify_kp_ui_temp;
586 }
587
588 if (temp_ui->next_UI [0] != nullptr){
589     ESP_LOGI( this->TAG, "Created Modify Kp UI successfully");
590 }
591
592

```

```

581     // Create MODIFY KI UI
582     if(temp_ui->next_UI[1] == nullptr){
583
584         ESP_LOGI( this->TAG, "Creating Modify Ki UI");
585
586         UI_tree* modify_ki_ui_temp = new UI_tree(modify_ki.title, modify_ki.
587         args, modify_ki.args_len,
588
589                     current_UI->next_UI[1],
590                     modify_ki.nextUI_len,
591                     [](){modify_ki.display_func()
592             ;},
593
594                     [](){modify_ki.btnUP_func();},
595                     [](){modify_ki.btnDOWN_func()
596             ;},
597
598                     [](){modify_ki.btnOK_func();},
599                     [](){modify_ki.btnESC_func()
600             ;});
601
602         temp_ui->next_UI[1] = modify_ki_ui_temp;
603     }
604
605
606     if(temp_ui->next_UI[1] != nullptr){
607         ESP_LOGI( this->TAG, "Created Modify Ki UI successfully");
608     }
609
610
611     // Create MODIFY KD UI
612     if(temp_ui->next_UI[2] == nullptr){
613
614         ESP_LOGI( this->TAG, "Creating Modify Kd UI");
615
616         UI_tree* modify_kd_ui_temp = new UI_tree(modify_kd.title, modify_kd.
617         args, modify_kd.args_len,
618
619                     current_UI->next_UI[1],
620                     modify_kd.nextUI_len,
621                     [](){modify_kd.display_func()
622             ;},
623
624                     [](){modify_kd.btnUP_func();},
625                     [](){modify_kd.btnDOWN_func()
626             ;},
627
628                     [](){modify_kd.btnOK_func();},
629                     [](){modify_kd.btnESC_func()
630             ;});
631
632         temp_ui->next_UI[2] = modify_kd_ui_temp;
633     }
634
635
636     if(temp_ui->next_UI[2] != nullptr){
637         ESP_LOGI( this->TAG, "Created Modify Kd UI successfully");
638     }

```

```

619     }
620
621     // Create MODIFY SETPOINT UI
622     if(temp_ui->next_UI[3] == nullptr){
623
624         ESP_LOGI( this->TAG , "Creating Modify Setpoint UI");
625
626         UI_tree* modify_kd_ui_temp = new UI_tree(modify_setpoint.title,
627             modify_setpoint.args, modify_setpoint.args_len,
628                                         current_UI->next_UI[1],
629                                         modify_setpoint.nextUI_len,
630                                         [](){modify_setpoint.
631             display_func();},
632                                         [](){modify_setpoint.
633             btnUP_func();},
634                                         [](){modify_setpoint.
635             btnDOWN_func();},
636                                         [](){modify_setpoint.
637             btnOK_func();},
638                                         [](){modify_setpoint.
639             btnESC_func();});
640         temp_ui->next_UI[3] = modify_kd_ui_temp;
641     }
642
643     if(temp_ui->next_UI[3] != nullptr){
644         ESP_LOGI( this->TAG , "Created Modify Setpoint UI successfully");
645     }
646
647     display_controller::display_controller( uint8_t fps,
648                                         uint8_t priority,
649                                         uint8_t width,
650                                         uint8_t height)
651 : fps(fps),
652   priority(priority),
653   scr_width(width),
654   scr_height(height)
655 {
656
657     ESP_LOGI(this->TAG , "Display controller is initialising\n");
658
659     this->temp_PID_params = center_controller_component->get_PID_parameters
660     ();
661
662     // Initialize screen
663     this->display = new Adafruit_SSD1306(this->scr_width, this->scr_height,

```

```
    &Wire, OLED_RESET);  
658  
659 // Create Mutex variable  
660 this->xMutex_menu_curr_opt = xSemaphoreCreateMutex();  
661  
662 create_all_ui();  
663  
664 // Ckecking display succeeded or not  
665 while(!(~this->display->begin(SSD1306_SWITCHCAPVCC, 0x3C))) {  
666  
667     ESP_LOGE(this->TAG, "SSD1306 allocation failed");  
668  
669     delay(1000);  
670 }  
671  
672 button_init();  
673 // Clear the buffer  
674 this->display->clearDisplay();  
675  
676 ESP_LOGI(this->TAG, "Initialized Display controller successfully!");  
677  
678 }  
679  
680 display_controller::~display_controller()  
681 {  
682 }  
683  
684 void display_controller::draw_menu(){  
685  
686     while (true){  
687  
688         this->display->clearDisplay();  
689  
690         this->display->setTextSize(1); // Normal 1:1 pixel scale  
691  
692         this->display->setTextColor(WHITE); // Draw white text  
693  
694         if(xSemaphoreTake(this->xMutex_menu_curr_opt, (this->fps)/  
portTICK_PERIOD_MS) == pdTRUE){  
695  
696             if((this->current_UI->title.length()*6) >= this->display->width()){  
697  
698                 this->display->setCursor(0,0);  
699  
700             }else{  
701
```

```
702         this->display->setCursor((this->display->width()-this->current_UI  
703             ->title.length()*6)/2,0);  
704     }  
705  
706     this->display->print("-");  
707     this->display->print(this->current_UI->title);  
708     this->display->println("-");  
709  
710     xSemaphoreGive(this->xMutex_menu_curr_opt);  
711 }else {  
712     ESP_LOGI(this->TAG, "Can't take currentUI");  
713 }  
714  
715     this->display->setTextSize(1);  
716  
717 // TODO: Display current UI  
718 if(xSemaphoreTake(this->xMutex_menu_curr_opt, (this->fps)/  
719 portTICK_PERIOD_MS) == pdTRUE){  
720  
721     this->current_UI->display_func();  
722     xSemaphoreGive(this->xMutex_menu_curr_opt);  
723 }else {  
724     ESP_LOGI(this->TAG, "Can't take currentUI");  
725 }  
726  
727     this->display->display();  
728 }  
729 }  
730  
731 void display_controller::draw_menu_wrapper(void* arg){  
732  
733     display_controller* display_controller_instance = static_cast<  
734         display_controller*>(arg);  
735  
736     display_controller_instance->draw_menu();  
737 }  
738  
739 void display_controller::get_btn_OK(){  
740     while (true)  
741     {  
742         if (btn_OK.buttonPressed) {  
743             ESP_LOGI(this->TAG, "Button OK has been pressed");  
744 }
```

```
745
746     if(xSemaphoreTake(this->xMutex_menu_curr_opt, this->fps/
747     portTICK_PERIOD_MS) == pdTRUE){
748
749         this->current_UI->btnOK_func();
750
751         xSemaphoreGive(this->xMutex_menu_curr_opt);
752
753     }else {
754         ESP_LOGI(this->TAG, "Can't take currentUI");
755     }
756
757     btn_OK.buttonPressed = false;
758 }
759 vTaskDelay(200/portTICK_PERIOD_MS);
760 }
761
762 void display_controller::get_btn_OK_wrapper(void* arg){
763
764     display_controller* display_controller_instance = static_cast<
765         display_controller*>(arg);
766     display_controller_instance->get_btn_OK();
767 }
768
769
770 void display_controller::get_btn_ESC(){
771
772     while (true)
773     {
774         if (btn_ESC.buttonPressed) {
775
776             ESP_LOGI( this->TAG, "Button ESC has been pressed");
777
778             if(xSemaphoreTake(this->xMutex_menu_curr_opt, this->fps/
779             portTICK_PERIOD_MS) == pdTRUE){
780
781                 this->current_UI->btnESC_func();
782
783                 xSemaphoreGive(this->xMutex_menu_curr_opt);
784
785             }else {
786                 ESP_LOGI(this->TAG, "Can't take currentUI");
787             }
788             btn_ESC.buttonPressed = false;
```

```
788     }
789     vTaskDelay(200/portTICK_PERIOD_MS);
790 }
791 }
792
793 void display_controller::get_btn_ESC_wrapper(void* arg){
794
795     display_controller* display_controller_instance = static_cast<
796         display_controller*>(arg);
797     display_controller_instance->get_btn_ESC();
798 }
799
800 void display_controller::get_btn_UP(){
801
802     while (true)
803     {
804         if (btn_UP.buttonPressed) {
805
806             ESP_LOGI(this->TAG, "Button UP has been pressed");
807
808             if(xSemaphoreTake(this->xMutex_menu_curr_opt, this->fps/
809             portTICK_PERIOD_MS) == pdTRUE){
810
811                 this->current_UI->btnUP_func();
812
813                 xSemaphoreGive(this->xMutex_menu_curr_opt);
814             }else {
815                 ESP_LOGI(this->TAG, "Can't take currentUI");
816             }
817
818             btn_UP.buttonPressed = false;
819         }
820
821         vTaskDelay(200/portTICK_PERIOD_MS);
822     }
823 }
824
825 void display_controller::get_btn_UP_wrapper(void* arg){
826
827     display_controller* display_controller_instance = static_cast<
828         display_controller*>(arg);
829     display_controller_instance->get_btn_UP();
830 }
```

```
831
832
833 void display_controller::get_btn_DOWN(){
834
835     while (true)
836     {
837         if (btn_DOWN.buttonPressed) {
838             ESP_LOGI( this->TAG, "Button DOWN has been pressed");
839
840             if(xSemaphoreTake(this->xMutex_menu_curr_opt, this->fps/
841             portTICK_PERIOD_MS) == pdTRUE){
842
843                 this->current_UI->btnDOWN_func();
844
845                 xSemaphoreGive(this->xMutex_menu_curr_opt);
846
847             }else {
848                 ESP_LOGI(this->TAG, "Can't take currentUI");
849             }
850
851             btn_DOWN.buttonPressed = false;
852         }
853         vTaskDelay(200/portTICK_PERIOD_MS);
854     }
855
856 void display_controller::get_btn_DOWN_wrapper(void* arg){
857
858     display_controller* display_controller_instance = static_cast<
859         display_controller*>(arg);
860     display_controller_instance->get_btn_DOWN();
861
862
863 Adafruit_SSD1306* display_controller::get_display(){
864     return this->display;
865 }
866
867 UI_tree* display_controller::get_current_ui(){
868
869     return this->current_UI;
870 }
871
872
873 void display_controller::set_current_ui(UI_tree* ui){
874 }
```

```
875     this->current_UI = ui;
876
877 }
878
879 uint8_t display_controller::get_fps(){
880     return this->fps;
881 }
882
883 struct_PID_parameters display_controller::get_temp_PID_params(){
884     return this->temp_PID_params;
885 }
886
887 void display_controller::set_temp_PID_params(struct_PID_parameters
888     new_value){
889     this->temp_PID_params = new_value;
890 }
891
892 void display_controller::run(){
893     if(xTaskCreatePinnedToCore( get_btn_OK_wrapper,
894                                 "get_btn_OK",
895                                 2048,
896                                 this,
897                                 6,
898                                 nullptr,
899                                 1) == pdPASS){
900
901         ESP_LOGI(this->TAG, "Created get_btn_OK task successfully!");
902     }else{
903
904         ESP_LOGE(this->TAG, "Created get_btn_OK task failed!");
905         while (true){}
906
907     }
908
909     if(xTaskCreatePinnedToCore( get_btn_ESC_wrapper,
910                                 "get_btn_ESC",
911                                 2048,
912                                 this,
913                                 6,
914                                 nullptr,
915                                 1) == pdPASS){
916
917         ESP_LOGI(this->TAG, "Created get_btn_ESC task successfully!");
918
919     }else{
```

```
920
921     ESP_LOGE(this->TAG, "Created get_btn_ESC task failed!");
922     while (true){}
923
924 }
925
926 if(xTaskCreatePinnedToCore( get_btn_UP_wrapper,
927                             "get_btn_UP",
928                             2048,
929                             this,
930                             6,
931                             nullptr,
932                             1) == pdPASS){
933
934     ESP_LOGI(this->TAG, "Created get_btn_UP task successfully!");
935
936 }else{
937
938     ESP_LOGE(this->TAG, "Created get_btn_UP task failed!");
939
940     while (true){}
941
942 }
943
944 if(xTaskCreatePinnedToCore( get_btn_DOWN_wrapper,
945                             "get_btn_DOWN",
946                             2048,
947                             this,
948                             6,
949                             nullptr,
950                             1) == pdPASS){
951
952     ESP_LOGI(this->TAG, "Created get_btn_DOWN task successfully!");
953
954 }else{
955
956     ESP_LOGE(this->TAG, "Created get_btn_DOWN task failed!");
957
958     while (true){}
959
960 }
961
962 if(xTaskCreatePinnedToCore( draw_menu_wrapper,
963                             "draw_menu",
964                             2048,
965                             this,
```

```
966                     this->priority ,  
967                     nullptr ,  
968                     1) == pdPASS){  
969  
970     ESP_LOGI(this->TAG , "Created draw_menu task successfully!");  
971  
972 }else{  
973  
974     ESP_LOGE(this->TAG , "Created draw_menu task failed!");  
975  
976     while (true){}  
977  
978 }  
979 }  
980  
981 display_controller* display_controller_component;
```

Program 6.12: display_controller.cpp

Chapter 7

Demo

*You can visit the Project's repository on github to see more code's detail.
You also watch final result video on my youtube channel by clicking here.*

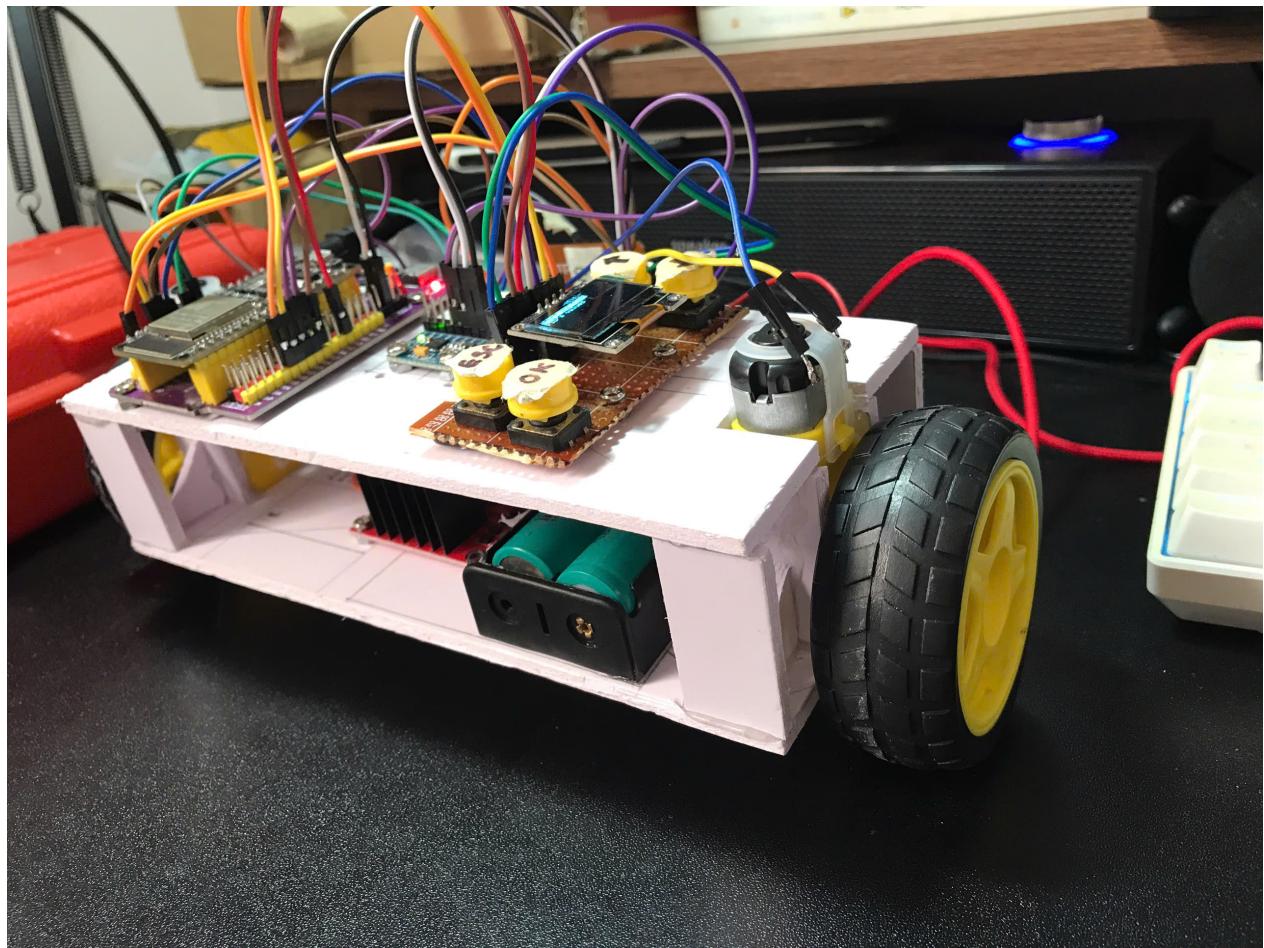


Figure 7.1: Self-Balancing Robot front view

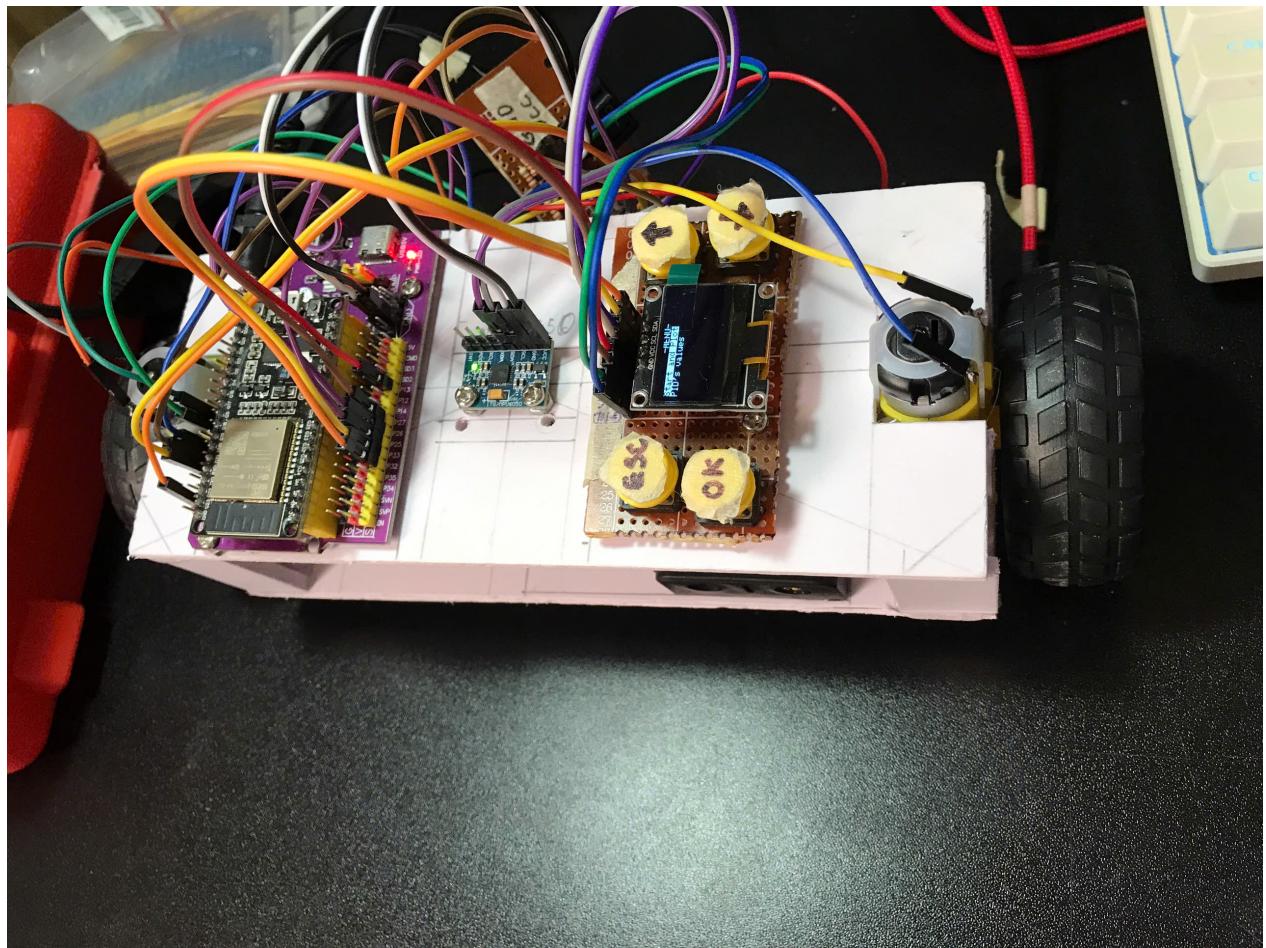


Figure 7.2: Self-Balancing Robot up view

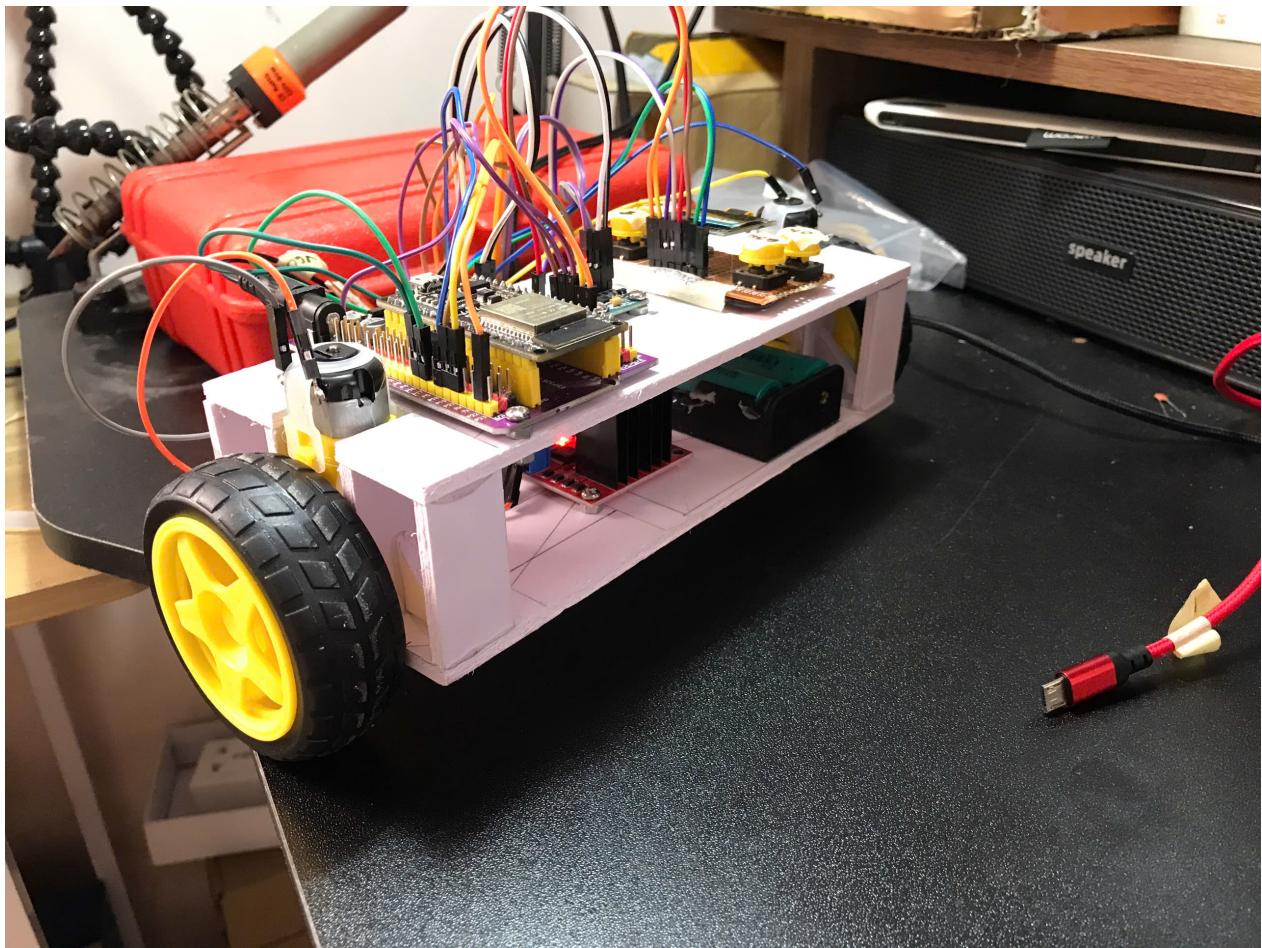


Figure 7.3: Self-Balancing Robot back view

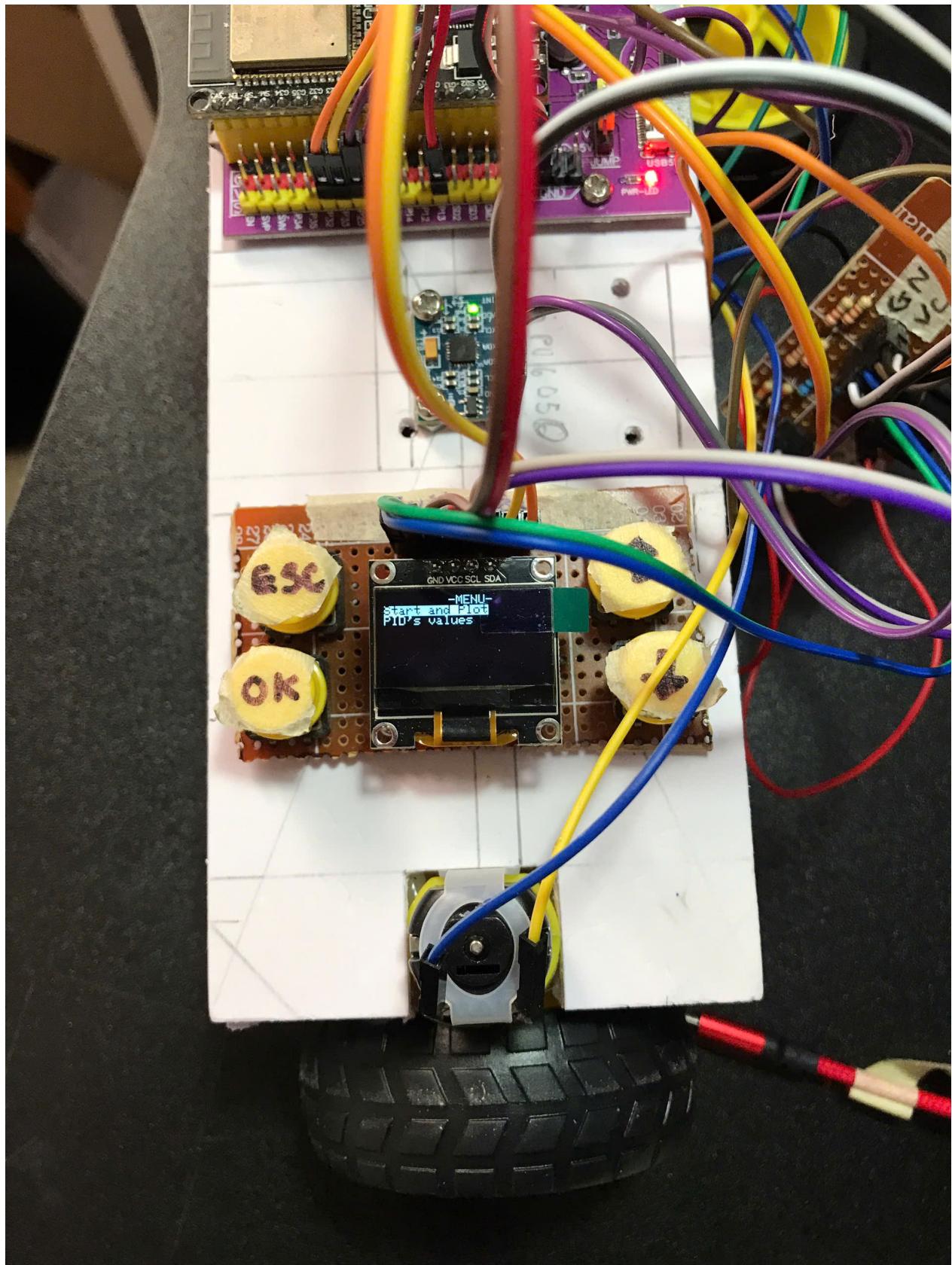


Figure 7.4: Menu UI

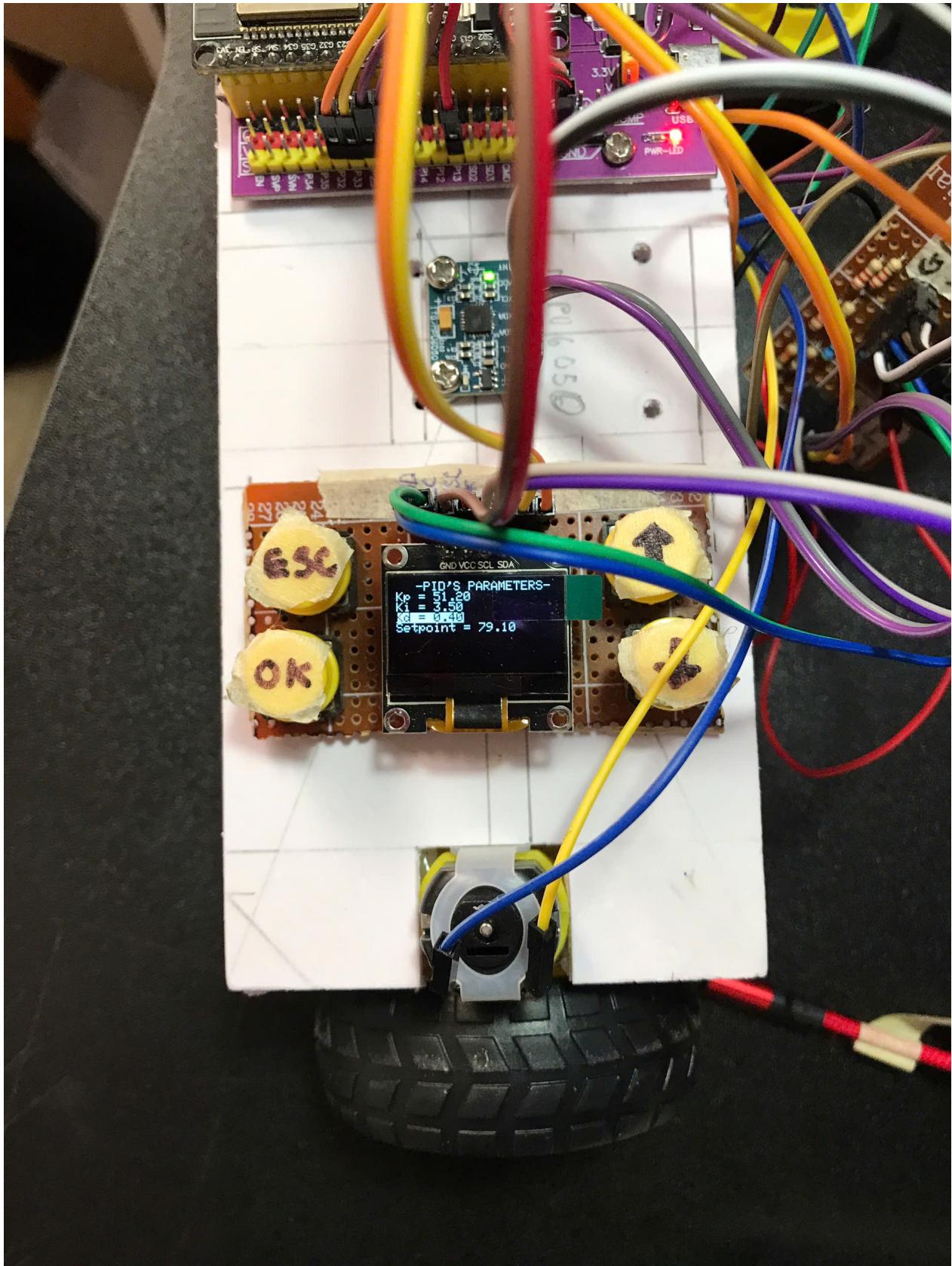


Figure 7.5: PID parameters UI

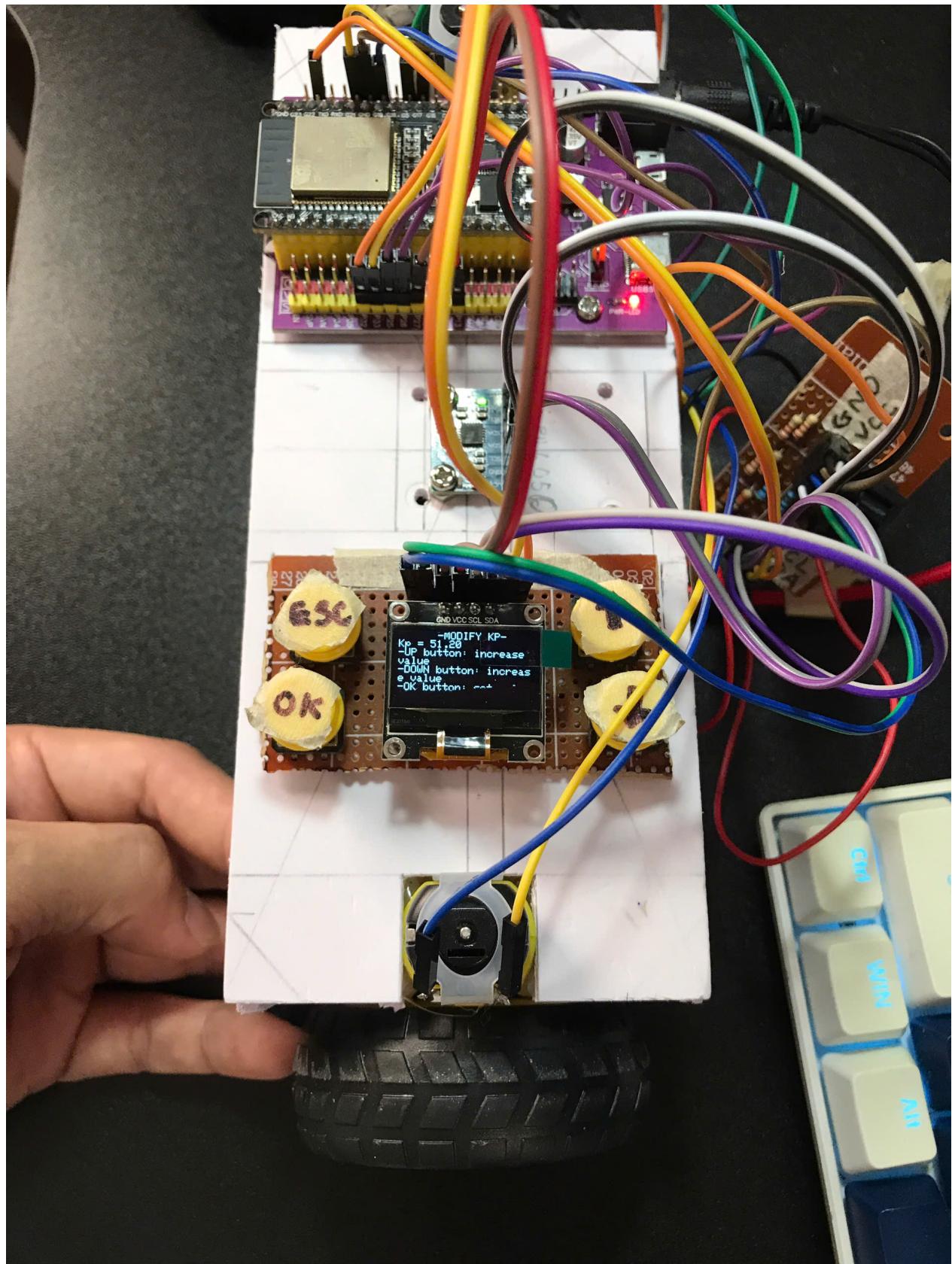


Figure 7.6: Modify Kx UI

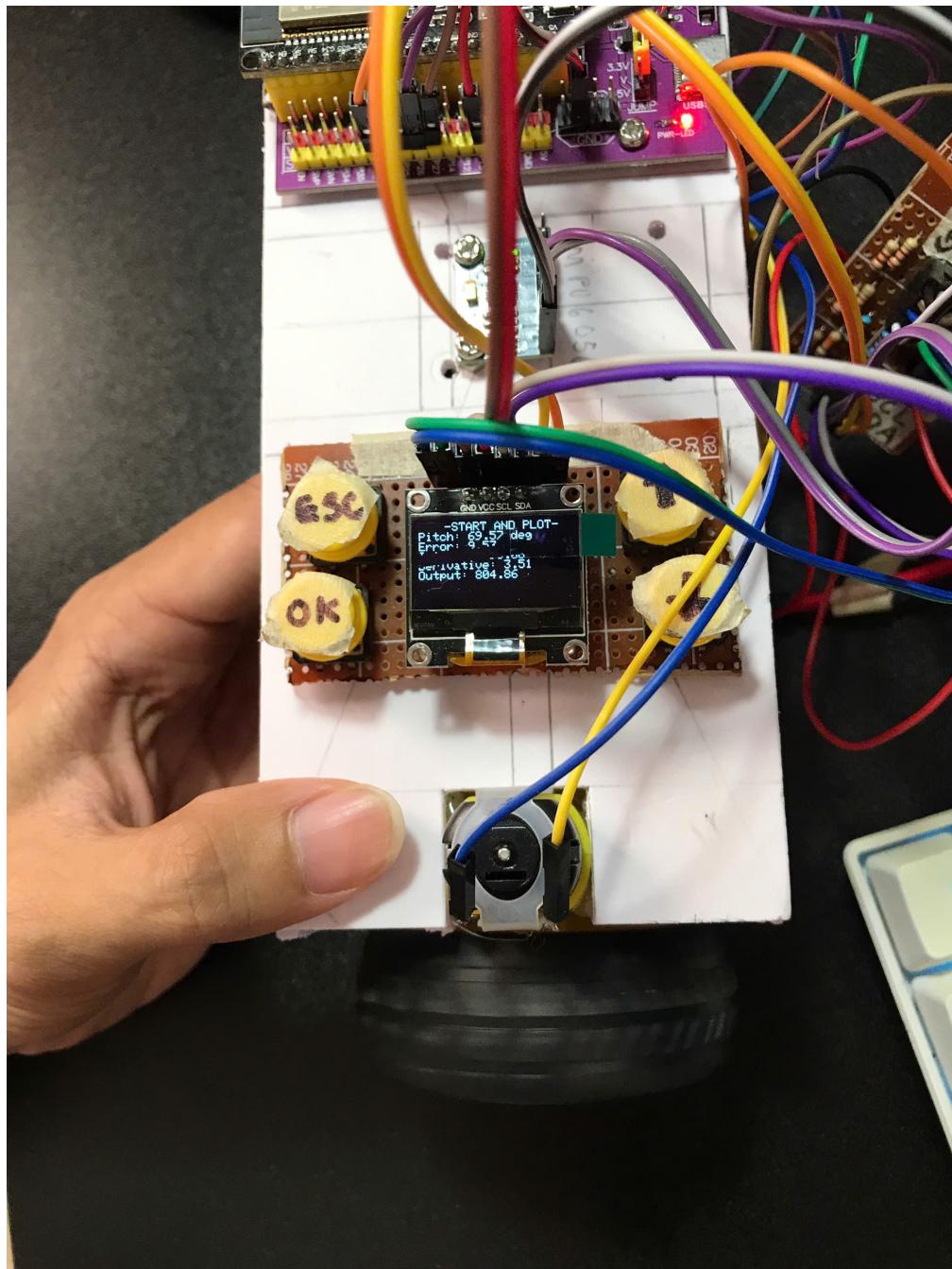


Figure 7.7: Start and plot UI

Thank you for paying your time to reading this Self-Balancing Robot report.

End.