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

LIST OF TABLES

LIST OF IMAGES

Listings

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 Kp, Ki, and Kd 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***:

```
lib_deps =
  adafruit/Adafruit SSD1306@^2.5.10
  adafruit/Adafruit GFX Library@^1.11.9
  adafruit/Adafruit Unified Sensor@^1.1.14
  adafruit/Adafruit MPU6050@^2.2.6
  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:

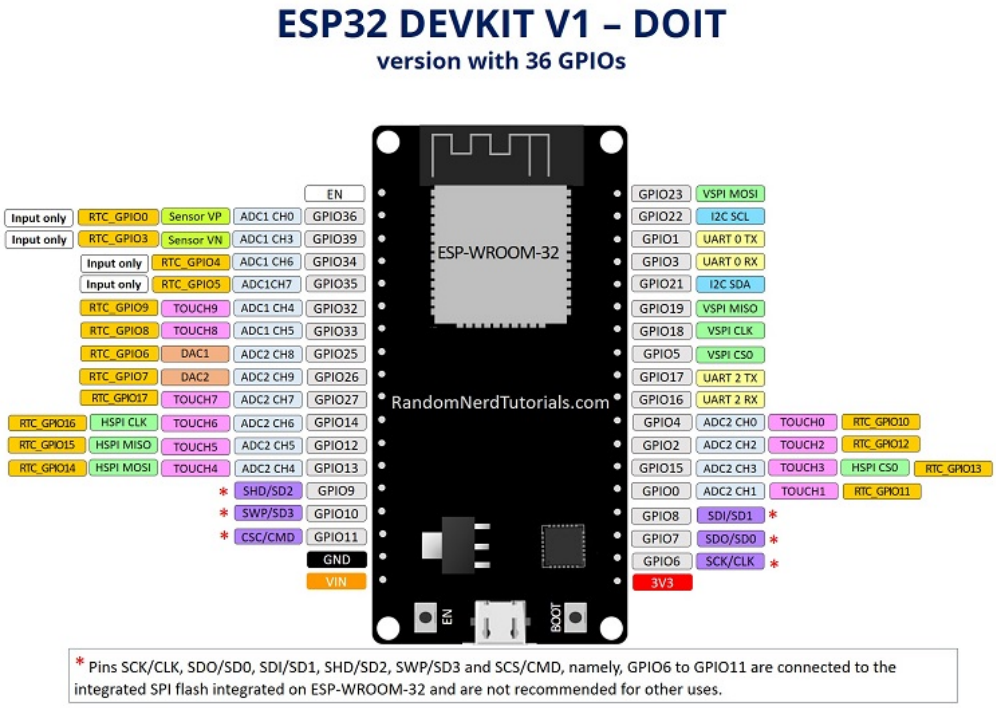

Figure 2.4: ESP 32 WROOM 32 pinouts

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.

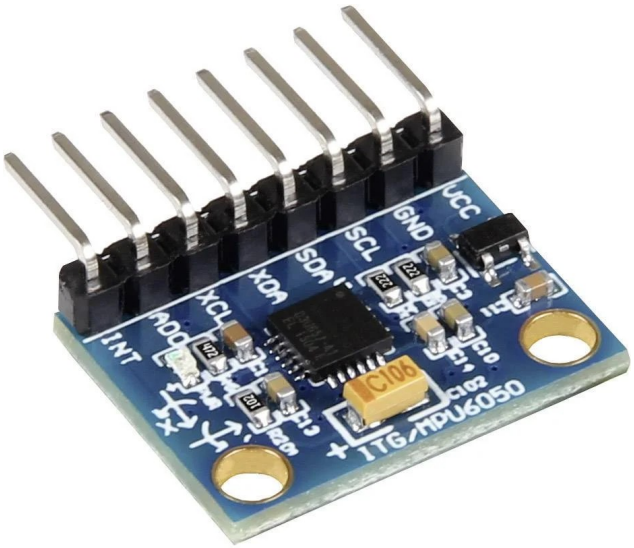

Figure 2.5: MPU 6050

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.
- UP (down arrow button): responsible for moving down or decreasing the parameter value.

Figure 2.7: Button

2.2.5 Li-on batterry

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 batterty 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.

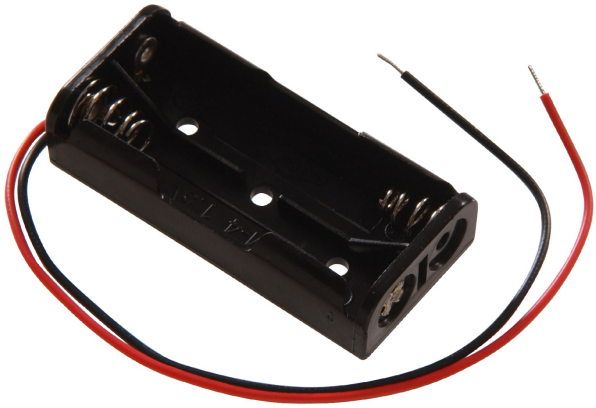

Figure 2.9: 2-pins pack

2.2.7 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.

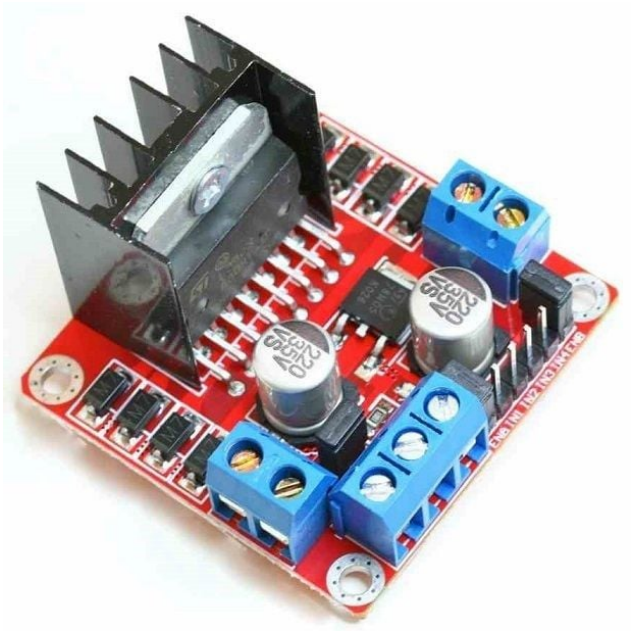

Figure 2.10: L298N motor driver

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.

2.2.8 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 2.11: DC motor and wheels

2.2.9 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 2.12: 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

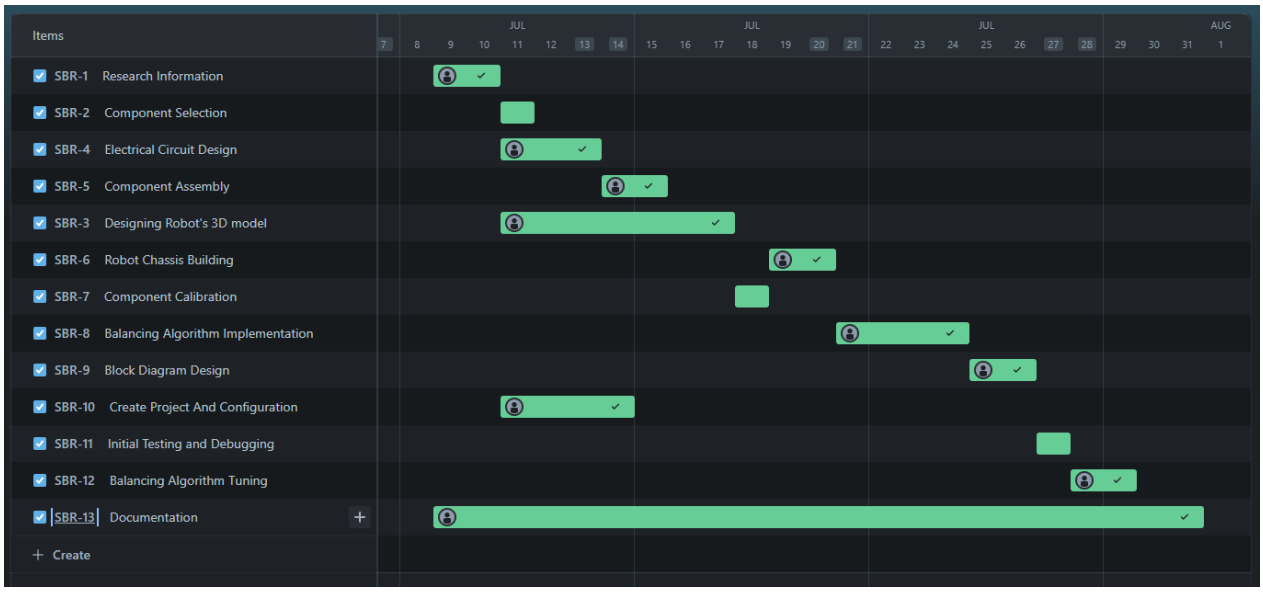

Figure 3.1: Project timeline

Chapter 4

Schematic

Include Robot Schematic

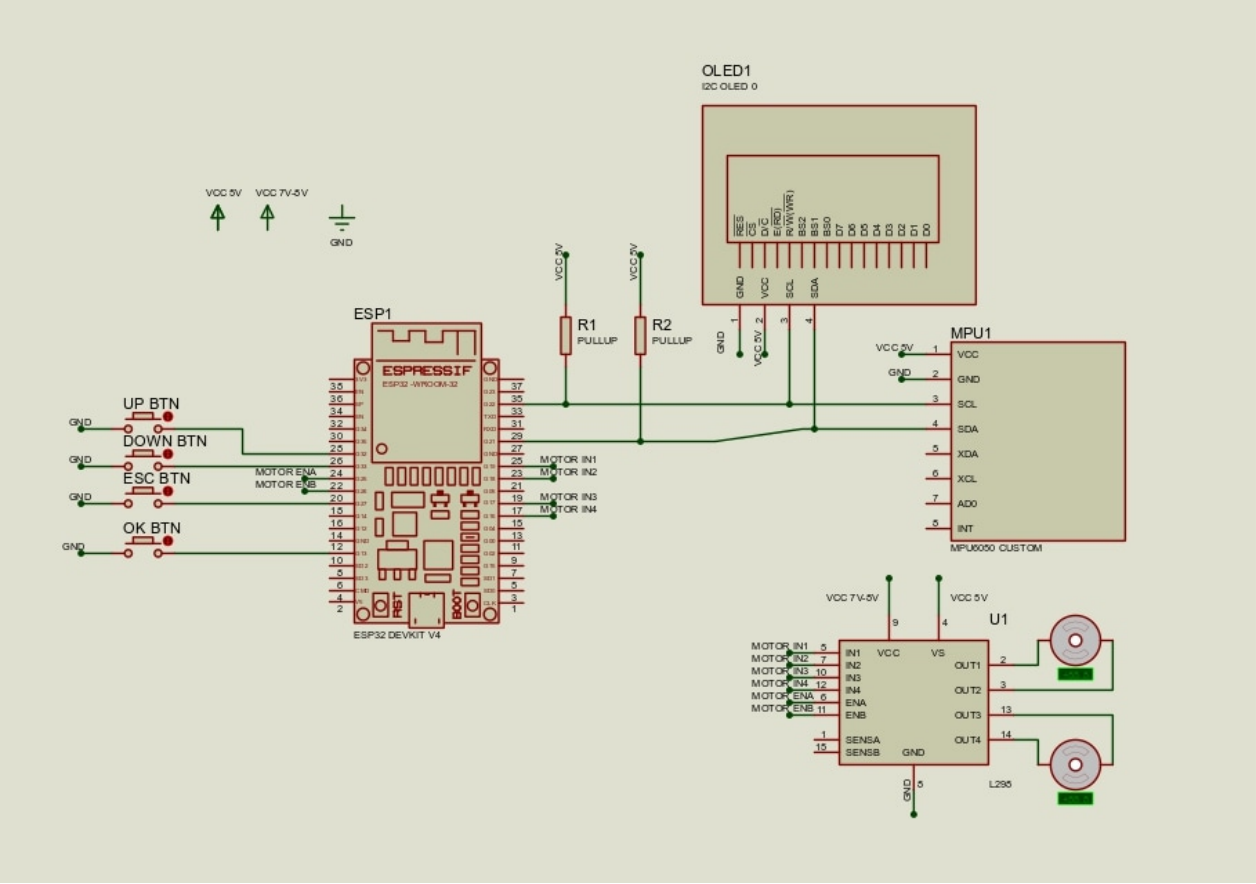

Figure 4.1: Self-balancing robot schematic

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

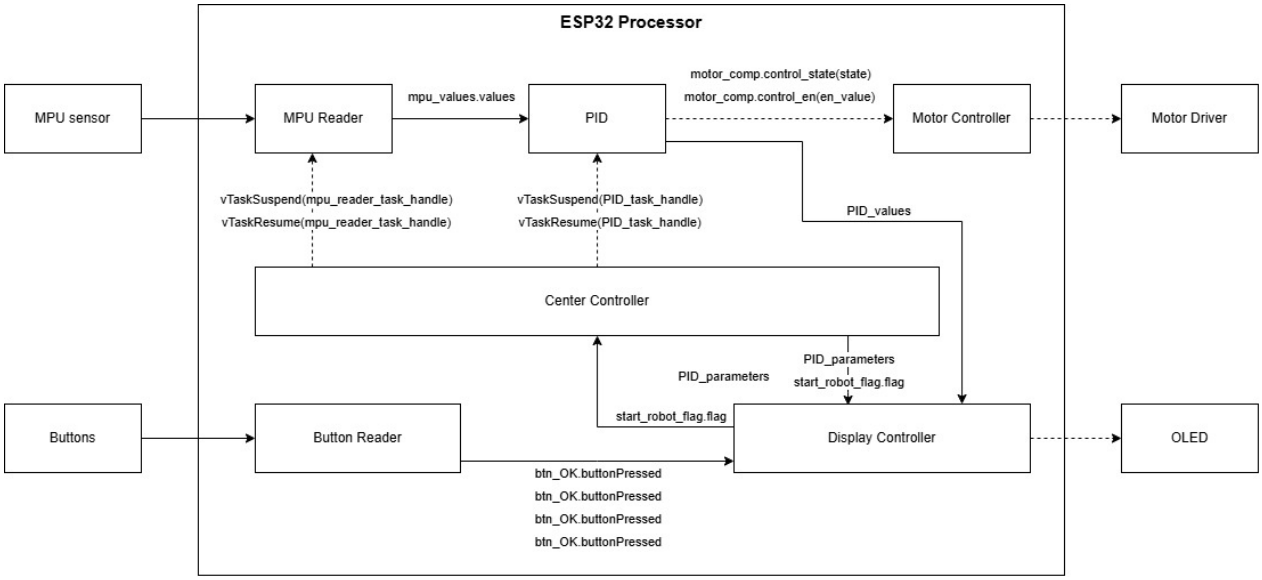

Figure 5.1: Block diagram of Robot

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 filted_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:

Receving 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

```
#pragma once
#ifndef MPU_READER_HPP
#define MPU_READER_HPP

#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
#include <Arduino.h>
#include <global.hpp>

class mpu_reader
{
private:

    const char* TAG = "MPU READER";
    const int8_t sample_time;
    const uint8_t priority;

    Adafruit_MPU6050* mpu;
    TaskHandle_t mpu_reader_task_handle;

    struct{
        // QueueHandle_t queue;
        SemaphoreHandle_t bi_semaphore;
        struct_mpu_reader values;
    } mpu_values;

    Adafruit_Sensor *mpu_accel;
    Adafruit_Sensor *mpu_gyro;

    void sensors_setup();

    void mpu_reading();

    static void mpu_reading_wrapper(void* arg);

public:

    TaskHandle_t get_task_handle();

    // QueueHandle_t get_mpu_values_queue();

    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     this->mpu_values.bi_semaphore = xSemaphoreCreateBinary();
161 
162     // Serial.println("MPU READER, Initialized successfully");
163     ESP_LOGI(this->TAG, "Initialized sensors successfully\n");
164 
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 =
        0.9934);
40      ~PID_block();
41
42      QueueHandle_t get_PID_values_queue();
43      TaskHandle_t get_task_handle();
44      void set_PID_parameters(struct_PID_parameters params);
45      void run();
46
47      void set_prev_time();
48  };
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
```

```
PID_block::~PID_block()
{
}

void PID_block::PID_compute(){

    struct_mpu_reader mpu_values;

    // QueueHandle_t q_mpu_values = mpu_reader_component->
   get_mpu_values_queue();

    SemaphoreHandle_t sem_mpu = mpu_reader_component->get_semaphore_mpu();

    float currentAngle = 0;
    float duration_time = 0;

    float pitch = 0;
    float error = 0;
    float previousError = 0;
    float integral = 0;
    float derivative = 0;
    float output = 0;

    static int motorSpeed = 0;

    vTaskSuspend(PID_task_handle);
    this->prev_time = millis();

    while (true)
    {

        xSemaphoreTake(sem_mpu, portMAX_DELAY);

        if(duration_time < (this->delay_time/1000)) continue;

        mpu_values = mpu_reader_component->get_mpu_values();

        duration_time = (millis() - this->prev_time) * 0.001;

        pitch = atan2(mpu_values.accel.z, mpu_values.accel.x);
        currentAngle = this->alpha*(currentAngle + mpu_values.gyro.y *
   duration_time) + (1 - this->alpha)*(pitch);

        error = this->PID_params.setpoint - (currentAngle * RAD_TO_DEG);

        integral += error * duration_time;
```

```
68        derivative = (error - previousError) / duration_time;
69
70        output = this->PID_params.Kp * error + this->PID_params.Ki *
    integral + this->PID_params.Kd * derivative;
71
72        motorSpeed = constrain(output, -255, 255);
73
74        if (motorSpeed > 0) {
75
76            motor_controller_component->control_state(MOVE_FORWARD);
77            motor_controller_component->control_both_EN(motorSpeed);
78
79        } else {
80
81            motor_controller_component->control_state(MOVE_BACKWARD);
82            motor_controller_component->control_both_EN(abs(motorSpeed));
83
84        }
85
86        ESP_LOGI(this->TAG,
87                 "Current Angle = %.2f,\terror = %.2f,\tintegral = %.2f,\
    tderivative = %.2f,\toutput = %.2f,\tduration_time = %.2f",
88                 currentAngle*RAD_TO_DEG,
89                 error,
90                 integral,
91                 derivative,
92                 output,
93                 duration_time);
94
95        this->PID_values.values = {
96
97            .filted_pitch = currentAngle,
98            .gyro_angle_Y = mpu_values.gyro.y,
99            .output = output,
100           .error = error,
101           .integral = integral,
102           .derivative = derivative,
103
104       };
105
106       if(xQueueSend(this->PID_values.queue, &(this->PID_values.values),
    this->delay_time/portTICK_PERIOD_MS) == pdTRUE){
107
108           ESP_LOGI(this->TAG, "Sent PID values successfully");
109
110       }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
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,
    portMAX_DELAY) == pdTRUE){
72              // current_movement_state++;
73              // if(current_movement_state > 4) current_movement_state = 0;
74
75              // Serial.printf("current_movement_state = %d\n",this->
    current_movement_state);
76              ESP_LOGI(this->TAG,
77                      "current_movement_state = %d\n",
```

```
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) ==
    pdTRUE){
97
98              analogWrite(MOTOR_ENA_PIN, this->ENA_value.value);
99
100             xSemaphoreGive(this->ENA_value.xMutex_value);
101         }
102
103         if(xSemaphoreTake(this->ENB_value.xMutex_value, portMAX_DELAY) ==
    pdTRUE){
104
105             analogWrite(MOTOR_ENB_PIN, this->ENB_value.value);
106
107             xSemaphoreGive(this->ENB_value.xMutex_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<
    motor_controller*>(arg);
119 
120     motor_controller_instance->motor_work();
121 }
122 
123 void motor_controller::run(void){
124 
125     if( xTaskCreatePinnedToCore(motor_work_wrapper,
126                                 "motor_work",
127                                 2024,
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->
    motor_controller_task_handle));
137 
138     }else{
139         // Serial.println("Create motor_controller_task() task failed!");
140         ESP_LOGE(this->TAG, "Create motor_controller_task task failed!\n")
    ;
141 
142         while (true){}
143 
144     }
145 }
146 
147 TaskHandle_t motor_controller::get_task_hanlde(){
148     return this->motor_controller_task_handle;
149 }
150 
151 void motor_controller::write_state(uint8_t state){
152 
153     if(xSemaphoreTake(this->current_movement_state.xMutex_state,
    portMAX_DELAY) == pdTRUE){
154 
155         if(state < MOTOR_MIN_STATE){
156 
157             this->current_movement_state.state = MOTOR_MIN_STATE;
158 
159         }else if (state > MOTOR_MAX_STATE){
```

```
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
248         digitalWrite(MOTOR_IN3_PIN,
249                      this->motor_in_state[this->current_movement_state.
    state][IN3_SIGNAL_INDEX]);
250
251         digitalWrite(MOTOR_IN4_PIN,
252                      this->motor_in_state[this->current_movement_state.
    state][IN4_SIGNAL_INDEX]);
253
254         xSemaphoreGive(this->current_movement_state.xMutex_state);
255     }
256
257     if(xSemaphoreTake(this->ENA_value.xMutex_value, portMAX_DELAY) ==
    pdTRUE){
258
259         this->ENA_value.value = 0;
260
261         analogWrite(MOTOR_ENA_PIN, this->ENA_value.value);
262
263         xSemaphoreGive(this->ENA_value.xMutex_value);
264     }
265
266     if(xSemaphoreTake(this->ENB_value.xMutex_value, portMAX_DELAY) ==
    pdTRUE){
267
268         this->ENB_value.value = 0;
269
270         analogWrite(MOTOR_ENB_PIN, this->ENB_value.value);
271
272         xSemaphoreGive(this->ENB_value.xMutex_value);
273     }
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         this->current_movement_state.state = MIN_MOVE_STATE_VALUE;
286     }
287
```

```
288     if(state > MAX_MOVE_STATE_VALUE) {
289 
290         ESP_LOGI(this->TAG, "Input state is greater than
    MAX_MOVE_STATE_VALUE");
291 
292         this->current_movement_state.state = MAX_MOVE_STATE_VALUE;
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
    ][IN1_SIGNAL_INDEX]);
301 
302     // digitalWrite(MOTOR_IN2_PIN,
303     //              this->motor_in_state[this->current_movement_state.state
    ][IN2_SIGNAL_INDEX]);
304 
305     digitalWrite(MOTOR_IN3_PIN,
306                  this->motor_in_state[this->current_movement_state.state][
    IN3_SIGNAL_INDEX]);
307 
308     digitalWrite(MOTOR_IN4_PIN,
309                  this->motor_in_state[this->current_movement_state.state][
    IN4_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
    ][IN1_SIGNAL_INDEX]);
341
342     // digitalWrite(MOTOR_IN2_PIN,
343     //              this->motor_in_state[this->current_movement_state.state
    ][IN2_SIGNAL_INDEX]);
344
345     digitalWrite(MOTOR_IN3_PIN,
346                  this->motor_in_state[this->current_movement_state.state][
    IN3_SIGNAL_INDEX]);
347
348     digitalWrite(MOTOR_IN4_PIN,
349                  this->motor_in_state[this->current_movement_state.state][
    IN4_SIGNAL_INDEX]);
350
351     // this->ENA_value.value = 0;
352
353     // analogWrite(MOTOR_ENA_PIN, this->ENA_value.value);
354
355     this->ENB_value.value = 0;
356
357     analogWrite(MOTOR_ENB_PIN, this->ENB_value.value);
358
359
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

```cpp
#pragma once
#ifndef CENTER_CONTROLLER_HPP
#define CENTER_CONTROLLER_HPP

#include <Arduino.h>
#include "EEPROM.h"
#include "mpu_reader.hpp"
#include "motor_controller.hpp"
#include "PID_block.hpp"
#include "display_controller.hpp"
#include "global.hpp"

class center_controller
{
private:

    const char* TAG = "CENTER CONTROLLER";
    const int delay_time;

    struct{

        bool flag;
        SemaphoreHandle_t xMutex_flag;

    } start_robot_flag;

    struct{

        struct_PID_parameters params;
        SemaphoreHandle_t xMutex_PID_parameters;

    } PID_params;


    struct_eeprom_address eeprom_adresses;

    // QueueHandle_t q_angle_values;

    // struct_angle_values angle_values;

    void system_controller();

    static void system_controller_wrapper(void* arg);


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              .Ki       = EEPROM.readFloat(eeprom_adresses.eeprom_Ki_address
    ),
30              .Kd       = EEPROM.readFloat(eeprom_adresses.eeprom_Kd_address
    ),
31              .setpoint = EEPROM.readFloat(eeprom_adresses.
    eeprom_setpoint_address)
32          },
33
34          .xMutex_PID_parameters = xSemaphoreCreateMutex()
35
36      };
37
38      ESP_LOGI(this->TAG,
39              "eeprom's size = %d\n",
40              EEPROM.length());
41
42      ESP_LOGI(this->TAG,
43              "Kp = %.2f at address = %d\n",
44              this->PID_params.params.Kp,
45              eeprom_adresses.eeprom_Kp_address);
46
47      ESP_LOGI(this->TAG,
48              "Ki = %.2f at address = %d\n",
49              this->PID_params.params.Ki,
50              eeprom_adresses.eeprom_Ki_address);
51
52      ESP_LOGI(this->TAG,
53              "Kd = %.2f at address = %d\n",
54              this->PID_params.params.Kd,
55              eeprom_adresses.eeprom_Kd_address);
56
57      ESP_LOGI(this->TAG,
58              "Setpoint = %.2f at address = %d\n",
59              this->PID_params.params.setpoint,
60              eeprom_adresses.eeprom_setpoint_address);
61
62      ESP_LOGI(this->TAG, "Initialized Center controller successfully\n");
63  }
64
65  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,
       portMAX_DELAY) == pdTRUE){
72
73         this->PID_params.params = params;
74         xSemaphoreGive(this->PID_params.xMutex_PID_parameters);
75
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,
       portMAX_DELAY) == pdTRUE){
83
84         temp = this->PID_params.params;
85         xSemaphoreGive(this->PID_params.xMutex_PID_parameters);
86     }
87
88     return temp;
89 }
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)
       == pdTRUE){
98
99         this->start_robot_flag.flag = value;
100        xSemaphoreGive(this->start_robot_flag.xMutex_flag);
101    }
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,
    portMAX_DELAY) == pdFALSE){
118             ESP_LOGE(this->TAG, "Can't get xMutex_start_robot_flag\n");
119             continue;
120         }
121 
122         mpu_task_state = eTaskGetState(mpu_reader_component->
    get_task_handle());
123 
124         pid_task_state = eTaskGetState(PID_block_component->
    get_task_handle());
125 
126         if(this->start_robot_flag.flag){
127 
128             if(mpu_task_state == eSuspended){
129 
130                 vTaskResume(mpu_reader_component->get_task_handle());
131 
132                 mpu_task_state = eTaskGetState(mpu_reader_component->
    get_task_handle());
133 
134                 if (mpu_task_state == eRunning)
135                 {
136                     ESP_LOGI(this->TAG, "MPU reader's task resumed
    successfully\n");
137                 }
138 
139             }
140 
141             if(pid_task_state == eSuspended){
142 
143                 PID_block_component->set_PID_parameters(this->PID_params.
    params);
144                 PID_block_component->set_prev_time();
145 
146                 vTaskResume(PID_block_component->get_task_handle());
147 
148                 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
    successfully\n");
153                 }
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<
    center_controller*>(arg);
185
186     center_controller_instance->system_controller();
187 }
188
189 void center_controller::run(){
190
191     mpu_reader_component->run();
```

```
192
193     PID_block_component->run();
194
195     if(xTaskCreatePinnedToCore(system_controller_wrapper,"
        system_controller", 2048, this, 9, nullptr, 1)== pdPASS){
196         ESP_LOGI(this->TAG, "Created system_controller task successfully")
        ;
197     }else{
198         ESP_LOGE(this->TAG, "Created system_controller task failed!");
199         while (true){}
200     }
201
202     display_controller_component->run();
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_THRESHOLE 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_THRESHOLE) {
26       temp->buttonPressed = true;
27       temp->lastDebounceTime = currentTime;
28     }
```

```
  }

  portEXIT_CRITICAL_ISR(&mux);
}

void attach_btn_interrupt(Button* btn){

    attachInterruptArg(btn->PIN, handleButtonPress, btn, FALLING);
}

void button_init(void){
  // Initialize buttons and attach them to Interrupt
  init_ctrl_btn(btn_OK);
  init_ctrl_btn(btn_ESC);
  init_ctrl_btn(btn_UP);
  init_ctrl_btn(btn_DOWN);

  attach_btn_interrupt(&btn_OK);
  attach_btn_interrupt(&btn_ESC);
  attach_btn_interrupt(&btn_UP);
  attach_btn_interrupt(&btn_DOWN);
}
```

Program 6.10: button_reader.cpp

6.6 Display Controller

6.6.1 display_controller.hpp

```
#pragma once
#ifndef OLED_DISPLAY_HPP
#define OLED_DISPLAY_HPP

#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Arduino.h>

#include "button_reader.hpp"
#include "global.hpp"
#include "UI_tree.hpp"
#include "center_controller.hpp"
#include "PID_block.hpp"

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
```

```
19
20 #define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset
       pin)
21 class display_controller
22 {
23 private:
24     const char* TAG = "DISPLAY CONTROLLER";
25     const uint8_t fps;
26     const uint8_t priority; // draw_menu task's priority
27     const uint8_t scr_width;
28     const uint8_t scr_height;
29
30     Adafruit_SSD1306* display;
31     UI_tree* current_UI;
32     SemaphoreHandle_t  xMutex_menu_curr_opt;
33     struct_PID_parameters temp_PID_params;
34
35     void create_all_ui(void);
36
37     void draw_menu();
38     static void draw_menu_wrapper(void* arg);
39
40     void get_btn_OK();
41     static void get_btn_OK_wrapper(void* arg);
42
43     void get_btn_ESC();
44     static void get_btn_ESC_wrapper(void* arg);
45
46     void get_btn_UP();
47     static void get_btn_UP_wrapper(void* arg);
48
49     void get_btn_DOWN();
50     static void get_btn_DOWN_wrapper(void* arg);
51
52 public:
53     display_controller( uint8_t fps = 30,
54                         uint8_t priority = 5,
55                         uint8_t width = SCREEN_WIDTH,
56                         uint8_t height = SCREEN_HEIGHT);
57     ~display_controller();
58
59     Adafruit_SSD1306* get_display();
60
61     UI_tree* get_current_ui();
62
63     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] ==
     MENU_START_N_PLOT_OPTION)
41       {
42         display_controller_component->get_display()->setTextColor(BLACK,
     WHITE);
43       }else{
44         display_controller_component->get_display()->setTextColor(WHITE,
     BLACK);
45       }
46       display_controller_component->get_display()->println(F("Start and
     Plot"));
47
48       if (display_controller_component->get_current_ui()->args[0] ==
     MENU_PID_VALUES_OPTION)
49       {
50         display_controller_component->get_display()->setTextColor(BLACK,
     WHITE);
51       }else{
52         display_controller_component->get_display()->setTextColor(WHITE,
     BLACK);
53       }
54       display_controller_component->get_display()->println(F("PID's values
     "));
55     }
56
57     void btnUP_func(){
58
59       ESP_LOGI(title, "In menu_ui_btnUP_func, display_controller_component
     ->get_current_ui()->args[0] = %d\n", display_controller_component->
     get_current_ui()->args[0]);
60       if(display_controller_component->get_current_ui()->args[0] <=
     MENU_MIN_OPTION - 1) {
61         display_controller_component->get_current_ui()->args[0] =
     MENU_MAX_OPTION - 1;
62       }else{
63         --display_controller_component->get_current_ui()->args[0];
```

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

```
101         UI_tree* previous_ui = display_controller_component->
    get_current_ui()->prev_UI;
102 
103         display_controller_component->set_current_ui(previous_ui);
104       }
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(), &
    PID_values, display_controller_component->get_fps()/portTICK_PERIOD_MS)
     == pdTRUE){
120 
121         display_controller_component->get_display()->print("Pitch: ");
122         display_controller_component->get_display()->print(PID_values.
    filted_pitch*RAD_TO_DEG);
123         display_controller_component->get_display()->println(" deg");
124         // display_controller_component->get_display()->print("Gyro Y: ");
125         // display_controller_component->get_display()->print(PID_values.
    gyro_angle_Y);
126         // display_controller_component->get_display()->println(" deg/s");
127         display_controller_component->get_display()->print("Error: ");
128         display_controller_component->get_display()->println(PID_values.
    error);
129         display_controller_component->get_display()->print("Integral: ");
130         display_controller_component->get_display()->println(PID_values.
    integral);
131         display_controller_component->get_display()->print("Derivative: ")
    ;
132         display_controller_component->get_display()->println(PID_values.
    derivative);
133         display_controller_component->get_display()->print("Output: ");
134         display_controller_component->get_display()->println(PID_values.
    output);
135 
136       }else{
```

```
137         ESP_LOGE("DISPLAY CONTROLLER", "Can't recieve angle values!\n");
138         display_controller_component->get_display()->print("Pitch: ");
139         display_controller_component->get_display()->print(PID_values.
    filted_pitch*RAD_TO_DEG);
140         display_controller_component->get_display()->println(" deg");
141         // display_controller_component->get_display()->print("Gyro Y: ");
142         // display_controller_component->get_display()->print(PID_values.
    gyro_angle_Y);
143         // display_controller_component->get_display()->println(" deg/s");
144         display_controller_component->get_display()->print("Error: ");
145         display_controller_component->get_display()->println(PID_values.
    error);
146         display_controller_component->get_display()->print("Integral: ");
147         display_controller_component->get_display()->println(PID_values.
    integral);
148         display_controller_component->get_display()->print("Derivative: ")
    ;
149         display_controller_component->get_display()->println(PID_values.
    derivative);
150         display_controller_component->get_display()->print("Output: ");
151         display_controller_component->get_display()->println(PID_values.
    output);
152 
153       }
154 
155     }
156 
157     void btnESC_func(){
158 
159       if(display_controller_component->get_current_ui()->prev_UI !=
    nullptr){
160 
161 
162         center_controller_component->set_start_robot_flag(false);
163 
164         UI_tree* previous_ui = display_controller_component->
    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] ==
    PID_PARAMETERS_MODIFY_KP)
182       {
183         display_controller_component->get_display()->setTextColor(BLACK,
    WHITE);
184       }else{
185         display_controller_component->get_display()->setTextColor(WHITE,
    BLACK);
186       }
187 
188       display_controller_component->get_display()->printf("Kp = %.2f\n",
    center_controller_component->get_PID_parameters().Kp);
189 
190       if (display_controller_component->get_current_ui()->args[0] ==
    PID_PARAMETERS_MODIFY_KI)
191       {
192         display_controller_component->get_display()->setTextColor(BLACK,
    WHITE);
193       }else{
194         display_controller_component->get_display()->setTextColor(WHITE,
    BLACK);
195       }
196 
197       display_controller_component->get_display()->printf("Ki = %.2f\n",
    center_controller_component->get_PID_parameters().Ki);
198 
199       if (display_controller_component->get_current_ui()->args[0] ==
    PID_PARAMETERS_MODIFY_KD)
200       {
201         display_controller_component->get_display()->setTextColor(BLACK,
    WHITE);
202       }else{
203         display_controller_component->get_display()->setTextColor(WHITE,
    BLACK);
204       }
205 
206       display_controller_component->get_display()->printf("Kd = %.2f\n",
    center_controller_component->get_PID_parameters().Kd);
207 
```

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

```
244       if(display_controller_component->get_current_ui()->next_UI[
    display_controller_component->get_current_ui()->args[0]] != nullptr){
245         Serial.printf("display_controller_component->get_current_ui()->
    next_UI[%d] != nullptr\n", args[0]);
246         UI_tree* next_ui = display_controller_component->get_current_ui()
    ->next_UI[display_controller_component->get_current_ui()->args[0]];
247         display_controller_component->set_current_ui(next_ui);
248       }else{
249         Serial.printf("display_controller_component->get_current_ui()->
    next_UI[%d] == nullptr\n", args[0]);
250       }
251 
252     }
253 
254     void btnESC_func(){
255 
256       if(display_controller_component->get_current_ui()->prev_UI !=
    nullptr){
257         UI_tree* previous_ui = display_controller_component->
    get_current_ui()->prev_UI;
258         display_controller_component->set_current_ui(previous_ui);
259       }
260 
261     }
262   } PID_parameters;
263 
264   struct{
265 
266     String title = "MODIFY KP";
267     uint8_t *args;
268     uint8_t args_len = 0;
269     uint8_t nextUI_len = 0;
270 
271     void display_func(){
272 
273       display_controller_component->get_display()->printf("Kp = %.2f\n",
    display_controller_component->get_temp_PID_params().Kp);
274 
275       display_controller_component->get_display()->println("-UP button:
    increase value");
276       display_controller_component->get_display()->println("-DOWN button:
    increase value");
277       display_controller_component->get_display()->println("-OK button:
    set value");
278 
279     }
```

```
280
281     void btnUP_func(){
282       struct_PID_parameters temp = display_controller_component->
    get_temp_PID_params();
283
284       temp.Kp += 0.1F;
285
286       display_controller_component->set_temp_PID_params(temp);
287     }
288
289     void btnDOWN_func(){
290
291       // 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]);
292       struct_PID_parameters temp = display_controller_component->
    get_temp_PID_params();
293
294       temp.Kp -= 0.1F;
295
296       display_controller_component->set_temp_PID_params(temp);
297
298     }
299
300     void btnOK_func(){
301
302       center_controller_component->set_PID_parameters(
    display_controller_component->get_temp_PID_params());
303
304       EEPROM.writeFloat(center_controller_component->get_eeprom_adresses()
    .eeprom_Kp_address,
305                         display_controller_component->get_temp_PID_params
    ().Kp);
306       EEPROM.commit();
307
308     }
309
310     void btnESC_func(){
311
312       if(display_controller_component->get_current_ui()->prev_UI !=
    nullptr){
313
314         display_controller_component->set_temp_PID_params(
    center_controller_component->get_PID_parameters());
315
316         UI_tree* previous_ui = display_controller_component->
```

```
  get_current_ui()->prev_UI;
317         display_controller_component->set_current_ui(previous_ui);
318       }
319
320     }
321   } modify_kp;
322
323   struct{
324
325     String title = "MODIFY KI";
326     uint8_t *args;
327     uint8_t args_len = 0;
328     uint8_t nextUI_len = 0;
329
330     void display_func(){
331
332       display_controller_component->get_display()->printf("Ki = %.2f\n",
    display_controller_component->get_temp_PID_params().Ki);
333
334       display_controller_component->get_display()->println("-UP button:
    increase value");
335       display_controller_component->get_display()->println("-DOWN button:
    increase value");
336       display_controller_component->get_display()->println("-OK button:
    set value");
337
338     }
339
340     void btnUP_func(){
341       struct_PID_parameters temp = display_controller_component->
    get_temp_PID_params();
342
343       temp.Ki += 0.1F;
344
345       display_controller_component->set_temp_PID_params(temp);
346     }
347
348     void btnDOWN_func(){
349
350       // 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]);
351       struct_PID_parameters temp = display_controller_component->
    get_temp_PID_params();
352
353       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(
    display_controller_component->get_temp_PID_params());
362
363       EEPROM.writeFloat(center_controller_component->get_eeprom_adresses()
    .eeprom_Ki_address,
364                         display_controller_component->get_temp_PID_params
    ().Ki);
365       EEPROM.commit();
366
367     }
368
369     void btnESC_func(){
370
371       if(display_controller_component->get_current_ui()->prev_UI !=
    nullptr){
372
373         display_controller_component->set_temp_PID_params(
    center_controller_component->get_PID_parameters());
374
375         UI_tree* previous_ui = display_controller_component->
    get_current_ui()->prev_UI;
376         display_controller_component->set_current_ui(previous_ui);
377       }
378
379     }
380   } modify_ki;
381
382   struct{
383
384     String title = "MODIFY KD";
385     uint8_t *args;
386     uint8_t args_len = 0;
387     uint8_t nextUI_len = 0;
388
389     void display_func(){
390
391       display_controller_component->get_display()->printf("Kd = %.2f\n",
    display_controller_component->get_temp_PID_params().Kd);
392
```

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

```
429
430       if(display_controller_component->get_current_ui()->prev_UI !=
    nullptr){
431
432         display_controller_component->set_temp_PID_params(
    center_controller_component->get_PID_parameters());
433
434         UI_tree* previous_ui = display_controller_component->
    get_current_ui()->prev_UI;
435         display_controller_component->set_current_ui(previous_ui);
436       }
437
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
    \n", display_controller_component->get_temp_PID_params().setpoint);
451
452       display_controller_component->get_display()->println("-UP button:
    increase value");
453       display_controller_component->get_display()->println("-DOWN button:
    increase value");
454       display_controller_component->get_display()->println("-OK button:
    set value");
455
456     }
457
458     void btnUP_func(){
459       struct_PID_parameters temp = display_controller_component->
    get_temp_PID_params();
460
461       temp.setpoint += 0.1F;
462
463       display_controller_component->set_temp_PID_params(temp);
464     }
465
466     void btnDOWN_func(){
```

```
467
468       // 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]);
469       struct_PID_parameters temp = display_controller_component->
    get_temp_PID_params();
470
471       temp.setpoint -= 0.1F;
472
473       display_controller_component->set_temp_PID_params(temp);
474
475     }
476
477     void btnOK_func(){
478
479       center_controller_component->set_PID_parameters(
    display_controller_component->get_temp_PID_params());
480
481       EEPROM.writeFloat(center_controller_component->get_eeprom_adresses()
    .eeprom_setpoint_address,
482                         display_controller_component->get_temp_PID_params
    ().setpoint);
483       EEPROM.commit();
484
485     }
486
487     void btnESC_func(){
488
489       if(display_controller_component->get_current_ui()->prev_UI !=
    nullptr){
490
491         display_controller_component->set_temp_PID_params(
    center_controller_component->get_PID_parameters());
492
493         UI_tree* previous_ui = display_controller_component->
    get_current_ui()->prev_UI;
494         display_controller_component->set_current_ui(previous_ui);
495       }
496
497     }
498   } modify_setpoint;
499 #pragma endregion
500
501 void display_controller::create_all_ui(void){
502   // Creat MENU UI
503   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,
      nullptr, start_n_plot_ui.args_len,
524                                                   current_UI,
525                                                   start_n_plot_ui.
      nextUI_len,
526                                                   [](){start_n_plot_ui.
      display_func();},
527                                                   [](){},
528                                                   [](){},
529                                                   [](){},
530                                                   [](){start_n_plot_ui.
      btnESC_func();});
531
532     temp_ui->next_UI[0] = start_and_plot_ui_temp;
533   }
534
535   if(current_UI->next_UI[0] != nullptr){
536     ESP_LOGI( this->TAG, "Created Start and plot UI successfully");
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,
      PID_parameters.args, PID_parameters.args_len,
```

```
                                                    current_UI,
                                                    PID_parameters.
  nextUI_len,
                                                    [](){PID_parameters.
  display_func();},
                                                    [](){PID_parameters.
  btnUP_func();},
                                                    [](){PID_parameters.
  btnDOWN_func();},
                                                    [](){PID_parameters.
  btnOK_func();},
                                                    [](){PID_parameters.
  btnESC_func();});
    temp_ui->next_UI[1] = pid_parameters_ui_temp;
  }

  if(current_UI->next_UI[1] != nullptr){
    ESP_LOGI( this->TAG, "Created PID's parameters UI successfully");
  }

  temp_ui = temp_ui->next_UI[1];

  // Create MODIFY KP UI
  if(temp_ui->next_UI[0] == nullptr){

    ESP_LOGI( this->TAG, "Creating Modify Kp UI");

    UI_tree* modify_kp_ui_temp = new UI_tree(modify_kp.title, modify_kp.
  args, modify_kp.args_len,
                                             current_UI->next_UI[1],
                                             modify_kp.nextUI_len,
                                             [](){modify_kp.display_func()
  ;},
                                             [](){modify_kp.btnUP_func();},
                                             [](){modify_kp.btnDOWN_func()
  ;},
                                             [](){modify_kp.btnOK_func();},
                                             [](){modify_kp.btnESC_func()
  ;});
    temp_ui->next_UI[0] = modify_kp_ui_temp;
  }

  if(temp_ui->next_UI[0] != nullptr){
    ESP_LOGI( this->TAG, "Created Modify Kp UI successfully");
  }

```

```
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.
      args, modify_ki.args_len,
587                                              current_UI->next_UI[1],
588                                              modify_ki.nextUI_len,
589                                              [](){modify_ki.display_func()
      ;},
590                                              [](){modify_ki.btnUP_func();},
591                                              [](){modify_ki.btnDOWN_func()
      ;},
592                                              [](){modify_ki.btnOK_func();},
593                                              [](){modify_ki.btnESC_func()
      ;});
594     temp_ui->next_UI[1] = modify_ki_ui_temp;
595   }
596 
597   if(temp_ui->next_UI[1] != nullptr){
598     ESP_LOGI( this->TAG, "Created Modify Ki UI successfully");
599   }
600 
601   // Create MODIFY KD UI
602   if(temp_ui->next_UI[2] == nullptr){
603 
604     ESP_LOGI( this->TAG, "Creating Modify Kd UI");
605 
606     UI_tree* modify_kd_ui_temp = new UI_tree(modify_kd.title, modify_kd.
      args, modify_kd.args_len,
607                                              current_UI->next_UI[1],
608                                              modify_kd.nextUI_len,
609                                              [](){modify_kd.display_func()
      ;},
610                                              [](){modify_kd.btnUP_func();},
611                                              [](){modify_kd.btnDOWN_func()
      ;},
612                                              [](){modify_kd.btnOK_func();},
613                                              [](){modify_kd.btnESC_func()
      ;});
614     temp_ui->next_UI[2] = modify_kd_ui_temp;
615   }
616 
617   if(temp_ui->next_UI[2] != nullptr){
618     ESP_LOGI( this->TAG, "Created Modify Kd UI successfully");
```

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

```
    &Wire, OLED_RESET);

  // Create Mutex variable
  this->xMutex_menu_curr_opt = xSemaphoreCreateMutex();

  create_all_ui();

  // Ckecking display succeeded or not
  while(!( this->display->begin(SSD1306_SWITCHCAPVCC, 0x3C) )) {

    ESP_LOGE(this->TAG, "SSD1306 allocation failed");

    delay(1000);
  }

  button_init();
  // Clear the buffer
  this->display->clearDisplay();

  ESP_LOGI(this->TAG, "Initialized Display controller successfully!");

}

display_controller::~display_controller()
{
}

void display_controller::draw_menu(){

  while (true){

    this->display->clearDisplay();

    this->display->setTextSize(1);             // Normal 1:1 pixel scale

    this->display->setTextColor(WHITE);// Draw white text

    if(xSemaphoreTake(this->xMutex_menu_curr_opt, (this->fps)/
   portTICK_PERIOD_MS) == pdTRUE){

      if((this->current_UI->title.length()*6) >= this->display->width()){

        this->display->setCursor(0,0);

      }else{

```

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

```
745
746       if(xSemaphoreTake(this->xMutex_menu_curr_opt, this->fps/
    portTICK_PERIOD_MS) == pdTRUE){
747
748         this->current_UI->btnOK_func();
749
750         xSemaphoreGive(this->xMutex_menu_curr_opt);
751
752       }else {
753         ESP_LOGI(this->TAG, "Can't take currentUI");
754       }
755
756       btn_OK.buttonPressed = false;
757     }
758     vTaskDelay(200/portTICK_PERIOD_MS);
759   }
760 }
761
762 void display_controller::get_btn_OK_wrapper(void* arg){
763
764   display_controller* display_controller_instance = static_cast<
     display_controller*>(arg);
765   display_controller_instance->get_btn_OK();
766
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/
    portTICK_PERIOD_MS) == pdTRUE){
779
780         this->current_UI->btnESC_func();
781
782         xSemaphoreGive(this->xMutex_menu_curr_opt);
783
784       }else {
785         ESP_LOGI(this->TAG, "Can't take currentUI");
786       }
787       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<
      display_controller*>(arg);
796   display_controller_instance->get_btn_ESC();
797 
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/
      portTICK_PERIOD_MS) == pdTRUE){
809 
810         this->current_UI->btnUP_func();
811 
812         xSemaphoreGive(this->xMutex_menu_curr_opt);
813 
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<
      display_controller*>(arg);
828   display_controller_instance->get_btn_UP();
829 
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/
    portTICK_PERIOD_MS) == pdTRUE){
841
842         this->current_UI->btnDOWN_func();
843
844         xSemaphoreGive(this->xMutex_menu_curr_opt);
845
846       }else {
847         ESP_LOGI(this->TAG, "Can't take currentUI");
848       }
849
850       btn_DOWN.buttonPressed = false;
851     }
852     vTaskDelay(200/portTICK_PERIOD_MS);
853   }
854 }
855
856 void display_controller::get_btn_DOWN_wrapper(void* arg){
857
858   display_controller* display_controller_instance = static_cast<
    display_controller*>(arg);
859   display_controller_instance->get_btn_DOWN();
860
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
      new_value){
888   this->temp_PID_params = new_value;
889 }
890 
891 void display_controller::run(){
892   if(xTaskCreatePinnedToCore( get_btn_OK_wrapper,
893                               "get_btn_OK",
894                               2048,
895                               this,
896                               6,
897                               nullptr,
898                               1) == pdPASS){
899 
900     ESP_LOGI(this->TAG, "Created get_btn_OK task successfully!");
901 
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,
```

```
                                this->priority,
                                nullptr,
                                1) == pdPASS){

    ESP_LOGI(this->TAG, "Created draw_menu task successfully!");

  }else{

    ESP_LOGE(this->TAG, "Created draw_menu task failed!");

    while (true){}

  }
}

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.

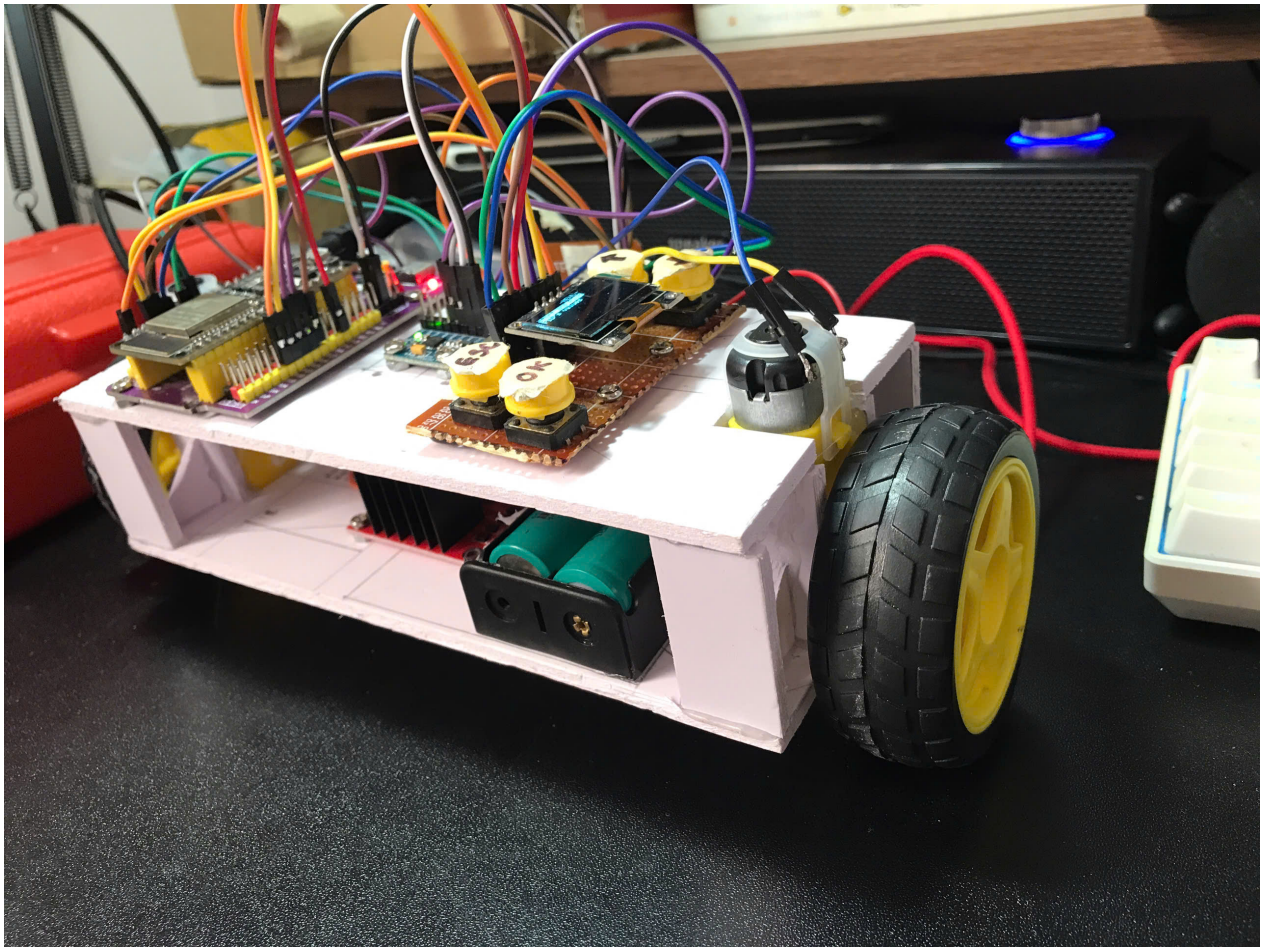

Figure 7.1: Self-Balancing Robot front view

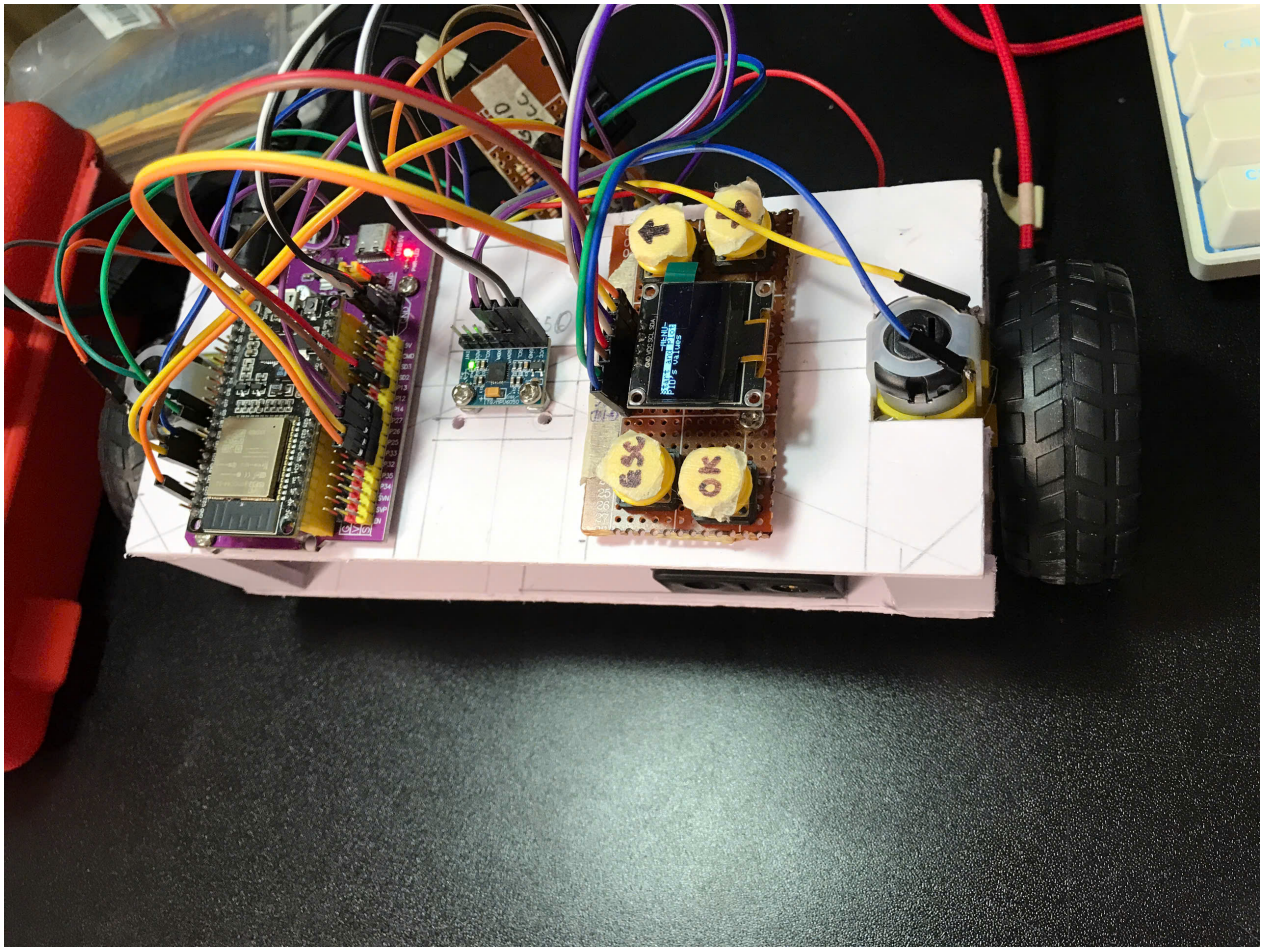

Figure 7.2: Self-Balancing Robot up view

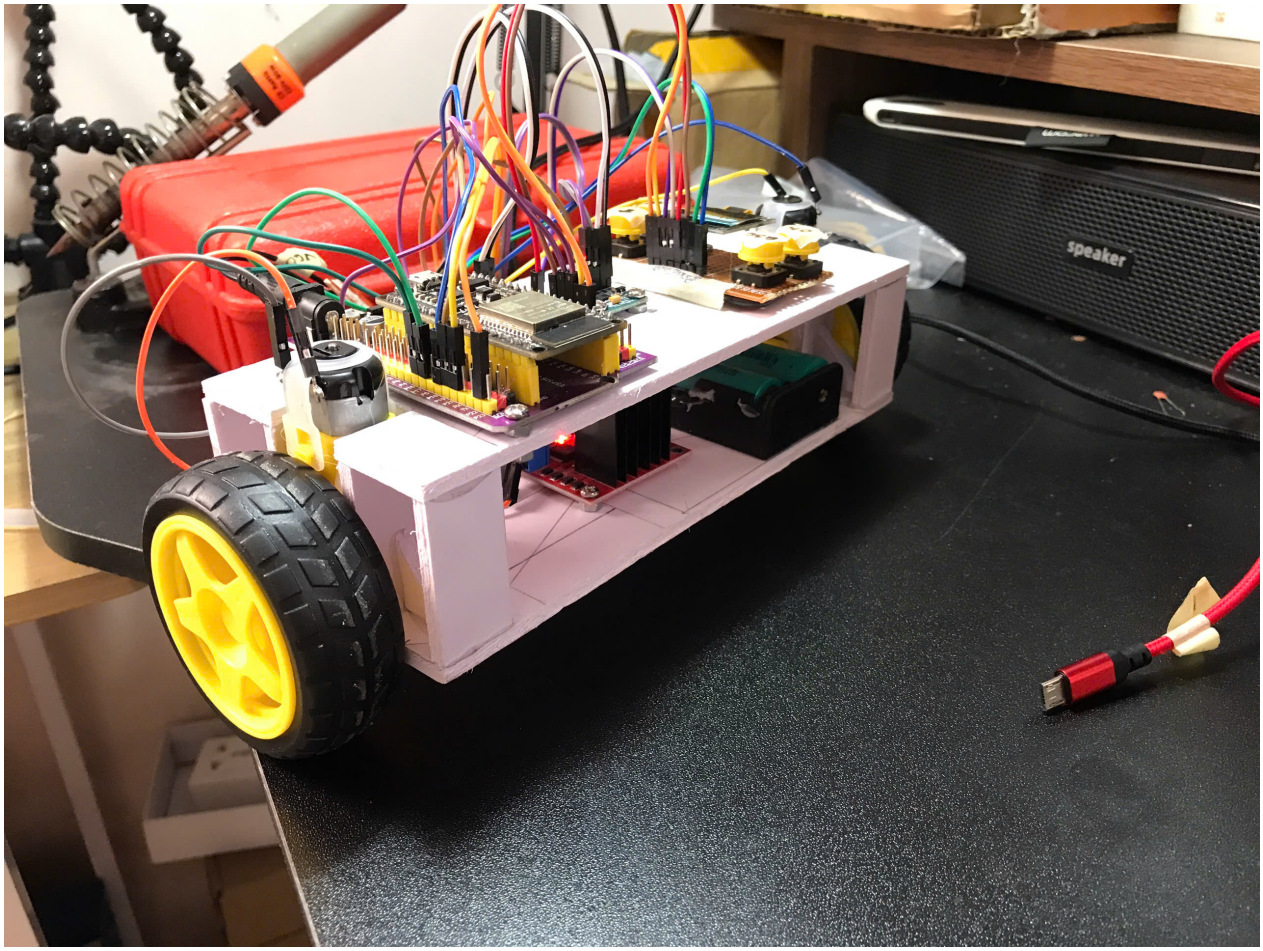

Figure 7.3: Self-Balancing Robot back view

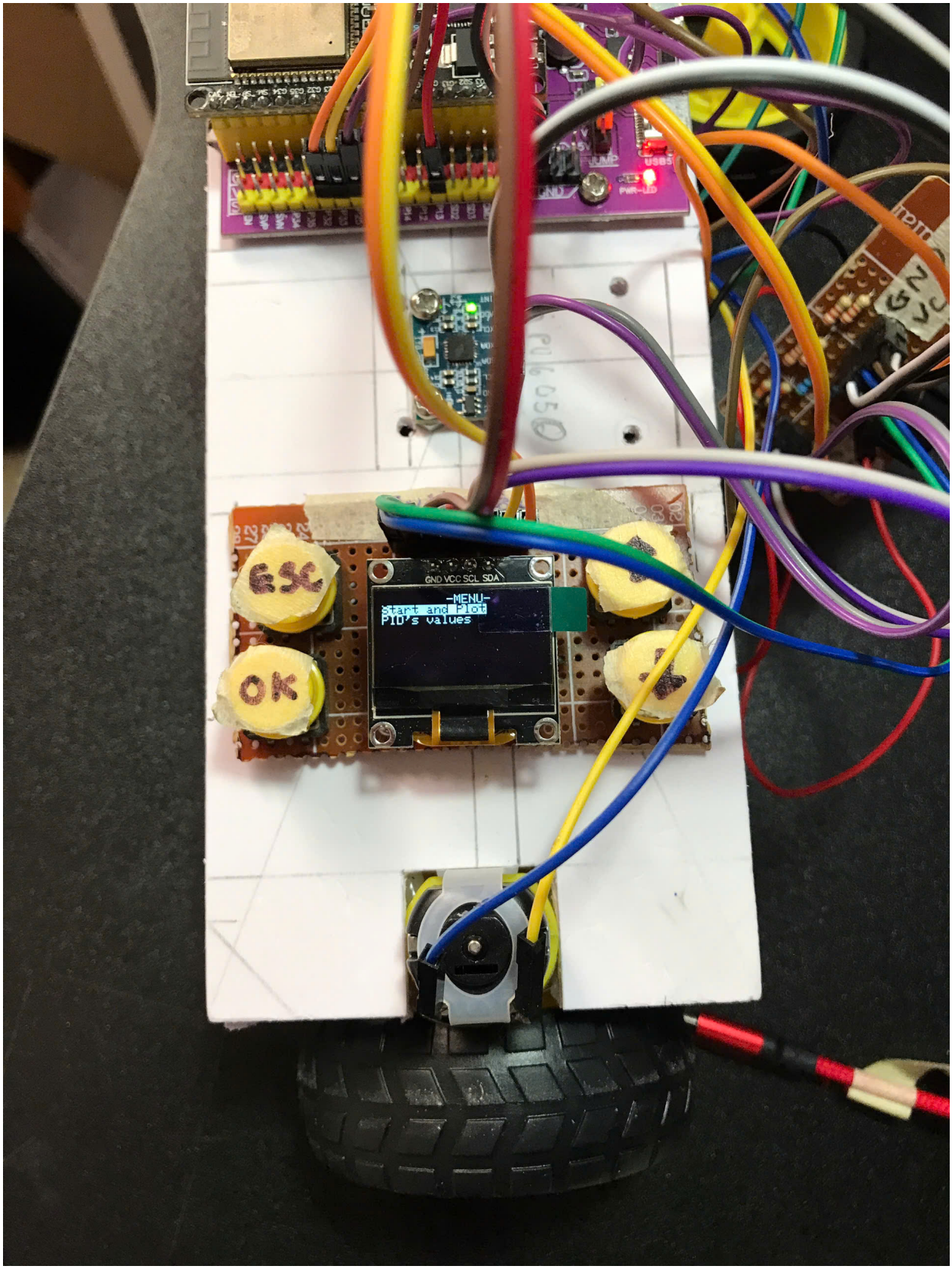

Figure 7.4: Menu UI

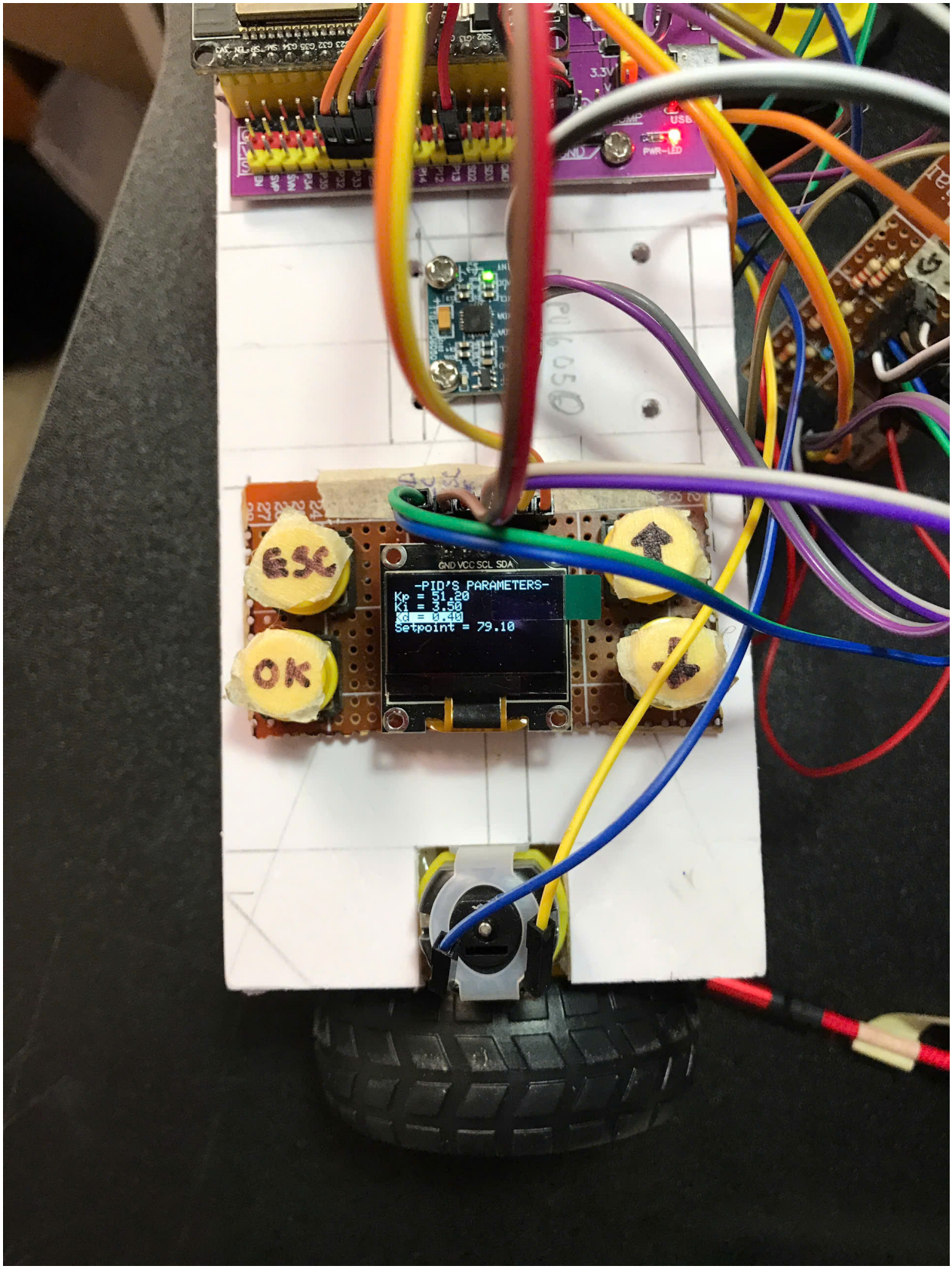

Figure 7.5: PID parameters UI

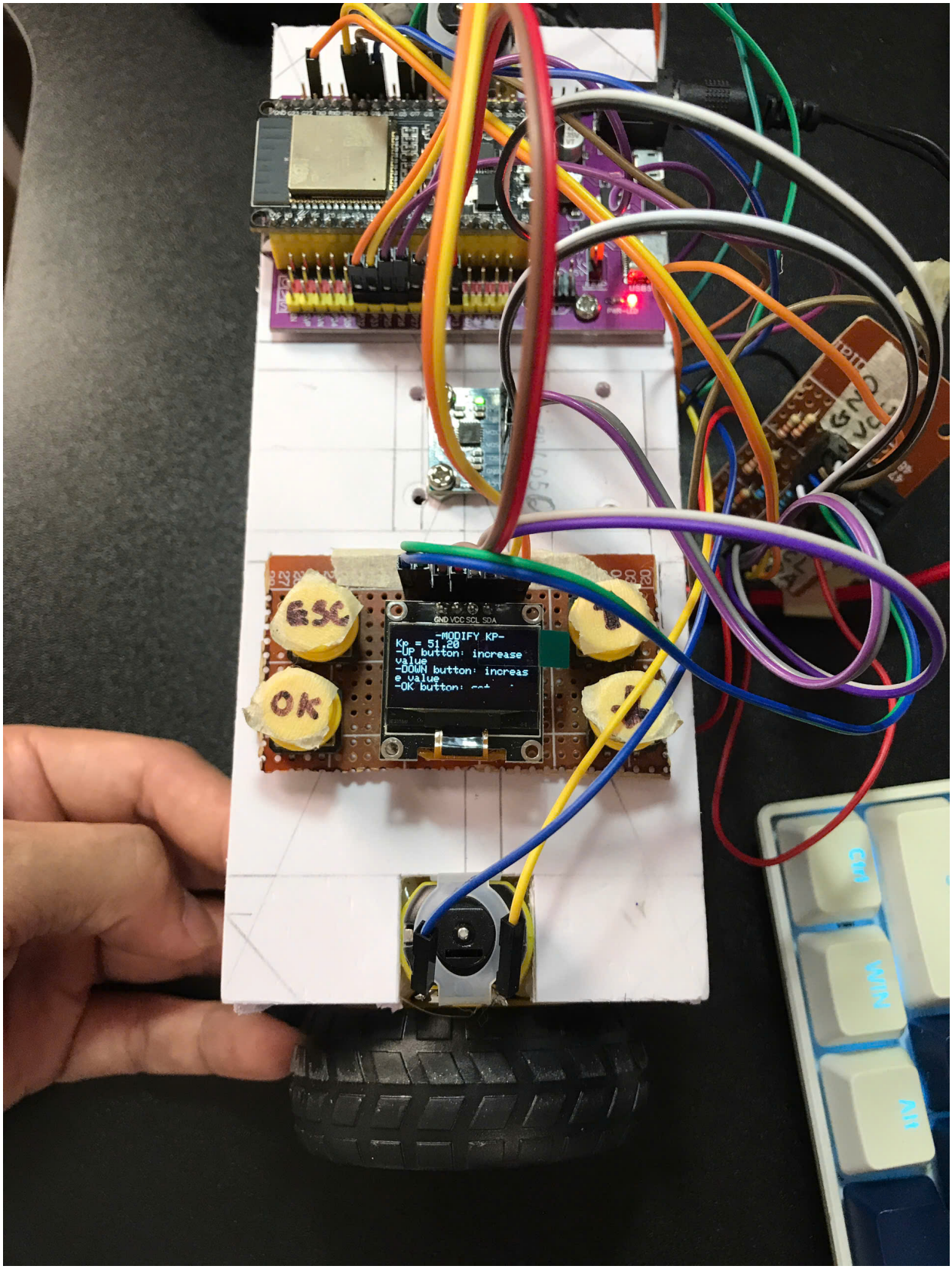

Figure 7.6: Modify Kx UI

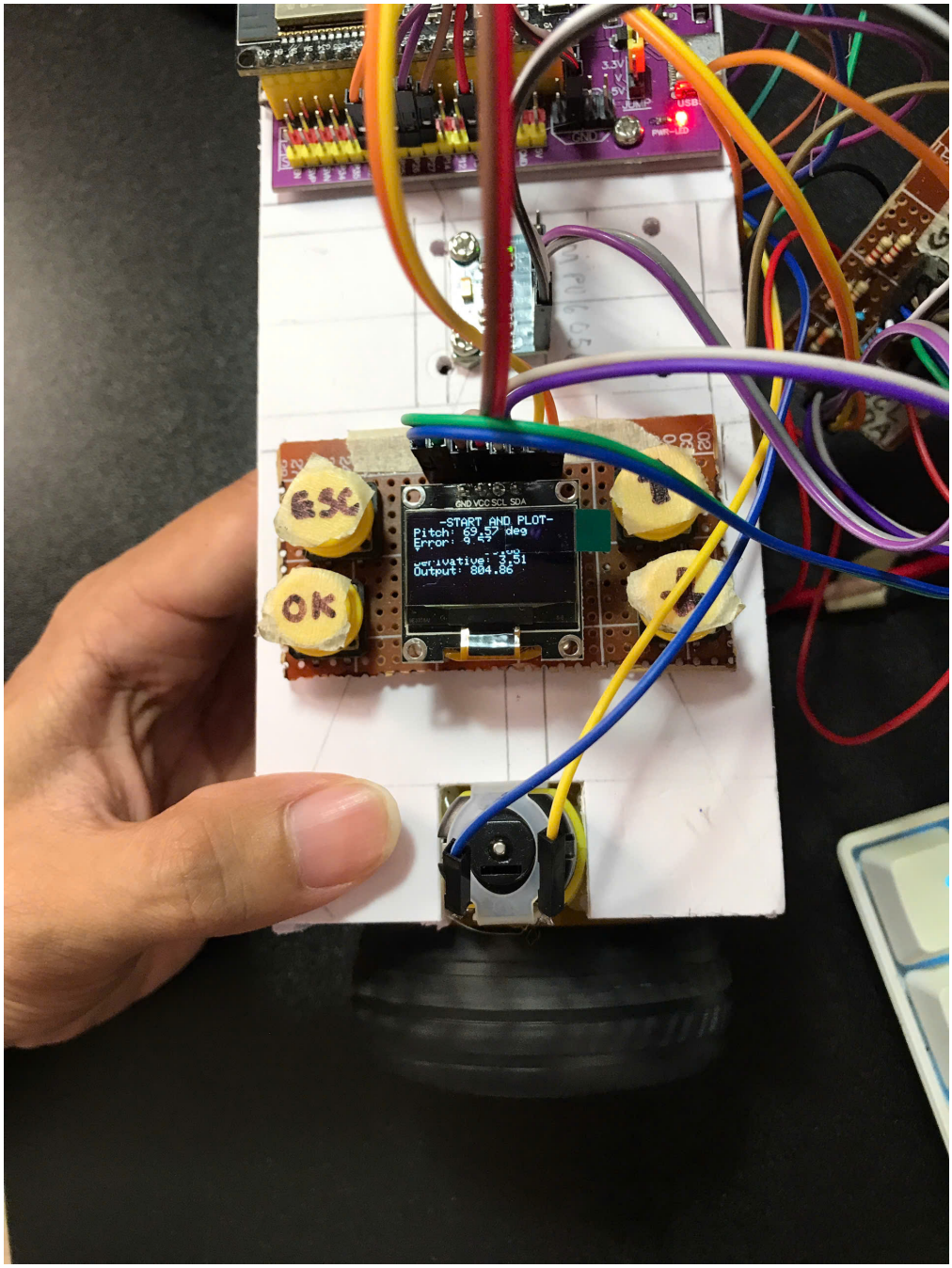

Figure 7.7: Start and plot UI

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

End.