

**MATERIAL OF
SELF BALANCING ROBOT**

**By
GERALDI KOLIN
(TEACHING ASSISTANT)**



**DR. RUSMAN RUSYADI
ALGORITHM, PROGRAMMING, AND DATA STRUCTURE
MECHATRONICS
SWISS GERMAN UNIVERSITY**

The Prominence Tower
Jalan Jalur Sutera Barat No. 15, Alam Sutera
Tangerang, Banten 15143 – Indonesia

Table of Contents

1. INTRODUCTION.....	2
2. HARDWARE	2
2.1. Parts and Components.....	2
2.2. Assembly.....	3
2.3. Wiring	8
2.4. Optional Changes.....	9
3. SOFTWARE.....	10
3.1. Libraries	10
3.2. ESP32 Code	12
3.2.1. WiFi and TCP Initialisation	12
3.2.2. MPU6050 Initialisation.....	12
3.2.3. PID Setup	14
3.2.4. Stability Control.....	14
3.2.5. Data Sending	14

1. INTRODUCTION

For semester two students in the faculty of Mechatronics, the material that is given is to learn the inner workings of programming and data structures. This includes the using programming languages, mainly C++, in combination of IDEs such as Arduino IDE and Qt Creator as the basis of writing codes. Previously, for the study of algorithm, programming, and data structure, the material was the line follower in combination with Qt as the controller integrated with QFirmata. Starting this semester (Odd 2024), the material will be the SBR (Self-Balancing Robot) which uses an ESP32 DevkitC V4, integrated with Qt Creator to make a UI as a monitoring tool. The communication protocol used in this material will be the Modbus TCP, where communication is only used to read the data sensors of the microcontroller (ESP32). Further details, such as the library, code, and functions, will be explained further in this documentation.

2. HARDWARE

This section will cover all the necessary details related to the hardware used as the learning material. Many of the parts are outsourced, with some self-made components to accommodate incompatible parts.

2.1. Parts and Components

The components that are used for the Self-Balancing Robot are listed as below,

Part Name	Amount
SBR Frame	1
Motor Mount	2
Motor Spacer	2
Wheels	2
12VDC Motor	2
ESP32 DevkitC V4	1
ESP32 Expansion Board	1
L298N	1
MPU6050	1
Rocker Switch	1
Battery Holder	1
Battery 18650	2
FtF Jumper	9
MtF Jumper	6
DC Power Jack	1
M3x30 Philips	4
M3 Nut	12
M3x5 Philips	8
M3x8 Spacer	8

Table 2.1.1. Parts and Components.

The parts listed are available on the online market to buy, however the list does not include the necessary tools to build the robot as a whole. This might include things such as soldering iron, heat shrinks,



suckers, etc. It is important to note that the *Motor Mount* and *Motor Spacer* are custom made, which means they need to be ordered from the SGU Workshop, or printed by a third-party service. For better clarification, below will be the figure for some of the components,

Figure 2.1.1. 12VDC Motor and Wheel.

2.2. Assembly

For students, the assembly of the robot itself would be simple, as it does not require any further modification towards frame. Any modifications such as holes should have been made by the teaching assistant. Below are the steps to create a fully functioning SBR,

1. Fasten 4 spacers for the ESP32 Shield on the top rack of the frame with the M3 Nuts.

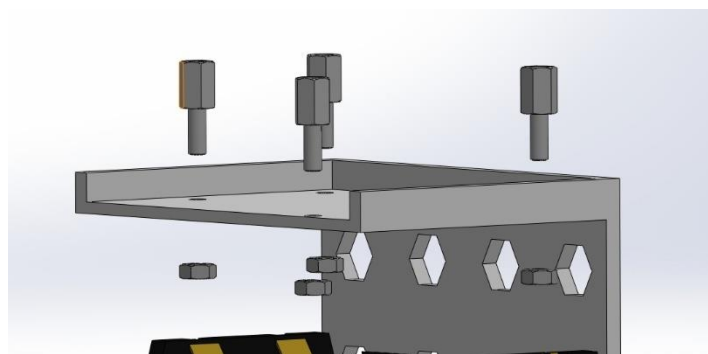


Figure 2.2.1. Step 1.

2. Screw on the ESP32 Shield on the spacers with the M3x5 Philips.

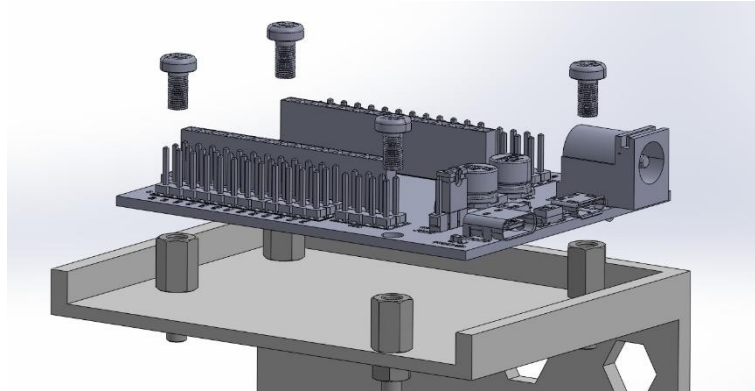


Figure 2.2.2. Step 2.

3. Fasten the spacers on the MPU6050 and the L298N with M3 Nuts, with the hole side of the spacer facing down.

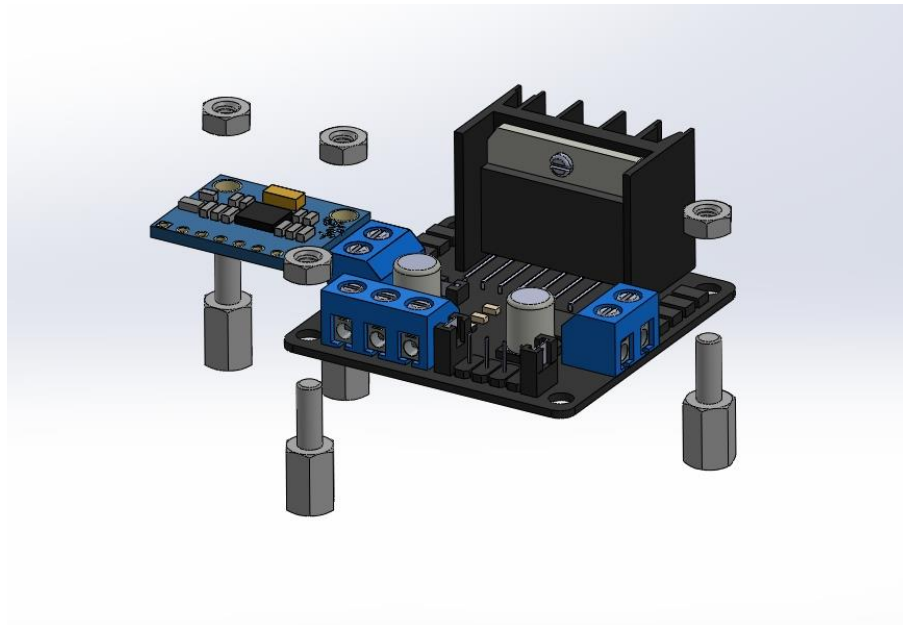


Figure 2.2.3. Step 3.

4. Solder the wire from the battery holder on the rocker switch (middle slot). Then solder the wires from the power jack (tip) where the positive (red) attaches to the rocker switch (side slot). Also attach a male jumper to the rocker switch (side slot), for the power cable on the L298N.
5. Solder the ground wires accordingly, where the ground (black) attaches to the power jack (sleeve).

6. Glue the rocker switch according to the figure.

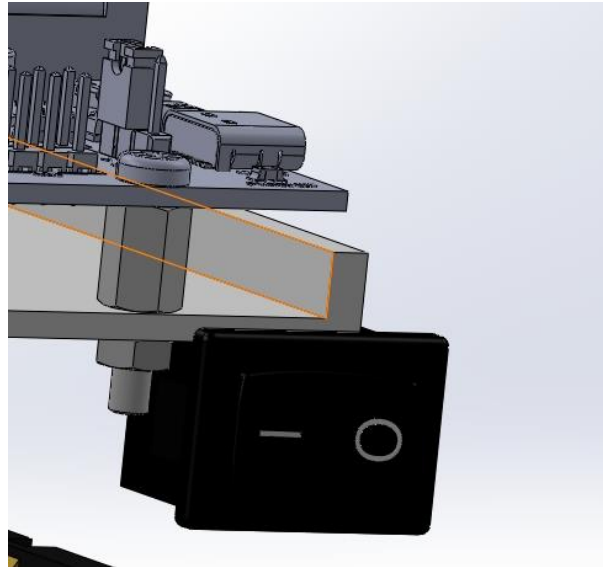


Figure 2.2.4. Step 6.

7. Solder Male tipped jumper wires to the motors, and fasten them on the L298N along with the male power jumper from the switch.
8. Screw the MPU6050 and L298N on the bottom rack with the Philips M3x5 according to the figure.

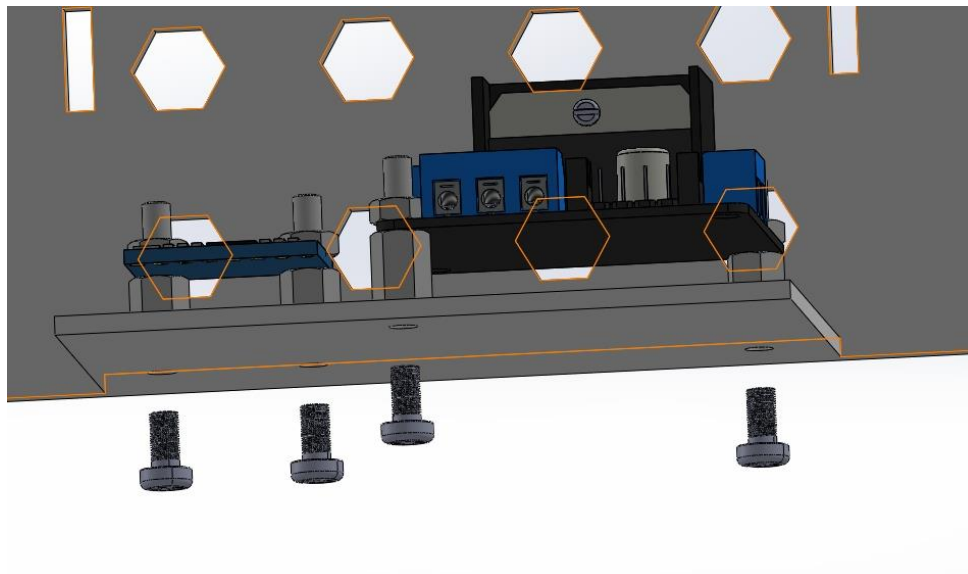


Figure 2.2.5. Step 8.

9. Slide the motor mount through the slot on the side of the frame.

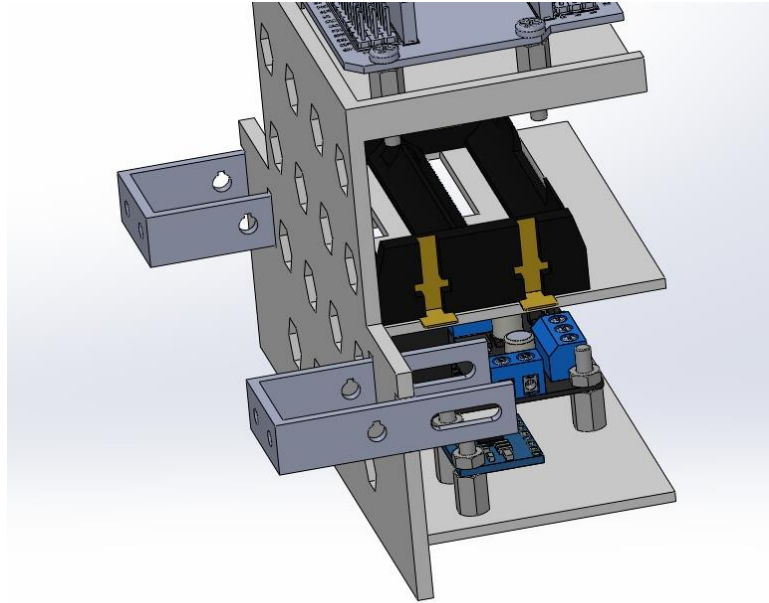


Figure 2.2.6. Step 9.

10. Attach the motor on the motor mount with the motor spacer place between the frame and the motor. Use the Philips M3x30 and the M3 nuts to fasten the motor.

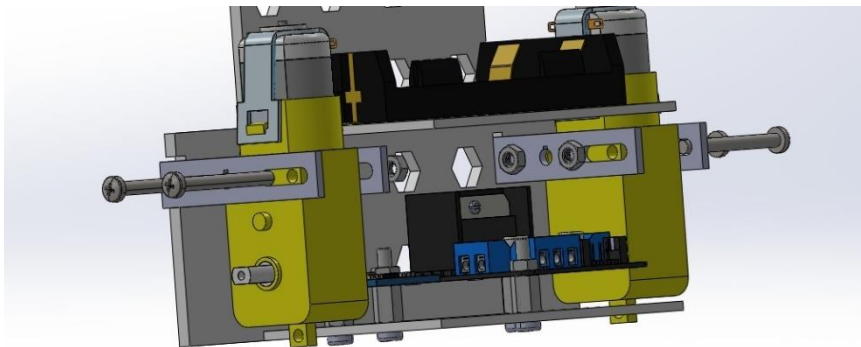


Figure 2.2.7. Step 10.

11. Place the wheels on the motor, while holding the motor.

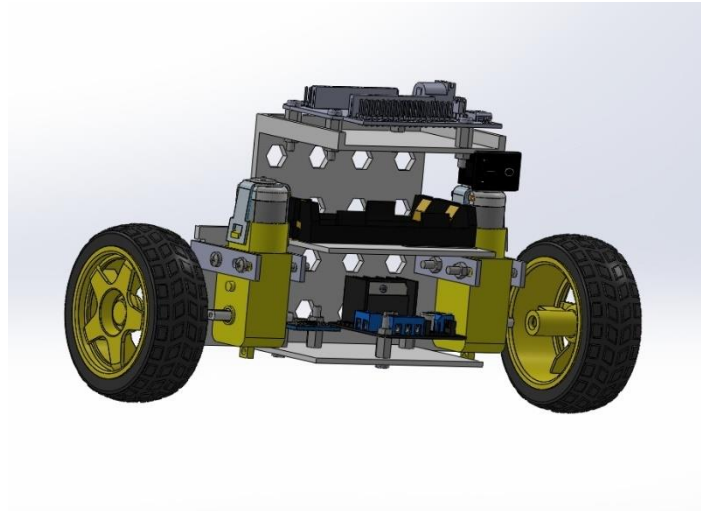


Figure 2.2.8. Step 11.

12. Place the batteries on the battery holder and plug in the power jack to the shield.
13. ***Important*** Cut the CMD Pin of the ESP32 and then place the ESP32 on the shield.

Below would be how the robot would look like fully assembled.

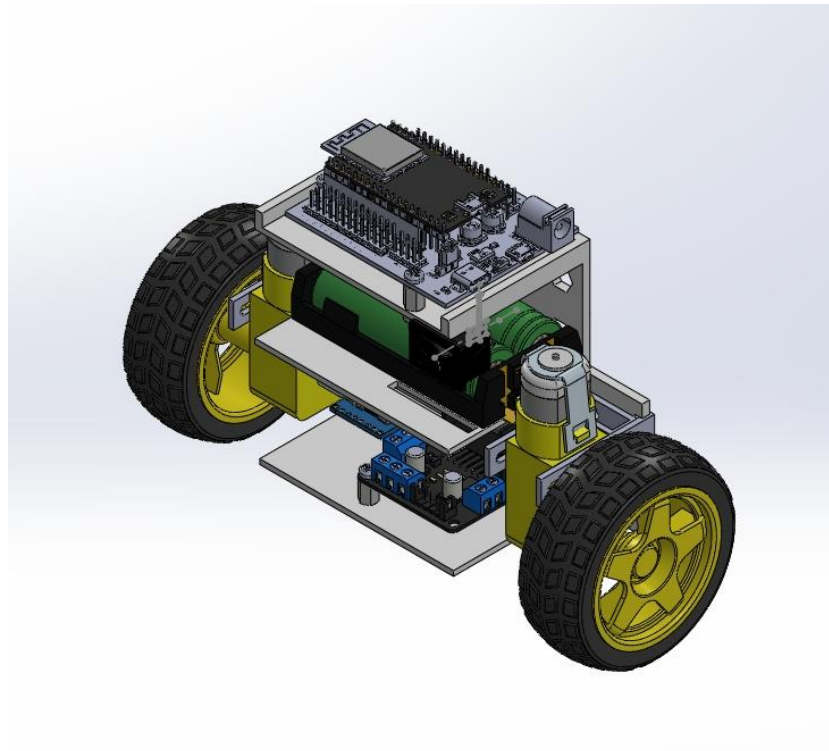


Figure 2.2.9. Final Assembly.

2.3. Wiring

The wiring of the robot itself cannot be rearranged as needed, however if done so the defines on the program needs to also be changed so that the microcontroller can recognise the new existing pins. The schematic symbol of the SBR is as below,

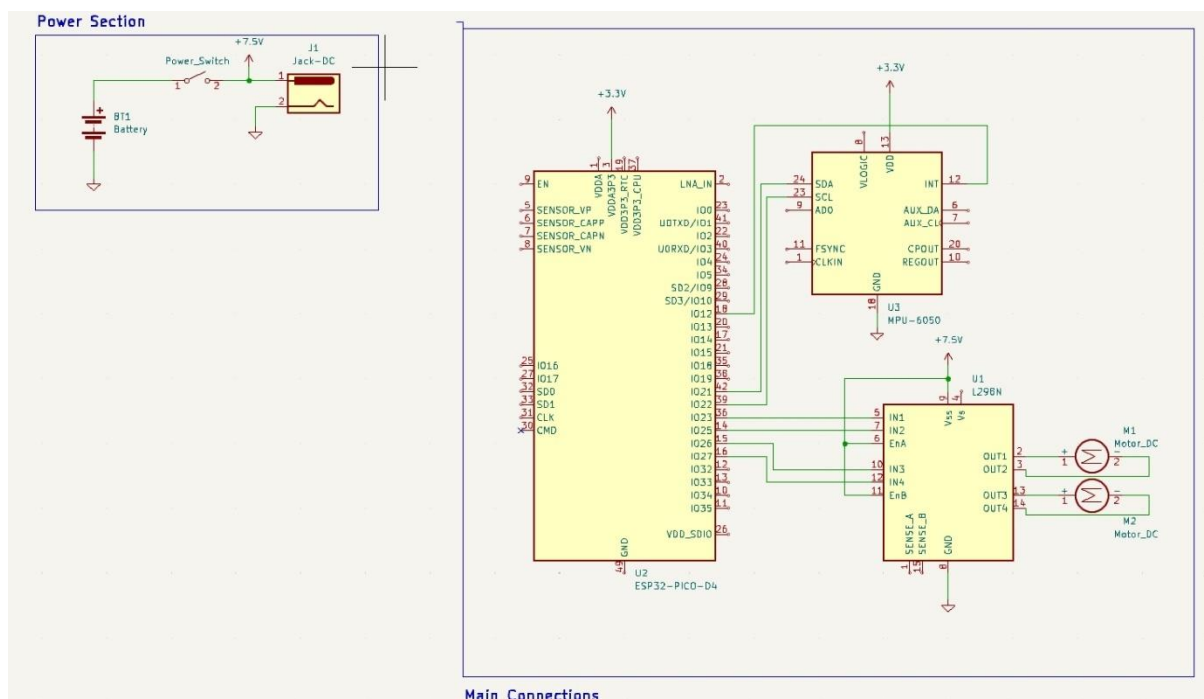


Figure 2.3.1. Wiring Diagram

Since the MPU needs to be able to communicate via I2C, the pins of the SDA need to be connected to the GPIO 21 and SCL needs to be connected to GPIO 22. Those pins are usually used for serial communication between the board and other devices. If more detail is needed, the data sheet can be viewed as it is publicly opened on the internet. Orientation of the motors wires also needs to be noted as it will correspond to the direction of the motor.

The power pin for the MPU6050 needs to be connected to the board power source which can be the 3.3V or the 5V output (3.3VDC is recommended). The ground pin of both the MPU6050 and the L298N needs to be connected to the board as well. This is to achieve a common ground among the circuit. The L298N needs to be powered with a higher voltage, thus it is connected to the 7.5V directly from the battery via the rocker switch. Since the EN Pins on the L298N is already connected (Jumped on the board), it is short circuited with 7.5V making it not require any cables from the board.

2.4. Optional Changes

If you want a more stable robot, there are some changes that can be done to the robot, but the components need to be bought individually. This can include the motor of the robot, which is rated at 12V but fed with 7.5 Volts. A smaller motor with more torque and lower speed (Rated at 3-6V) can be used to drive the motor as balancing doesn't require to high of a speed.

The frame itself can also be changed with a more visible centre of mass, which would allow the robot to balance more reliably. The mounting of the motor can also be changed if needed, however this requires more self-design to accommodate any changes.

3. SOFTWARE

For the software side of the robot, Arduino IDE is used to upload the code to the ESP32. The monitoring UI uses Qt Creator using Modbus TCP to communicate the data via WiFi router. If required, the ESP32 board library needs to be initially installed through the IDE. The IDE used is the latest IDE (version 2.3.2), and the board library is named *esp32(2.0.11)* by Espressif. After installing the, you can choose the board from the tools tab (Tools->Board->DOIT ESP32 DEVKIT V1). No necessary changes need to be made on the board settings.

There are several libraries used in this program, which can be installed directly through the Arduino IDE. The required libraries are as follows,

- MPU6050 by Electronic Cats
- PID_v2 by Brett Beaurgard
- modbus-esp8266 by Andre Sarmiento Barbosa

Along with the libraries, there should also be custom classes that are used to be included in the program. They can be include using the `#include "(name).h"` code. To change variables that manipulate the robot, such as pins, constants, and offsets, you can do so from the *defines.h* header (you can edit through the IDE or through Microsoft Studios). It is also important that you install the CH340 driver to communicate the IDE to the ESP through the serial port (USB port).

To explain the code briefly, the code initially setups the required functions and settings. This includes the I2C communication to the MPU, the stable position, network setup, and PID setup. The stable position setup is used by sampling the current position 100 times. During this, the robot needs to be at its stable position for at least 2 seconds. Then it runs the main looping function where it samples the pitch (tilting angle) every refresh rate that the ESP32 can handle. It will then run it through the PID function while outputting the required PWM to the motor driver to correct the error of the angle. After that, it will send the value that was recorded from sensor to the holding register. Any data can be sent to the holding register of the Modbus TCP, however please keep in mind that it the code for reading the holding register from the Qt also needs to be modified.

3.1. Libraries

All the required libraries and headers are included as below,

```
#include <Wire.h>
#include <WiFi.h>
#include <ArduinoOTA.h>
#include <Arduino.h>

#include <PID_v2.h>

#include "I2Cdev.h"
#include <MPU6050_6Axis_MotionApps20.h>
#include <ModbusIP_ESP8266.h>

#include "motor_driver.h"
#include "defines.h"
#include "globals.h"

#include <stdio.h>
```

```
#include "esp_types.h"
#include "soc/timer_group_struct.h"
#include "driver/periph_ctrl.h"
#include "driver/timer.h"
#include "driver/ledc.h"
#include "esp32-hal-ledc.h"
```

Please note that you do not need to install libraries that have the quotation marks, as they are already pre-built in the library. However, the *globals.h*, *defines.h*, *motor-driver.h* can be downloaded from the .ZIP file in GClass. Run the .INO first which will create a save folder, then place the classes in the save folder.








	Processing	10/02/2024 11:46	File folder	
	defines	23/02/2024 21:02	C Header Source F...	1 KB
	globals	23/02/2024 21:02	C++ Source File	1 KB
	globals	23/02/2024 21:02	C Header Source F...	1 KB
	main	26/02/2024 15:56	INO File	6 KB
	motor_driver	27/02/2024 18:00	C++ Source File	1 KB
	motor_driver	23/02/2024 21:02	C Header Source F...	1 KB

Figure 3.1.1. Folder Location.

3.2. ESP32 Code

The code that runs the ESP32 itself is on the file names *main.ino* where it is opened with the Arduino IDE. The explanation for this code will be explained in the file, or through the class sessions. The most important codes will be the following,

3.2.1. WiFi and TCP Initialisation

```
Serial.println("Starting WiFi Initialization");
WiFi.begin(WIFI_SSID, WIFI_PASS);
while(WiFi.status() != WL_CONNECTED)
{
    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.println("WiFi Connected to");
Serial.println("IP Address  : ");
Serial.print(WiFi.localIP()); // To connect to the ModbusTCP server, please
write this IP address.

//Modbus TCP Server Initialization
regData.server();
for( int i = 0 ; i <= REG_AMOUNT ; i++)
{
    regData.addHreg(i);
}
```

Tries to connect to the local router, keep in mind that the SSID and Password need to be changed accordingly.

Initialization of the TCP server on the ESP32.

3.2.2. MPU6050 Initialisation

```
Serial.println("");
Serial.println("Initializing I2C devices...");
mpu.initialize();
pinMode(INTERRUPT_PIN, INPUT);
Serial.println(F("Testing device connections..."));
Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") :
F("MPU6050 connection failed"));

Serial.println(F("Initializng DMP..."));
devStatus = mpu.dmpInitialize();

mpu.setXGyroOffset(220);
mpu.setYGyroOffset(76);
mpu.setZGyroOffset(-85);
mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

if(devStatus == 0)
{
    mpu.CalibrateAccel(6);
    mpu.CalibrateGyro(6);
}
```

Enable I2C to the MPU6050

Connects to the MPU6050

Initialize the offsets

Calibrates the MPU6050

```
mpu.PrintActiveOffsets();

Serial.println(F("Enabling DMP..."));
mpu.setDMPEnabled(true);

// enable Arduino interrupt detection
Serial.print(F("Enabling interrupt detection (Arduino external interrupt
"));
Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
Serial.println(F(")..."));
attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady,
RISING);
mpuIntStatus = mpu.getIntStatus();

Serial.println(F("DMP ready! Waiting for first interrupt..."));
dmpReady = true;
packetSize = mpu.dmpGetFIFOPacketSize();

}
else
{
    Serial.print(F("DMP Initialization failed (code "));
    Serial.print(devStatus);
    Serial.println(F(")"));
}

Serial.println("Sampling MPU for Stable Position...");
currentSample = 0;
while( currentSample <= INIT_SAMPLE_SIZE )
{
    if(!dmpReady) return;
    if(mpu.dmpGetCurrentFIFOPacket(fifoBuffer))
    {
        mpu.dmpGetQuaternion(&q, fifoBuffer);
        mpu.dmpGetGravity(&gravity, &q);
        mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
        ++currentSample;
        delay(5);
        angleSample = angleSample + (ypr[1] * 180/M_PI);
    }
}
targetAngle = (angleSample / INIT_SAMPLE_SIZE) - CENTER_OF_MASS_OFFSET;

Serial.println("Target Angle = ");
Serial.print(targetAngle);
```

Attach the interrupt pin to the ESP32.

Samples the current pitch position 100 times.

3.2.3. PID Setup

```
stabilityControl.SetOutputLimits(-250, 250);  
stabilityControl.SetMode(PID::Automatic);  
stabilityControl.SetSampleTime(10);  
stabilityControl.Start(angleSample, 0, targetAngle);
```

PID Initialization

3.2.4. Stability Control

```
double targetSpeed = stabilityControl.Run(angleSample); // Raw PID  
targetSpeed = -targetSpeed;  
double finalSpeed = targetSpeed;  
finalSpeed = constrain(finalSpeed, -250, 250);  
  
setMotorSpeed(finalSpeed);  
  
if(!dmpReady) return;  
if(mpu.dmpGetCurrentFIFOPacket(fifoBuffer))  
{  
    mpu.dmpGetQuaternion(&q, fifoBuffer);  
    mpu.dmpGetGravity(&gravity, &q);  
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);  
  
    angleSample = (ypr[1] * 180/M_PI);  
}
```

The angle is inputted into the PID function which returns the target speed for the motors.

Set the motors to the target speed.

Gets the current angle of the MPU6050.

3.2.5. Data Sending

```
regData.Hreg(0, angleSample);  
regData.task();
```

Sends the angle to the register number.

Best of Luck <3
-Author