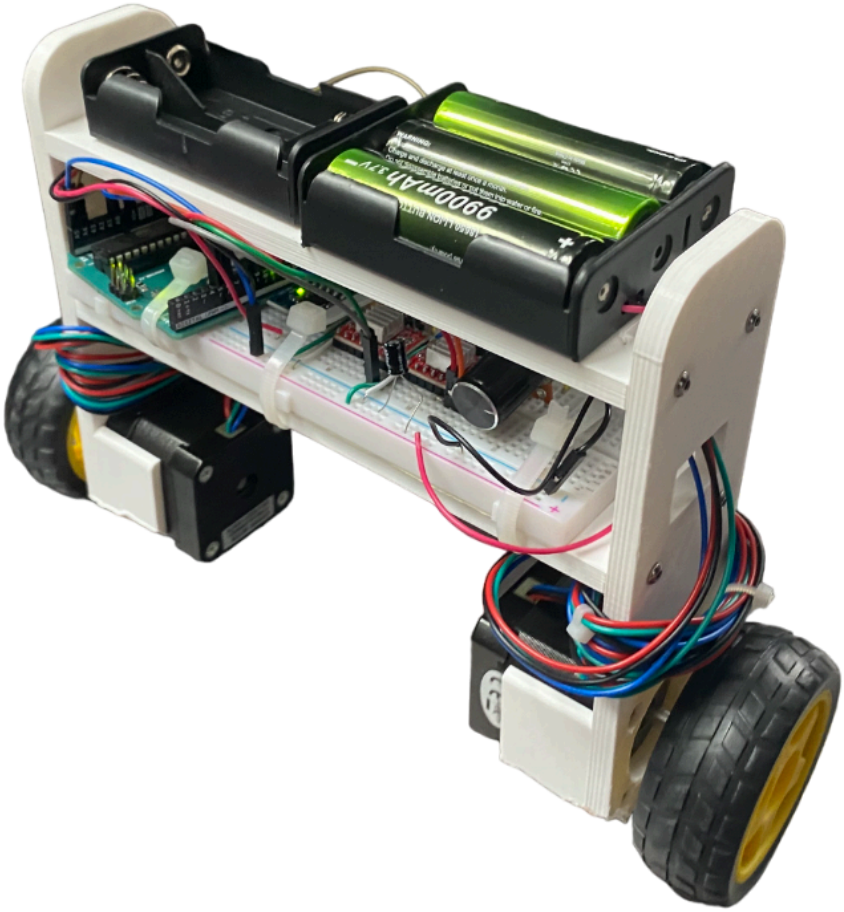
Self Balancing Robot
Using PID Closed Loop Control

Eliott Lacomme Ruiz



I am very happy with how this robot turned out! It is officially my first successful robot that I designed from scratch! - *ERLR 7/12/2024*

Introduction

The goal of this project was to dive into closed-loop control with a PID controller and learn to use a stepper motor. The reason for this was to understand how to use these components for next year's NSL competition. I also wanted to have a successful robot that I designed, printed, made the electronics and code for to have the complete mechatronics experience. I wanted the process of building the robot to be as professional as possible, so I created a spreadsheet with links to websites and YouTube videos, including annotations and general notes. I planned the sequence of tests to build multiple components and ensure smooth integration according to a schedule I made for myself. Additionally, I made a bill of materials and logged my hours as accurately as possible.

This was an excellent CAD, electronics, and introduction to control loop project, and I am very happy with the result.

Design

When designing the robot, I wanted it to have a high center of gravity and be wide enough to fit a breadboard, as I did not have any protoboards for soldering components. It is crucial that the center of gravity is above the pivot point of the robot (the wheels) because it is more stable in this configuration. This principle is similar to how it is easier to balance an upside-down broom on your finger compared to a pencil.

I aimed to make the design as simple as possible, requiring the least amount of mechanical parts. The robot consists of three main parts: the stepper motor mount, the upper deck, and the lower deck. Figure 1 shows the full design of the robot.

The stepper motor mount, as shown in Figure 2 and 3, is 6.5”x2”x ¼” 3D printed PLA mount with a 25% infill. This mount holds the stepper motors with four holes for 6mm M2 screws to secure the motors tightly. Since the stepper motors are heavy, I extruded a shell around them for support to prevent them from vibrating out of the mount. The mount also includes slots on the side tall enough to fit a USB cable to program the microcontroller when it is on the breadboard.

The upper deck, as shown in Figure 4, is a 7”x2”x ¼” 3D printed PLA mount with 60% infill to give it some weight. This deck holds the batteries of the robot and the slots on either side of the deck allow for the power and ground wires to be run through to be connected to the breadboard. On both sides of the deck, there are two holes for M2 heat threaded inserts to allow for a sleek and uniform design, as seen in Figure 5. Four 12mm M2 screws are used to hold the deck to the stepper mount.

The lower deck, as shown in Figure 6, is a 7”x2”x ¼” 3D printed PLA mount with 20% infill. To make it as lightweight as possible, I cut out long sections of the lower deck but left diagonal supports for structural stability. This section will hold the breadboard with all of the electronics and is secured with zip ties. Since the breadboard is a long solid piece, it does not need much support from the lower deck to stay in place. The slots on the far side are there to allow the stepper motor wires to have easy access to the electronics. On both sides of the deck, there are two holes for M2 heat threaded inserts to allow for a sleek and uniform design. Four 12mm M2 screws are used to hold the deck to the stepper mount. Full chassis assembly is shown on Figure 7.

One design feature that was overlooked was the thickness of the extruded shell of the stepper motor compared to the wheel diameter. I had previously measured out the shell thickness so it would not interfere with the wheels but I did not account for the deformation of the wheels under a heavy load. As a result, I had to burn off the edges of the shell on the stepper motors so the edges would not touch the ground when the robot was leaning. The result is shown in Figure 8.

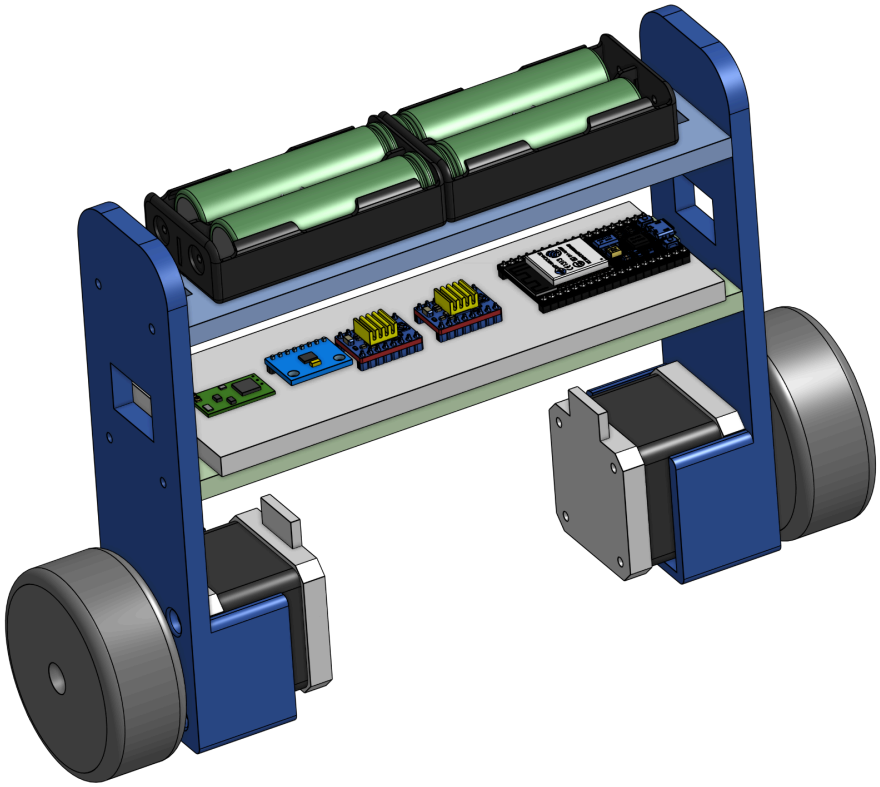


Figure 1: Full assembly of the robot

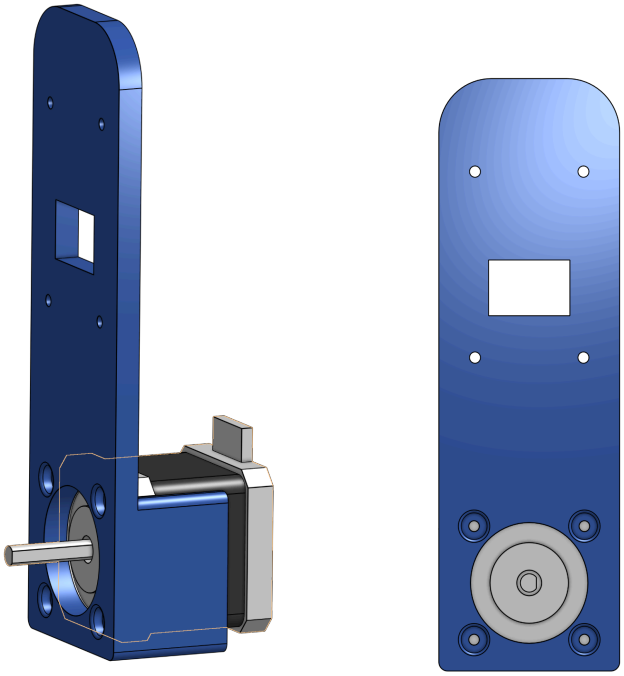


Figure 2: Stepper motor mount from side and front view

Figure 3: 3D printed stepper motor mount with stepper motors and wheels

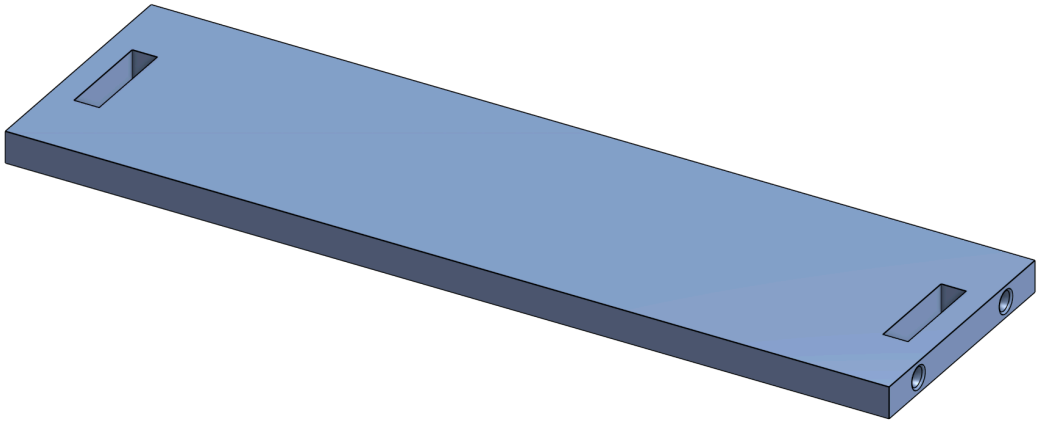


Figure 4: Upper deck model

Figure 5: 3D printed upper deck with head threaded inserts

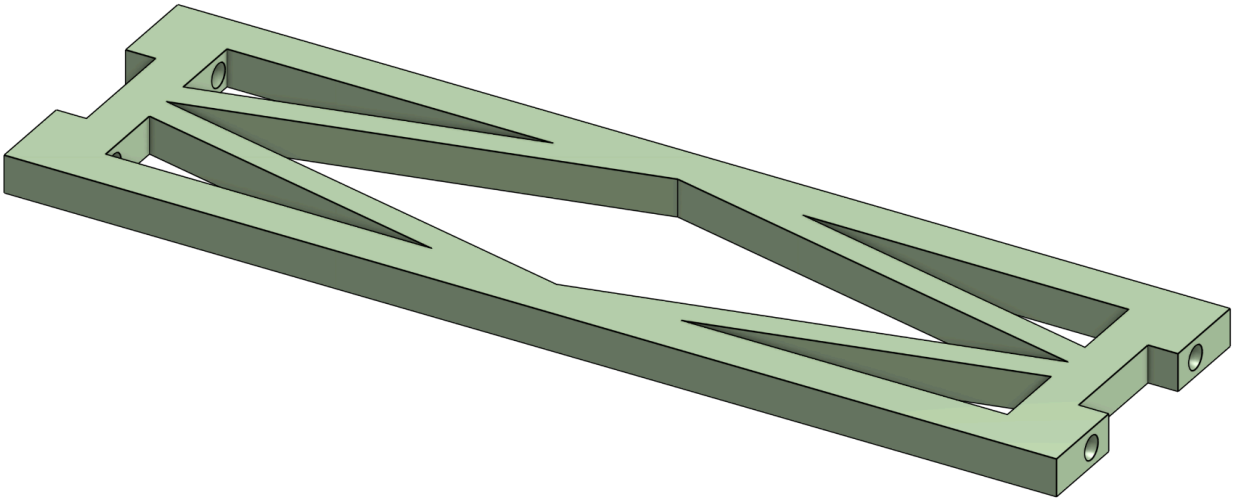


Figure 6: Lower deck model

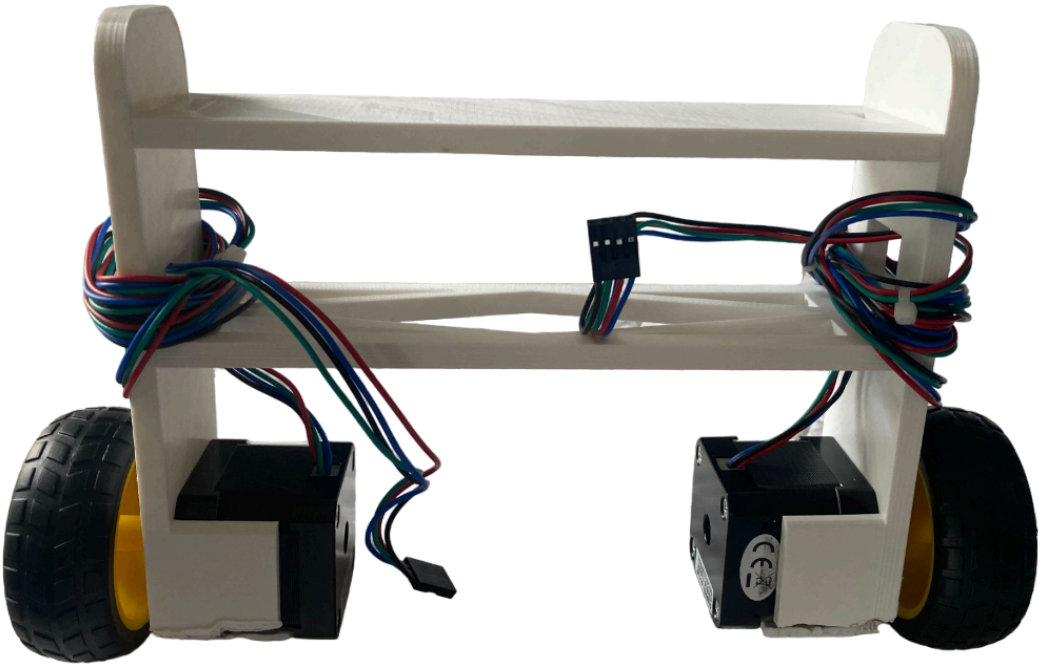


Figure 7: Full chassis assembly with stepper motors and wheels

Figure 8: Modified stepper motor mount

Electronics

The electronics went through two different design iterations but consisted of mostly the same parts. The current iteration of the electronics uses an Arduino Uno, two A4988 stepper motor drivers, two NEMA17 42-40 stepper motors, one MPU-6050 IMU, three 18650 9900 mAh LiPo batteries, one 10k potentiometer, and one 100µF decoupling capacitor. The schematic for the current electronic configuration is shown in Figure 9.

The reason for switching to a new microcontroller was due to an error in my original schematic. Initially, I had two separate sets of 18650 batteries plugged into each side of the breadboard power rails. I am unsure why I made this choice, as one side was supposed to feed the stepper motor drivers and microcontroller, which needed above 5V to work, while the other was supposed to be 3.3V to supply the sensors and IC of the motor driver. When I plugged both pairs of batteries into the power rails, I sent too much current through the 3.3V regulator on the ESP32 and fried it. Despite this, the ESP32 still worked, but the USB port could no longer transmit data to my computer, preventing communication with the ESP32 and making it impossible to upload new code. The original electronics are shown in Figure 10.

I decided to switch to an Arduino Uno instead of buying another ESP32, as it was the only other microcontroller I had available and I wanted to get it working as a proof of concept first. Unfortunately, due to the nature of the Arduino Uno, I was unable to get the radio communication working through the s-bus protocol since the Arduino Uno required an external circuit to invert the signal coming from the transmitter to read it. I built this simple circuit and tested it, but I could not get sensible information from the Arduino. I decided to remove it altogether since I was already behind schedule. I also removed the OLED display since it was unnecessary for the completion of the robot.

This was a silly mistake, but it was an excellent learning experience since it taught me not to skip any tests before fully integrating the components.

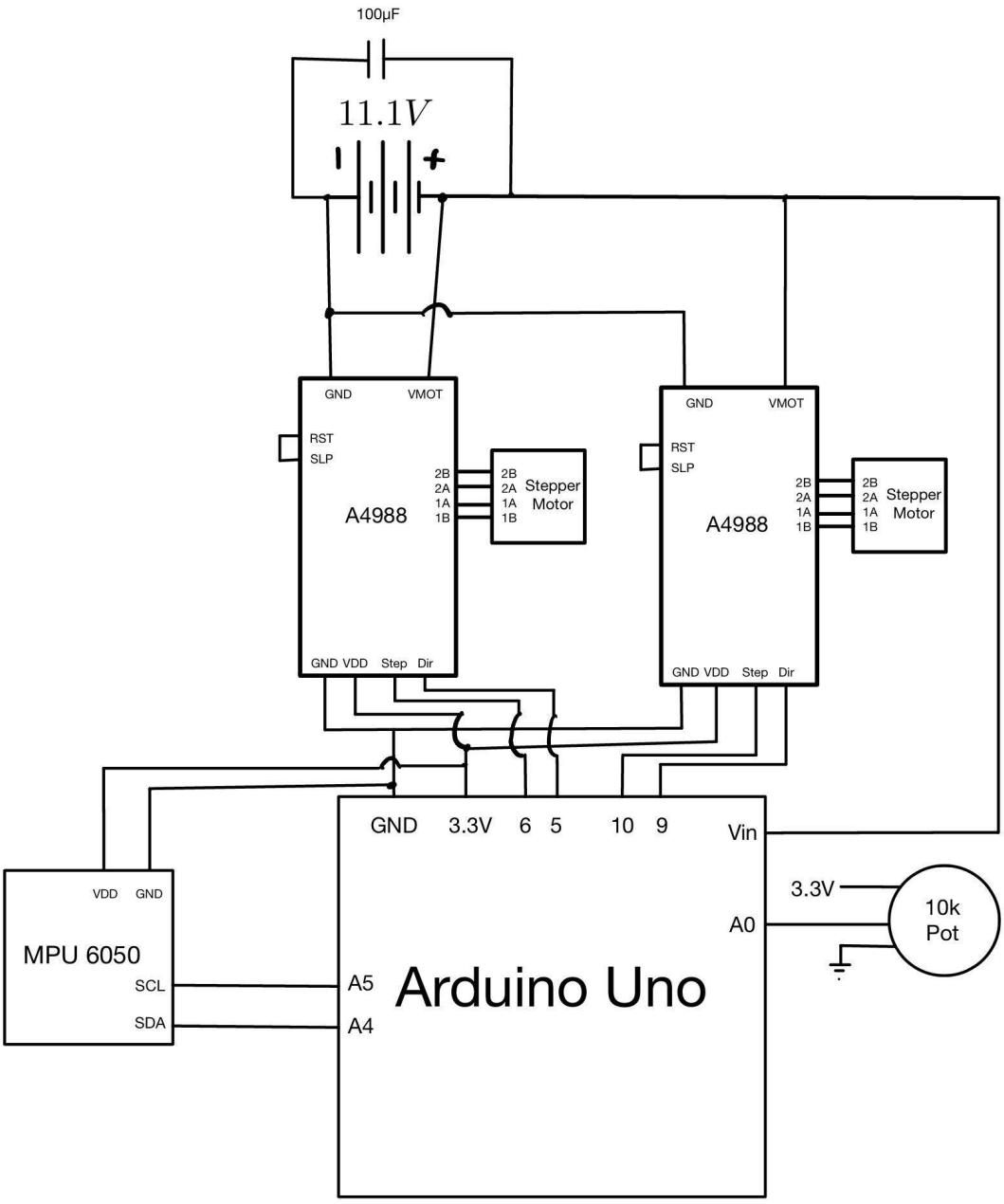


Figure 9: New schematic for self balancing robot with Arduino Uno

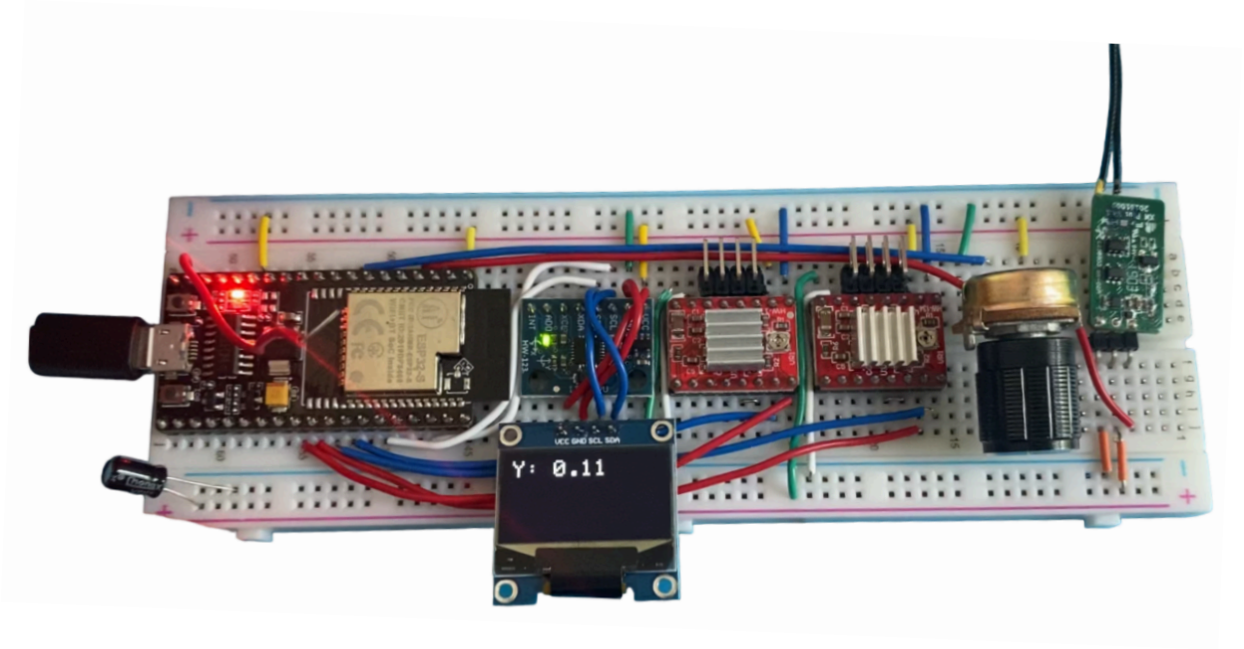


Figure 10: Original breadboarded electronics

Control Algorithm

One of the main goals of this project was to understand how to use a closed-loop PID (Proportional-Integral-Derivative) controller. A PID controller is a control loop that uses the error value between a set point (the angle read when the robot is perfectly straight) and the current angle of the robot to maintain desired output levels despite disturbances or changes in the system. The PID output feeds into the stepper motor as a velocity, and the cycle begins again. Figure 11 demonstrates the flow of the PID controller.

The proportional term produces an output that is proportional to the current error value. The proportional gain (*Kp*) determines the reaction to the current error. This is given by the formula:

$$P = Kp \cdot e(t) \quad \textbf{(1)}$$

Where *P* is the proportional term, *Kp,* is the proportional gain, and *e(t)* is the error at time *t*. The proportional term is used to eliminate the error quickly, but can lead to oscillations as it can overshoot the setpoint and oscillate back and forth.

The integral term deals with the accumulation of past errors. The integral gain (*Ki*) determines the reaction based on the sum of past errors, helping to eliminate the residual steady-state error. This is described by the mathematical formula:

$$I = Ki \cdot \int_0^t e(t)dt \quad \textbf{(2)}$$

Where *I* is the integral term and *Ki* is the integral gain. This term will ensure the output reaches and stays at the setpoint with no error.

The derivative term predicts future error based on its rate of change. The derivative gain (*Kd*) determines the reaction to the rate at which the error is changing, providing a damping effect and improving system stability.

$$D = Kd \cdot \frac{de(t)}{dt} \quad \textbf{(3)}$$

Where *D* is the derivative term and *Kd* is the derivative gain. The derivative term provides an output based on the rate of change of the error, which will dampen the speed of the response and reduce oscillations.

The PID control loop is a very popular control system as it is simple, effective, and requires an understanding of the system to tune it properly.

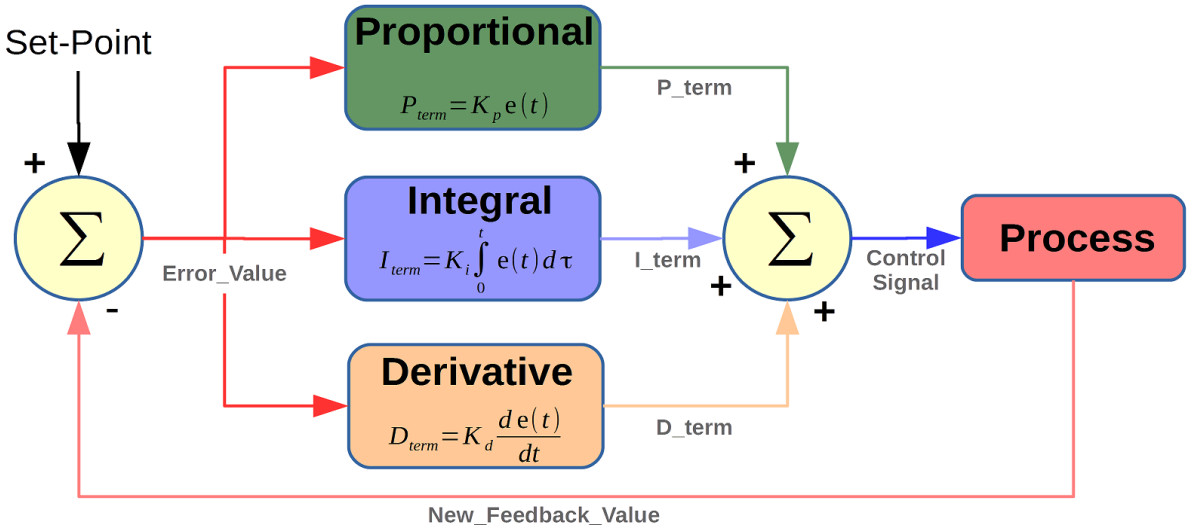


Figure 11: PID controller flowchart

Code

Originally, the code was supposed to run on the ESP32, but due to a poorly made schematic, I had to switch to an Arduino Uno and decided to remove much of the original functionality to see if the robot would work with the most basic components before spending more money on another ESP32.

The new code first initializes the MPU6050 and stepper motors and calibrates them. The MPU6050 communicates with the microcontroller using the I2C protocol. Then, using an attach interrupt function, the angles are analyzed and sent to the PID control loop whenever the MPU6050 has read and processed new data. The PID control loop then executes the algorithm and produces an output. This output is mapped to the speed of the stepper motors and subsequently runs for one step (1.8°) on the stepper motor.

To control the robot's setpoint, I put a potentiometer on the breadboard to adjust it manually while testing. The setpoint represents the verticality of the robot. Instead of setting the setpoint to 0°, the MPU6050 might read a stable angle value greater or less than 0° due to the inclination of the ground, how the sensor was mounted, etc. The potentiometer reduces ambiguity by allowing real-time adjustment of the robot's setpoint.

As a safeguard, I only allowed the stepper motors to run if the pitch (angle) of the robot was between -25° and 25° from the vertical position. If the robot were to fall, the stepper motors would not turn to react to the new position. Additionally, to reduce oscillations further, I prevented the stepper motors from running if the output of the PID controller was between -100 and 100, which mapped out to +/- 3 degrees from the upright position. With these adjustments and proper tuning of the parameters, the robot performed outstandingly. Figure 12 illustrates the flow of the code.

The original code was more ambitious, as it used a custom website with the ESP32 to tune the PID loop parameters in real-time instead of having to plug the microcontroller back into my computer every time to adjust the values. It also could read receiver input from my FrSky XM+ while using my Taranis Qx7 radio transmitter to control the robot's direction. The integration of these features was never tested, but the code and theory were written. The PID controller's output would be adjusted by the values read from the pitch and yaw inputs on the transmitter stick and then sent as speed values to the stepper motors. It also had an OLED display that showed the setpoint and pitch values along with the website's IP address.

Check out the code linked on my GitHub page below. The test scripts I used before fully integrating the components along with the final code are published.

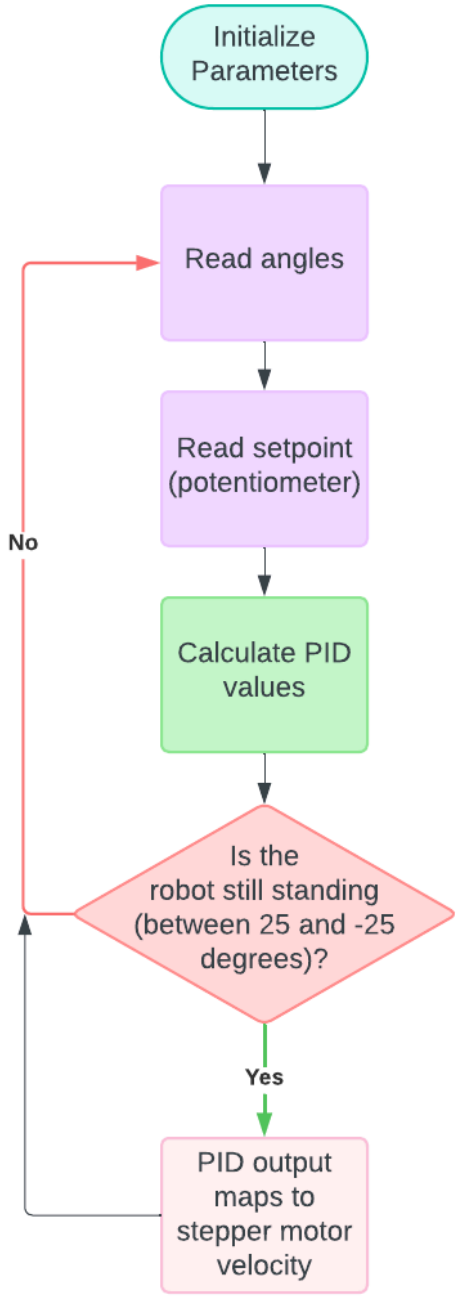


Figure 12: Code flowchart

GitHub: ***https://github.com/EliottLacommeRuiz/Self-Balancing-Robot***

Results

After successfully completing the robot, I wanted to push it to its limits. As seen in the Google Drive linked below, I conducted two tests: the inclined ramp test and the push test.

The inclined ramp test consisted of placing the robot on a ramp, adjusting its setpoint so it would balance, and observing if the robot could maintain its position on the ramp. Ramp inclines of 0°, 6°, 12°, 15°, and 18° were tested. The robot was able to maintain its position up to a 15° incline before the wheels started to slip. After observing the robot on the 18° incline, I believe the main cause of the slipping is the wheels. The wheels are made of two parts: the plastic outer wheel and the rim, as seen in Figure 13. The rim would spin quickly to overcome the force of the falling robot, but the outer wheel would slip and not spin with the rim. For future improvements, if overcoming obstacles or climbing steep ramps is necessary, I recommend using wheels that are made of one part or have more surface contact than these wheels.

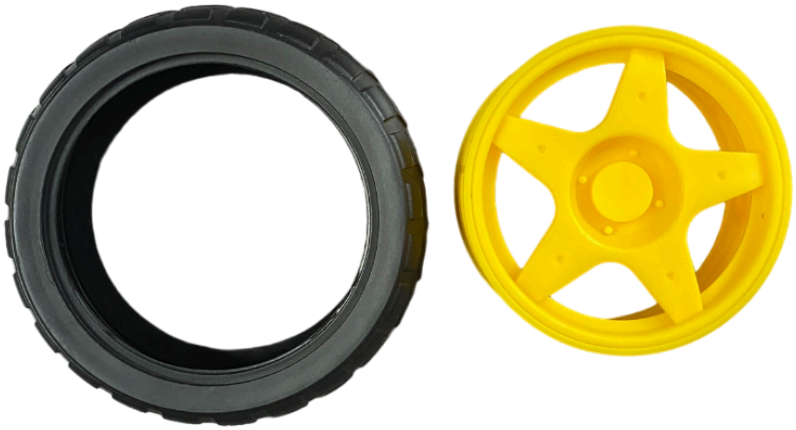


Figure 13: Wheel rim and outer plastic

The push test was designed to visually determine if the robot could withstand small disturbances and oscillations to stay upright. I conducted three push tests where I progressively pushed the robot harder from both sides to see how well it would recover. As seen in the videos, the robot can withstand aggressive nudges before returning to a stable position.

(Videos of the robot functioning can be found here: *Self Balancing Robot (7/11/2024)* *)*

Total Hours Spent: 85 hours (including report)

Future Development

In the future, I would like to add remote control functionality. I already have everything set up for it to work, but I just need another ESP32, so when I buy another one, I would like to revisit this project and fix it up.

Additionally, I would like to make the robot even more responsive. Currently, it takes quite some time to adjust itself back to a stable position. It would be fascinating to see if I can get it to a point where it does not fall over even with a hard shove.

I would also like to continue using similar hardware, like the heat threaded inserts and M-size screws, to keep everything ergonomic and consistent.

Bill of Materials

Self Balancing Robot - Bill of Materials

Materials	Name	Cost	Quantity	Source	Status
Stepper motor drivers	HiLetgo A4988	$10.19	5	https://a.co/d/8xkJmJ4	Delivered
	WWZMDiB A4988	$7.99	3	https://a.co/d/6g0JAdM	Not using/found
	A4988	$0.99	1	3D Printer Parts A4988 D	Not using/found
Stepper motors	STEPPERONLINE NEMA17 (42-40)	$26.00	2	https://amzn.to/2M3aJK2	Delivered
Battery Holder	DAIERTEK 18650 Battery Holder	$9.99	8	https://a.co/d/1YDkMoL	Delivered
Battery	18650 9900mAh battery with charger	$23.98	4 and charger	https://a.co/d/15O8f9T	Delivered
IMU	HiLetgo MPU-6050	$9.99	3	https://a.co/d/izGjSoG	Delivered
ESP32	ESP32 Dev V1	$0.00	0	NA	Have it already
Wheels	From another kit				Have it already
Chassis	3D printed				Have it already
Reciever	FrSky XM+	$0.00	0	NA	Have it already
Transmitter	Taranis QX7	$0.00	0	NA	Have it already
M Bolt Set	M Bolt Set	$23.77	1255	https://a.co/d/0iKx9rV4	Delivered
Threaded Inserts	Threaded Inserts	$11.49	365	https://a.co/d/0gfsbmz5	Delivered
		Total: $103.92			