# Self-Balancing Robot

Course: ELEC 391 201 2024W2

Authors: Team B-17 (Tomaz Zlindra, Muntakim Rahman, Xianyao Li)

Instructors: Dr. Joseph Yan and Dr. Cristian Grecu

Submission Date: April 13th, 2025

# Contents

# 1 Introduction

One of the engineering challenge we need to deal with is unlike the LTI system we focused on in ELEC 221 / 341, the bot is a non-linear, time-variant system, moreover, since the inverted pendulum have a tendency to tip over, this system is also unstable, that means most of the PID tuning techniques are not applicable to the system. For example, the Ziegler-Nichols method is a fast and effective tuning technique, but it cannot be applied to the system since the method requires a stable system. the non-linearity of the system also implies that the system can only be controlled in a region where the response of the system can be approximated as a linear system, and the time-variance of the system implies that the system is unobservable since we cannot determine the internal state of the system based on its output measurements. These factors bring extra challenge for controlling the system, and we need to come up with an effective control scheme that can deal with those issues.

Furthermore, there are also enginnering problems we need to solve with the PID controller. For example, the D-term amplifies higher frequency measurement or process noise, which can cause large amounts of change in the output. Another problem with the PID controller is the integral windup, when a large change in setpoint occurs, the I-term will accumulates a significant error during windup, causing overshoot and degraded stability and potentially makes the system ubstable. These engineering problems need to be solved by having modification to the algorithm, so that it can be addressed.

# 2 Requirements, Specifications and Constraints

## 2.1 Functional Requirements

Table 1 lists the functional requirements of the self-balancing robot. These are prioritized by category, either a core or additional feature. The core functionality of the robot relates to its ability to autonomously balance itself and be driven by a human user via Bluetooth. The additional features were designed to enhance the robot's capabilities, in detecting environmental factors and providing useful information to the user.

| ID | Requirement Description | Priority |
|---|---|---|
| FR1 | The robot must balance on a flat surface by controlling the DC motors. | Core |
| FR2 | The robot must maintain balance while driven externally on a flat surface. | Core |
| FR3 | The robot must maintain balance while turning at a reasonable speed. | Core |
| FR4 | The robot must move forward and backward under external control. | Core |
| FR5 | The robot must be controlled via Bluetooth with four directional commands. | Core |
| FR6 | The robot must be able to recover when disturbed | Core |
| FR7 | The robot is powered solely by a pack of 8 $\times$ 1.2V rechargeable batteries. | Core |
| FR8 | The distance sensor must detect obstacles and stop when it gets too close to an object | Additional Feature |
| FR9 | The color sensor must be able to detect at least one color | Additional Feature |
| FR10 | The RFID module must authenticate an RFID tag and prevent unauthorized operation. | Additional Feature |
| FR11 | The Raspberry Pi Zero must process and transmit a camera feed through Pi Connect | Additional Feature |
| FR12 | The AWS Bucket should store all the saved images and videos from the camera | Additional Feature |

Table 1: Functional Requirements of the Self-Balancing Robot

## 2.2 Performance Requirements

Table 2 outlines the performance requirements for the self-balancing robot, providing specific quantitative targets to assess the system's effectiveness. These requirements closely align with the functional requirements, but with the addition of measurable criteria—such as recovery angle, speed and transmission speed. By translating functional

goals into performance metrics, we ensured that the robot's behavior could be consistently validated against clearly defined benchmarks. This approach helped guide both the design process and iterative improvements throughout development.

| ID | Requirement Description | Priority |
|---|---|---|
| PR1 | The robot must respond to a 15 degree disturbance and re-balance | Core |
| PR2 | The robot must maintain balance with a positional drift of less than 15 cm over 10 seconds on a flat surface. | Core |
| PR3 | The Bluetooth latency for control commands must be less than 100 ms. | Core |
| PR4 | The robot must be able to maintain a Bluetooth connection within a range of at least 5 meters. | Core |
| PR5 | The robot must be to move forward, backward, left (45°) and right (45°) within 5 seconds | Core |
| PR6 | The obstacle detection system must stop the robot within 1 cm of an obstacle detected between 2–30 cm. | Additional Feature |
| PR7 | The color sensor must identify red and respond accordingly within 100ms | Additional Feature |
| PR8 | The RFID reader must successfully detect and authenticate tags within 1 second. | Additional Feature |
| PR9 | The Raspberry Pi Zero must transmit a camera feed at no less than 1 FPS. | Additional Feature |

Table 2: Performance Requirements of the Self-Balancing Robot

## 2.3 Constraints

### 2.3.1 Financial

This project operated under a strict financial constraint, with a maximum budget of $65 allocated specifically for additional features (excluding taxes and shipping). To maximize the number of features within this limit, we opted to purchase components from alternative vendors when they offered more competitive prices than DigiKey. We determined that the most effective use of our budget was to prioritize implementing multiple smaller features, rather than allocating the entire amount toward a single major feature.

For a detailed breakdown of the expenses, refer to the Google Sheet with all financial details An image can also be found in the Appendix in Figure 27.

### 2.3.2 Temporal

Although the final project deliverable was due on April 8th, 2025, we set several soft deadlines along the way to track our progress and prevent last-minute workload buildup. Some of the main deadlines are shown below.

**February 28th, 2025** Motor Control PWM for Arduino

**March 6th, 2025** Completed BLE Integration with Python App

**March 7th, 2025** Completed Distance Sensor and Color Sensor STM32 Firmware

**March 13th, 2025** Completed PID k-value Calculations

**March 14th, 2025** Completed RFID STM32 Firmware

**March 20th, 2025** Completed Raspberry Pi App

**March 21st, 2025** Completed Speaker Integration STM32 Firmware

**March 28th, 2025** Completed Robot Balancing

Although most deadlines were met, a few, particularly the robot balancing milestone were not achieved on time. This delay was primarily due to some questionable firmware design decisions that introduced unforeseen challenges. Fortunately, these issues were resolved, and the system was successfully completed in the end.

### 2.3.3 Materials

For the custom construction of our self-balancing robot, we were limited to using thick plastic for laser cutting and standard 3D printing filament, as these materials were readily available and did not impact our budget. While other materials—such as aluminum, carbon fiber, or specialized fasteners—could have offered enhanced strength or reduced weight, they would have required additional spending from our $65 budget. To stay within our constraints, we opted not to purchase any external materials and instead focused on optimizing our design using the provided components and available fabrication methods.

### 2.3.4 Environmental

The performance of a self-balancing robot can be significantly affected by environmental constraints such as battery levels, electromagnetic interference (EMI), temperature, and terrain conditions. Low battery voltage can reduce motor torque and sensor accuracy, leading to instability. EMI from nearby motors or power electronics can disrupt IMU readings and communication systems, making balance control less reliable. Temperature extremes can alter sensor calibration and battery efficiency, while uneven or slippery terrain can interfere with traction and cause the robot to lose balance. These factors must be carefully considered to ensure reliable operation in real-world environments.

# 3 Conceptual Design

## 3.1 High-level Functional Diagram

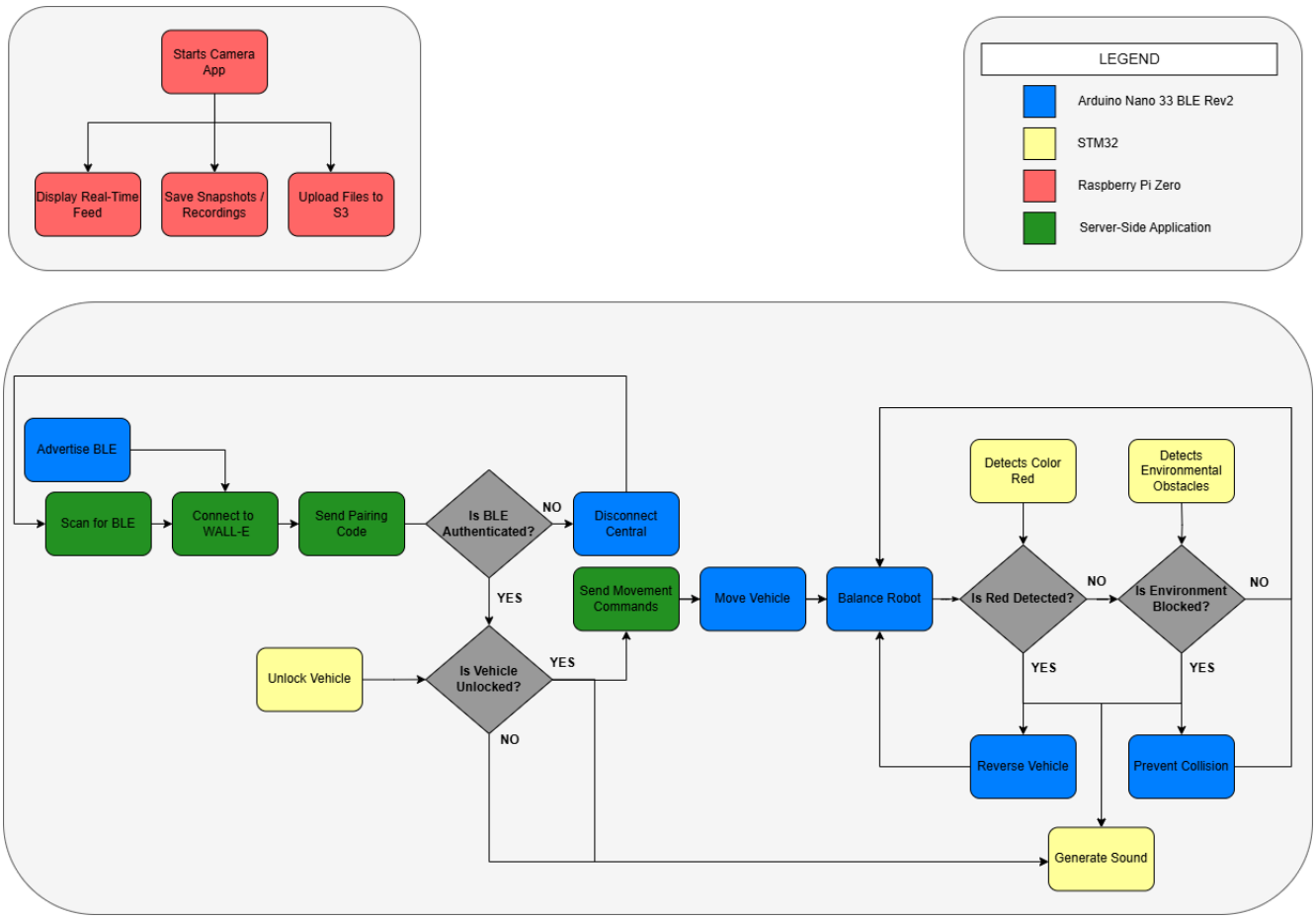


Figure 1: High-level Functional Diagram

The functional diagram in Figure (1) highlights our system-design for enforcing secure access of the robot with **RFID** and **Bluetooth Low Energy (BLE)**. We prevent users from balancing or moving the robot without the required authentication. Once successful, the firmware was programmed to prioritize safe operation of the robot, while accounting for environmental factors such as distance and color detection.

## 3.2 Architecture Diagram

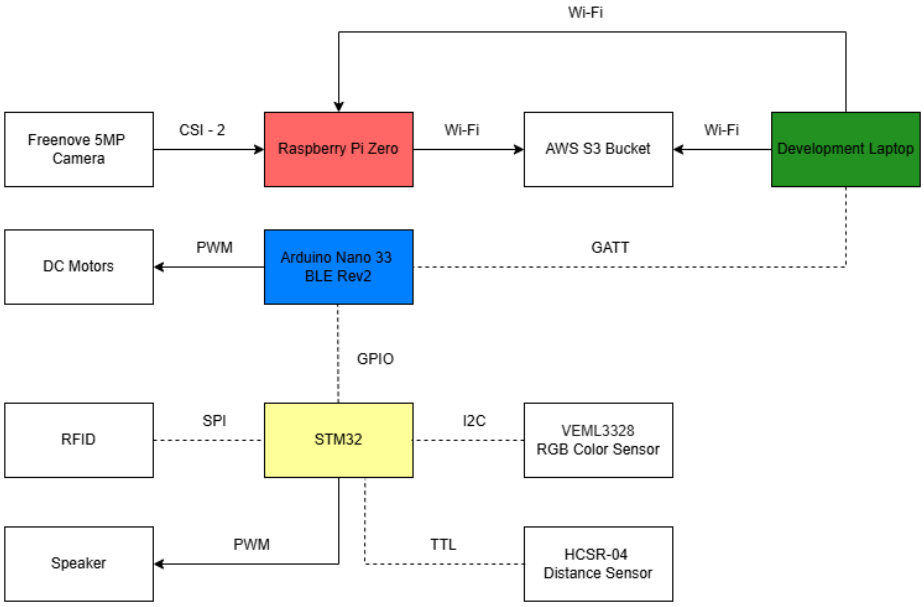


Figure 2: Architecture Diagram

The architecture diagram in Figure (2) describes the interfaces between robot peripherals and the three boards used in the project:

- **Arduino Nano 33 BLE Sense Rev2** was responsible for driving the motors and maintaining balance.
- **STM32L051K8T6** was responsible for environmental detection and physical authentication.
- **Raspberry Pi Zero W** was responsible for displaying and processing camera input for the user.

*Note that we wirelessly accessed the **Raspberry Pi** and **Arduino** from a development machine.*

# 4 Subsystem Design

## 4.1 Structural Design

### 4.1.1 Design Decisions

Since the transfer function of the robot was derived under the assumption that the center of mass of the pendulum is perfectly centered, the structural design follows the principle of evenly distributing mass to ensure that the center of mass is as close as possible to the rotor axis.

Another design decision we made was to mount the third layer as high as possible. This choice addresses the fact that the plant is a nonlinear and inherently unstable system, which can only be effectively controlled within a region where it can be approximated as linear. To evaluate the impact of the third layer's height on system behavior, we simulated the plant's step response at different mounting heights:

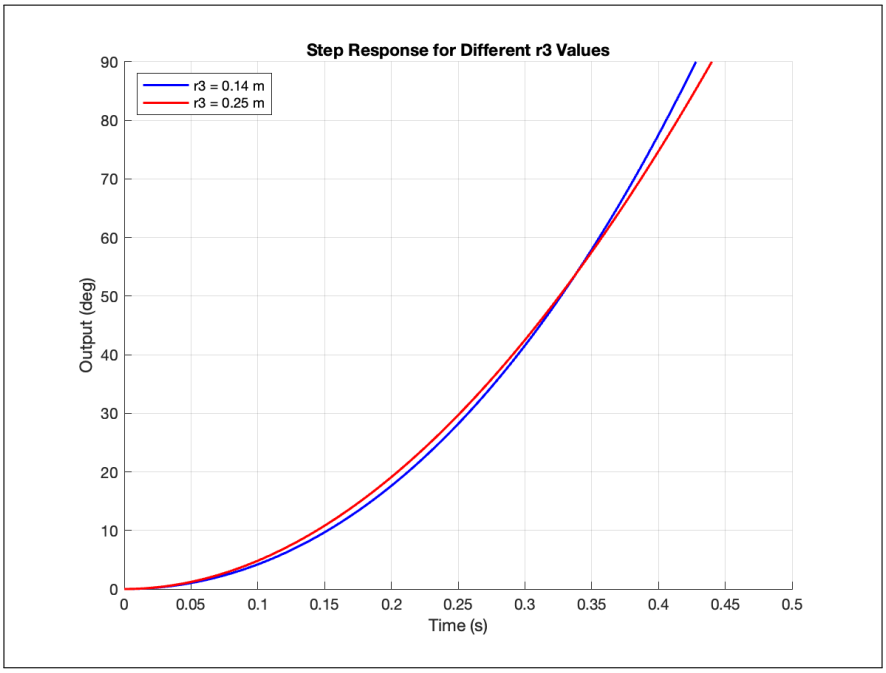


Figure 3: Step Response of the Plant with Different Height of the Third Layer

As shown in the figure, increasing the height of the third layer results in a step response that more closely resembles that of a linear system, making it easier to control.

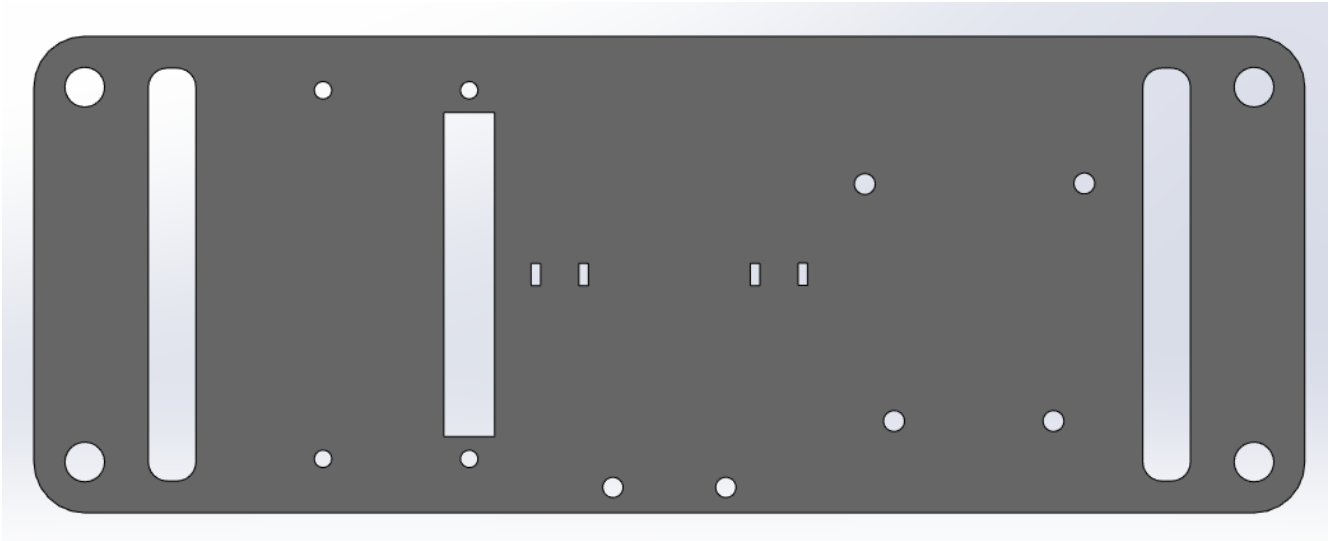

Figure 4: Design of the Third Layer

#### 4.1.2 Third Layer Design

On the third layer, we need to mount the Raspberry Pi, color sensor, RFID reader, and camera. The four mounting holes on the left side are for securing the Raspberry Pi, while the four holes on the right are for the RFID reader. These holes are positioned to ensure that both components are centered relative to the third layer once mounted. At the bottom, two mounting holes are used to attach the L-stand for the color sensor. Since we only have one color sensor, its placement causes a deviation in the center of mass. To compensate for this imbalance, the camera is zip-tied at the center, allowing its position to be adjusted to help realign the center of mass. Although not shown in the figure, two distance sensors are also mounted on the second layer symmetrically.

#### 4.1.3 L-stand Design

The design of the L-stand is shown below:

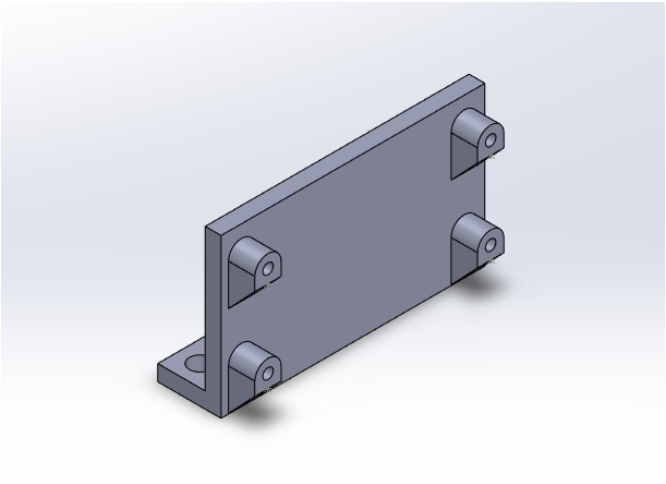

Figure 5: Design of the L-stand for Mounting the Distance Sensor

The L-stand includes four columns to provide clearance for the distance sensor, as there are components located on the back of the PCB. Ribs are added beneath each column to address potential issues during 3D printing. Since overhangs can lead to poor print quality or failed prints, the ribs provide structural support under the columns, eliminating overhangs and making the printing process more reliable and efficient.

The L-stand for the color sensor follows the same design principles with different dimension, thus is not shown here.

### 4.2 PID Modeling and Simulation

#### 4.2.1 Measuring Robot Parameters

With the corrected transfer function, we need to find the values of the parameters in order to get the transfer function.

We start with the motor constants. For $K_t$, from the datasheet we know that the stall current is 5.5 A and the torque at 12 V is 85 kg·mm, or 0.83N·m, we can then get the value of $K_t$ by dividing the torque by the stall current, which is 0.1516.

For $K_e$, from the datasheet we know that the free run speed of the rotor without the gearbox is 1047.19 rad/s, we can then get the value of $K_e$ by dividing the load voltage 12 V by the free run speed of the rotor, which is 0.01146.

To obtain the moment of inertia of the pendulum $J_p$, we need to simplify the plant model. The robot consists of four rods and three layers. Assuming the center of mass is perfectly centered, the four rods can be approximated as a single equivalent rod located at the center of the three layers. Similarly, the three layers can be modeled as three point masses, which are further simplified into a single point mass located at their collective center of mass. The resulting simplified model consists of a single rod rotating about one end and a point mass positioned at the center of mass of the three layers.

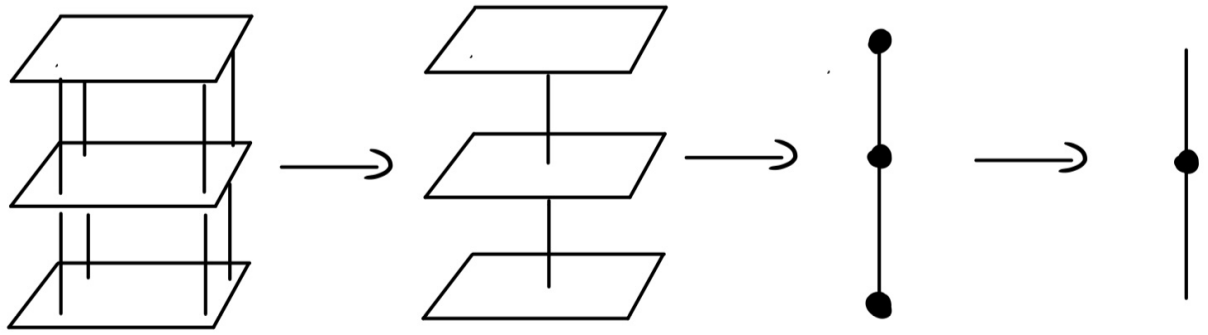

Figure 6: Simplification of the Plant Model

All of the other parameters can either be measured directly, or are negligible and have minimal impact on the overall transfer function.

#### 4.2.2 Transfer Function

Considering the difficulty of deriving the transfer function from first principles, and given the abundance of studies on the same topic, we chose to reference a published paper for guidance [1]. However, upon reviewing the paper, we identified errors in part of the derivation:

Assuming the correctness of

$$K_t \frac{V_s - K_e \dot{\theta}_m}{R_a} = J_m \ddot{\theta}_m + \tau_w + b\dot{\theta}_m \tag{1}$$

$$\tau_w = J_w \ddot{\theta}_w - F_{pH} r_w + m_w \ddot{\theta}_w r_w^2 \tag{2}$$

$$F_{pH} = m_p(r_w \ddot{\theta}_w + l_g \ddot{\theta}_p) \tag{3}$$

$$F_{pH} l_g = J_p \ddot{\theta}_p \tag{4}$$

$$\theta_m = \frac{\theta_w}{G_r} \tag{5}$$

Substitute (3) into (4) gives:

$$\begin{aligned} m_p(r_w \ddot{\theta}_w + l_g \ddot{\theta}_p) l_g &= J_p \ddot{\theta}_p \\ m_p l_g r_w \ddot{\theta}_w + m_p l_g^2 \ddot{\theta}_p &= J_p \ddot{\theta}_p \\ m_p l_g r_w \ddot{\theta}_w &= (J_p - m_p l_g^2) \ddot{\theta}_p \\ \ddot{\theta}_w &= \frac{J_p - m_p l_g^2}{m_p l_g r_w} \ddot{\theta}_p \\ \theta_w &= \frac{J_p - m_p l_g^2}{m_p l_g r_w} \theta_p \end{aligned}$$

So Eq. 4 in the paper is incorrect. The $\tau_w$ in denominator should be $r_w$. The signs are also wrong. Substitute (3) into (2) gives:

$$\begin{aligned} \tau_w &= J_w \ddot{\theta}_w - m_p(r_w \ddot{\theta}_w + l_g \ddot{\theta}_p) r_w + m_w \ddot{\theta}_w r_w^2 \\ &= (J_w + (m_w - m_p) r_w^2) \ddot{\theta}_w - m_p l_g r_w \ddot{\theta}_p \end{aligned}$$

Substitute the above and (5) into (1) and expand the first fraction:

$$\frac{K_t}{R_a} V_s - \frac{K_t}{R_a} \frac{K_e}{G_r} \dot{\theta}_w - \frac{b}{G_r} \dot{\theta}_w - (J_w + (m_w - m_p) r_w^2) \ddot{\theta}_w + m_p l_g r_w \ddot{\theta}_p = \frac{J_m}{G_r} \ddot{\theta}_w$$

Move the angle terms to RHS and flip both sides:

$$\left(\frac{K_t}{R_a}\frac{K_e}{G_r}+\frac{b}{G_r}\right)\dot{\theta}_w+\left[\frac{J_m}{G_r}+J_w+(m_w-m_p)r_w^2\right]\ddot{\theta}_w-m_p l_g r_w\ddot{\theta}_p=\frac{K_t}{R_a}V_s$$

Substitute the expression of $\theta_w$ in terms of $\theta_p$:

$$(J_p-m_p l_g^2)\left(\frac{K_t}{R_a}\frac{K_e}{G_r}+\frac{b}{G_r}\right)\dot{\theta}_p+\left[(J_p-m_p l_g^2)\left[\frac{J_m}{G_r}+J_w+(m_w-m_p)r_w^2\right]+(m_p l_g r_w)^2\right]\ddot{\theta}_p$$
$$=\frac{K_t}{R_a}(m_p l_g r_w)V_s$$

Laplace transform:

$$\theta_p(s)\left[(J_p-m_p l_g^2)\left(\frac{K_t}{R_a}\frac{K_e}{G_r}+\frac{b}{G_r}\right)s+\left[(J_p-m_p l_g^2)\left[\frac{J_m}{G_r}+J_w+(m_w-m_p)r_w^2\right]+(m_p l_g r_w)^2\right]s^2\right]$$
$$=\frac{K_t}{R_a}(m_p l_g r_w)V_s(s)$$

Hence the final transfer function:

$$\frac{\theta_p}{V_s}(s)=\frac{\frac{K_t}{R_a}(m_p l_g r_w)}{\left[(J_p-m_p l_g^2)\left[\frac{J_m}{G_r}+J_w+(m_w-m_p)r_w^2\right]+(m_p l_g r_w)^2\right]s^2+(J_p-m_p l_g^2)\left(\frac{K_t}{R_a}\frac{K_e}{G_r}+\frac{b}{G_r}\right)s}$$

The definition of the parameters can be found in the paper.

Plug in the parameters we acquired, the resulting open loop transfer function is:

$$\frac{\theta_p}{V_s}(s)=\frac{977.5}{s^2+0.3439s}$$

Notice that the moment of inertia of the wheel and the rotor ($J_m$&$J_r$) and friction $b$ are approximated to zero, This is because their magnitudes are negligible compared to other parameters in the system, thus they have minimal impact on the overall transfer function.

#### 4.2.3 PID Parameter Calculation

From the previously derived transfer function, we observe that the two open-loop poles are located at 0 and -0.3439, indicating that the system is unstable. To stabilize the system, a feedback control loop must be introduced to shift the poles of the closed-loop transfer function to stable locations. To determine the PID parameters, we can apply the pole placement method:

Assuming that we want the resulting poles to be placed at $P_A$ and $P_B$, the denominator of the resulting transfer function is:

$$(s-P_A)(s-P_B)$$

This can be written explicitly as:

$$s^2-(P_A+P_B)s+P_A P_B$$

With a PD controller, the characteristic equation is:

$$s^2+(K\cdot K_d-(P_1+P_2))s+P_1P_2+KK_p$$

Where $P_1$ and $P_2$ are poles of the open loop transfer function, $K$ is the DC gain of the transfer function, which is 977.5.

We can relate the above two equations and the following equations can be obtained:

$$P_A P_B=P_1P_2+KK_p$$
$$-(P_A+P_B)=KK_d-(P_1+P_2)$$

The resulting PD gains will make the close loop transfer function be critically damped and stable.

To determine the desired pole locations, we first need to select an appropriate time constant $N$. By measuring the runtime of the main control loop, we found the worst-case execution frequency to be 25 Hz. According to the Nyquist theorem, the sampling rate must be at least twice the highest frequency component to accurately capture the system dynamics. Therefore, the maximum allowable control bandwidth is 12.5 Hz, corresponding to a minimum time constant of approximately 0.08. To ensure stability and responsiveness while staying within this limit, we chose a time constant of 1/4 seconds, which places the desired pole at $-8\pi$ and the resulting $K_p$ and $K_d$ are 0.6462 and 0.0511, respectively.

The I term is added after to address the drifting issue we encountered during the testing. The value of $K_i$ was found by trial and error, and the final value of $K_i$ is 11.5.

#### 4.2.4 Simulink Simulation

Simulink is used to validate the design of the PID controller. The model is shown below:

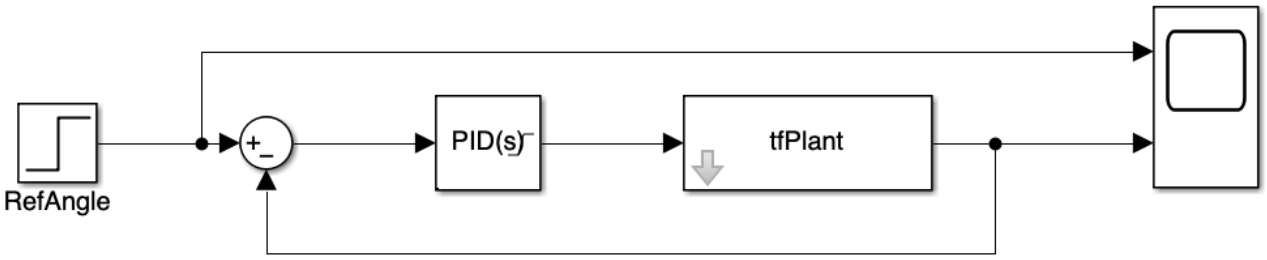


Figure 7: Simulink Model for the Bot

The step responses of the system under filter coefficient N = 50 and N = 25 were simulated to evaluate the performance of the robot under best- and worst-case scenarios:

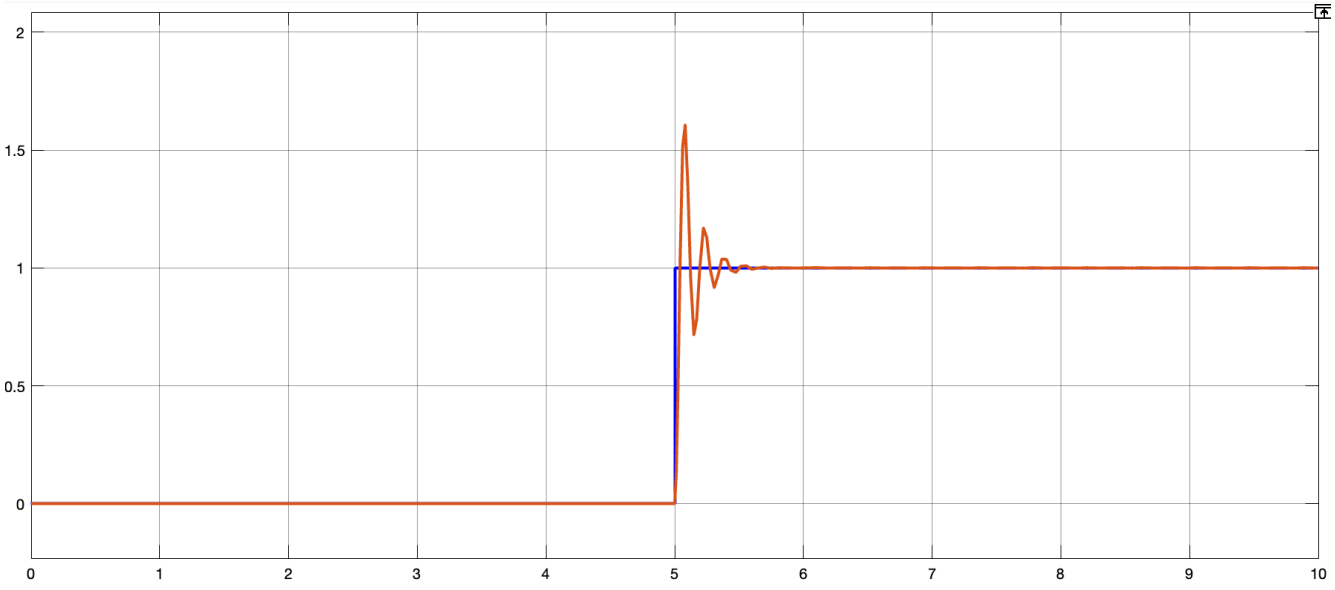


Figure 8: Step Response of the System with N = 25

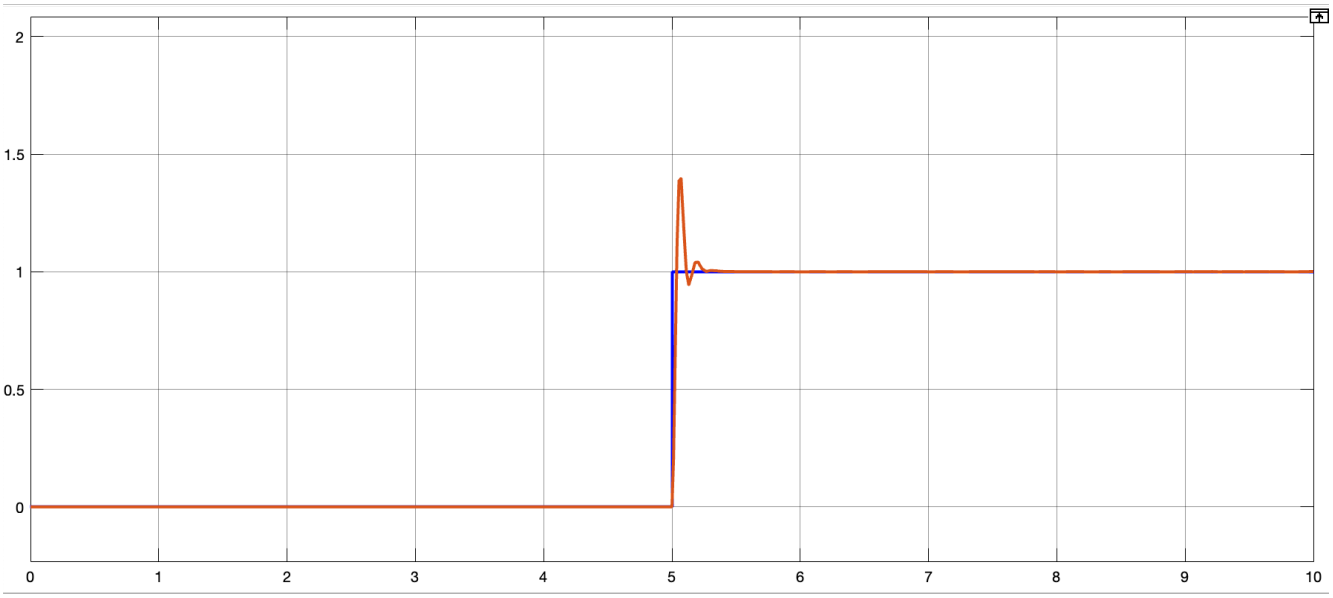


Figure 9: Step Response of the System with N = 50

We can see that for both cases the system is stable, with different degree of overshoot. This implies that the design of the PID controller is effective.

## 4.3 Arduino Nano 33 BLE Sense Rev2

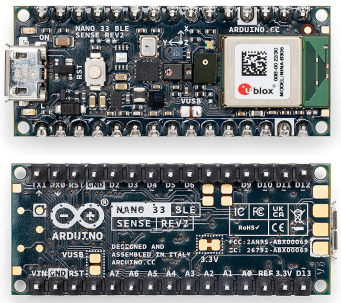


Figure 10: Arduino Nano 33 BLE Sense Rev2

The **Arduino Nano 33 BLE Sense Rev2** is a compact and powerful microcontroller board designed for low-power applications. It features the Nordic **nRF52840** chipset, which supports **BLE** communication, includes a range of built-in sensors, including a 9-axis IMU (accelerometer, gyroscope, and magnetometer). These integrated features make it well-suited for IoT, sensor-based applications, and portable devices.

This made it the ideal choice for our self-balancing robot project and was responsible for many of the core features. It integrated key core components in this project, particularly the **IMU** and the **BLE** module.

### 4.3.1 Arduino IDE

The **Arduino IDE** was integral in quickly testing new firmware changes to the board with built-in compilation and flashing tools. The extensive library support significantly streamlined the development process, reducing the need for bare metal programming and allowing us to focus on the functionality.

### 4.3.2 IMU Angle Measurement

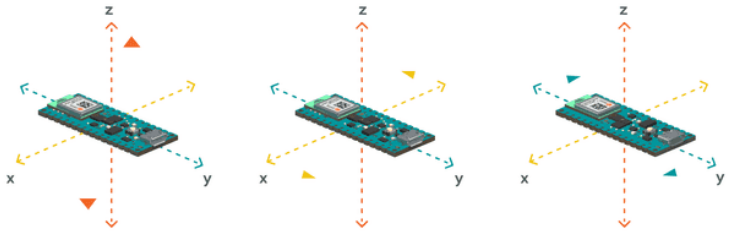


Figure 11: IMU

The **Inertial Measurement Unit (IMU)** was a critical microcontroller component for measuring velocity, orientation, and gravitational forces. This enabled the tracking of motion and orientation of the **Arduino** in three-dimensional space.

We used the accelerometer to measure linear acceleration and the gyroscope to monitor rotational movement. Both of these sensors had advantages and disadvantages, as shown in Table 8.2. By combining them, we ensured reliable and accurate angle measurements for optimized balancing in the control feedback loop.

### 4.3.3 Complementary Filter

The solution to combine the strengths of both sensors was to use a complementary filter, which combines the accelerometer and gyroscope readings with their respective normalized weights.

Since the gyroscope provided a rough estimate of the angle change, this was essential for detecting quick movements.

The formula used for the complementary angle is:

$$\theta_n = k(\theta_{n-1} + \theta_{g,n}) + (1 - k)\theta_{a,n},$$

where:

- $\theta_n$ is the current complementary angle,
- $\theta_{n-1}$ is the previous complementary angle,
- $\theta_{g,n}$ is the current gyroscope angle,
- $\theta_{a,n}$ is the current accelerometer angle,
- $k$ is the normalized weight.

$$\therefore \text{From extensive testing, a value of } k = 0.95 \text{ was selected.}$$

*Lower values of k caused the noise from the accelerometer to become significant, leading to large oscillations and loss of balance. Similarly, increasing k was avoided. The gyroscope bias would accumulate quickly (i.e., without a reference point) and cause the robot to lose balance.*

The code snippet for this exact implementation is demonstrated in Listing 1.

## 4.4 Autonomous-Balancing

The most challenging part of this project was to implement the PID controller to balance the robot. Since we had closed-loop control, with the measured angle acting as feedback, it was crticial that we were able to act upon new measurements when determining the control signal.

Initially, we attempted to implement these **Interrupt Service Routine (ISR)** to meet a designed control frequency of 20 Hz. However, this proved to accumulate complications in firmware logic and was not able to provide the expected results.

We ultimately designed a control loop, run in the main program. This followed the sequence shown in Figure 12.

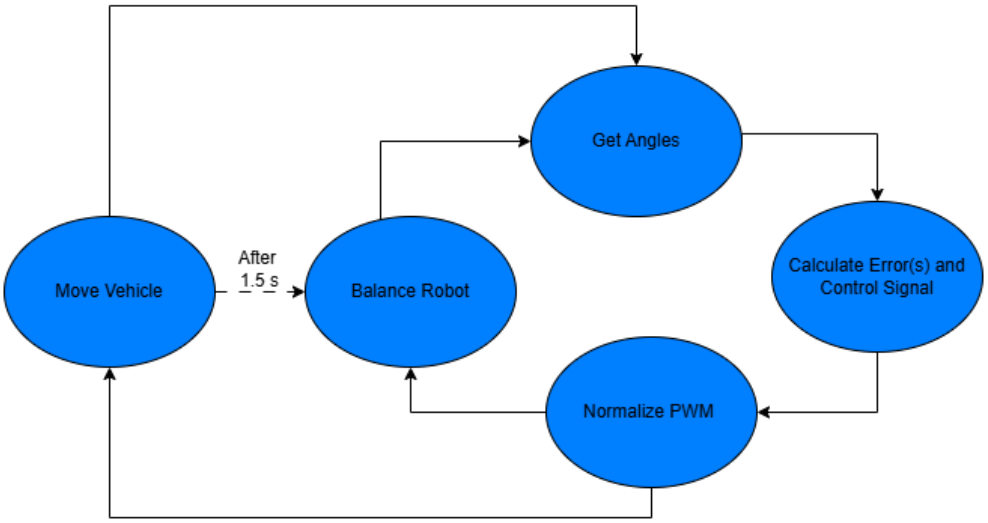


Figure 12: Control Loop Diagram

This ensured that we were able to inject **PWM** signals to both motors, without the constraints typical of an **ISR**. At the same time, this ensured that the robot was acting upon the most recent measurements.

The sign of the control signal determined whether to move the robot forward / backward to maintain balance.

The source code for this implementation is shown in Listing 2, in the Appendix.

### 4.4.1 Motor Control

To operate the robot wheels, we used the **DRV8833** motor driver to operate the robot with slow decay **PWM**. This entailed one of the motor terminals being driven to logic high, while the other was supplied the PWM signal.

| xIN1 | xIN2 | FUNCTION |
|---|---|---|
| PWM | 0 | Forward PWM, fast decay |
| 1 | PWM | Forward PWM, slow decay |
| 0 | PWM | Reverse PWM, fast decay |
| PWM | 1 | Reverse PWM, slow decay |

Figure 13: DRV8833 PWM Control

Supplying slow decay **PWM** enables the robot to gradually diminish its response to error angles, rather than abruptly stopping the motors. This was integral in ensuring the overall system behavior was stable and able to keep the robot balancing.

The source code for this implementation is shown in Listing 3, in the Appendix.

## 4.5 Movement

The robot was designed to move in four directions: **FORWARD, BACKWARD, LEFT, RIGHT**. As shown in Figure 12, this was designed to be integrated into the control loop.

Balancing was the highest priority of the system at any given time. To move the robot, we injected noise into the system for a short period of time (i.e. 1.5 seconds). After this, we returned to idle balancing to prevent the system from reaching a point of instability.

For the complete code snippet of this implementation, it can be found in Listing 4, in the Appendix.

### 4.5.1 Forward/Backward Movement

The logic for moving the robot forward and backward was very simple. It was a matter of adjusting the setpoint angle on either side of the "zero point". This induced movement in the indended direction (without sacrificing balance).

### 4.5.2 Left/Right Movement

The logic for moving the robot left and right was slightly more complicated and involved software logic to interchangeable turn and balance the robot. This can be summarized as follows:

- `After (every) 3 iterations` of control loop : `balance robot`
- `If error angle` $>$ `allowable threshold` (i.e. 0.8°): `balance robot`
- Otherwise:
  - To turn left: move left motor $CW$ and right motor $CCW$, with **PWM** adjusted by scale factor 0.6
  - To turn right: move left motor $CCW$ and right motor $CW$, with **PWM** adjusted by scale factor 0.5

## 4.6 Bluetooth Control

The robot was controlled via the **Arduino's Bluetooth Low Energy (BLE)** capabilities. The firmware was developed such that new connections need to be authorized by successfully transmitting the correct pairing code. This follows the workflow shown in Figure 1.

In order to send movement commands, the connection and authentication process must be successfully complete to ensure secure access. A key part of this process was to provide the expected pairing code to the microcontroller.

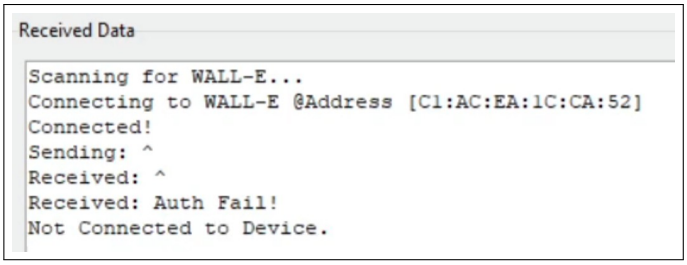


Figure 14: Unauthorized Bluetooth Connection

If unsuccesful, the robot would immediately disconnect and await an authorized user as shown in Figure 14.

The firmware is also designed such that movement commands are placed on a lower priority. Since **BLE** can be computationally expensive, we polled for new commands every 100 ms in the main loop. This was done to ensure that the control functionality is placed at highest priority, and we can account for the infrequent nature of user driven commands.

### 4.6.1 Robot Driver App

We developed a **Flask** application, which used `GET/POST` request **HTTP** methods to asynchronously request device information from the **HTML DOM** elements and send commands to the **Arduino** microcontroller.

This application was run from our computer and temporarily hosted to the Internet using the **ngrok** tool. This enabled us to access the robot via public URL from other devices (including smartphones).

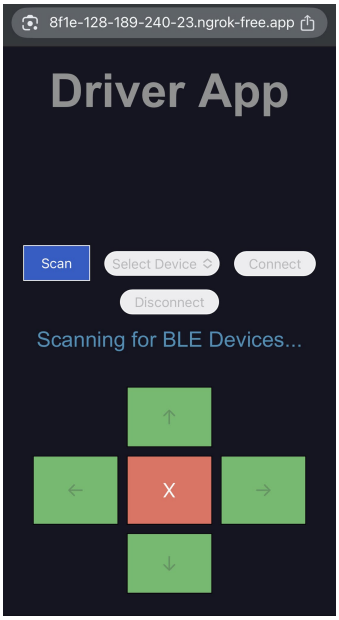


Figure 15: Robot Driver App on iPhone 14

The **Flask** application had the expected pairing code programmed in the source code such that users can easily connect to and control the robot. We programmed the `DRIVE, LEFT, RIGHT, BACK` commands to correspond to specific bytes to be transmitted via **BLE**.

This is shown in the code snippet in Listing 11 in the Appendix.

## 4.7 STM32L051K8T6 Microcontroller

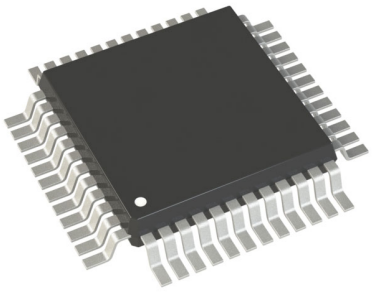

Figure 16: STM32L051K8T6 Microcontroller

The STM32L051K8T6 is a low-power microcontroller from STMicroelectronics, part of the STM32L0 series. It features an ARM Cortex-M0+ core running at up to 32 MHz, with 64 KB of flash memory and 8 KB of SRAM. For

our project, this microcontroller was used to interface with the HC-SR04 (section 4.7.1), TCS34725 (section 4.7.2), RC522 (section 4.7.3) and the CEM-1202 (section 4.7.4) modules. The pinout diagram is shown in Figure 17 (note that USART it was never used in the implementation).

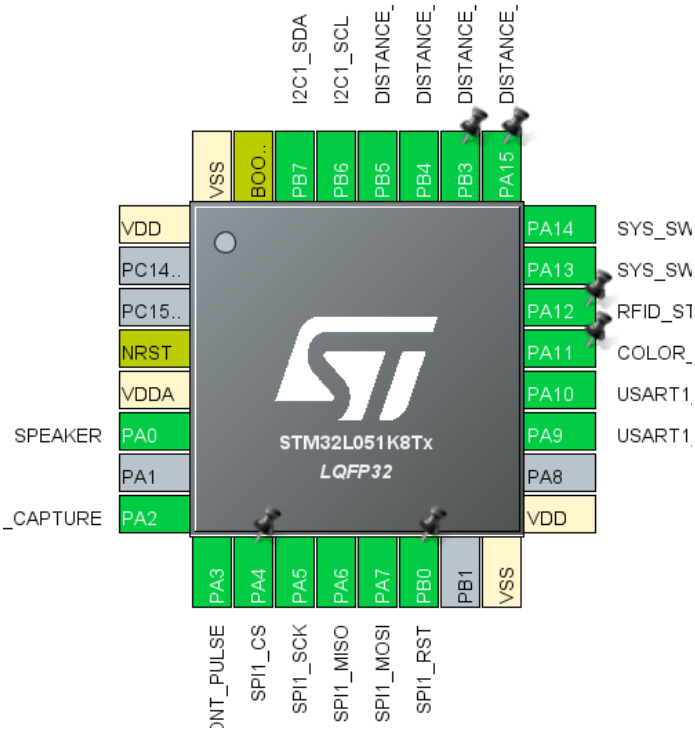


Figure 17: Zoomed In Pinout

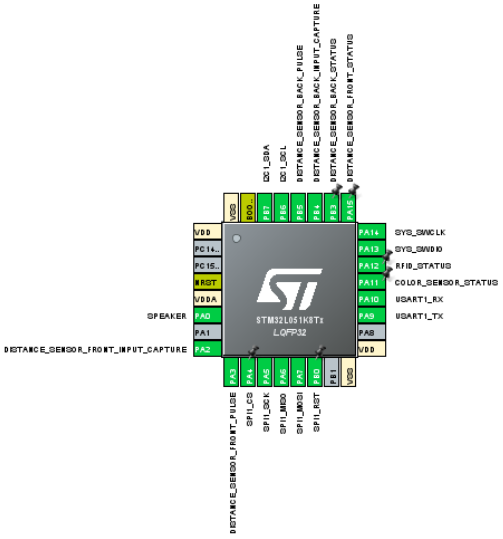

Figure 18: Zoomed Out Pinout

To streamline the firmware development process for the STM32, we utilized the STM32Cube IDE, provided by STMicroelectronics. This development environment facilitated faster progress by offering built-in tools and libraries, allowing us to focus on application logic rather than dealing with bare-metal programming.

### 4.7.1 Distance Sensing

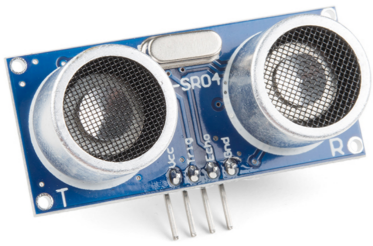

Figure 19: HC-SR04 Ultrasonic Ranging Module

As an extra feature, we decided to use two **HC-SR04** sensors for sensing proximity to objects. It operates by emitting an ultrasonic pulse from its transmitter and measuring the time it takes for the pulse to reflect off a nearby surface and return to the receiver. The module communicates via a simple TTL interface, requiring a 'trigger' signal to start the measurement and an 'echo' signal to indicate when the reflected pulse is received. By calculating the time interval between the trigger and echo signals, the distance to the object can be determined.

We decided to use timers and interrupt service routines to measure the time of this pulse. Specifically, we used the input capture mode, which would trigger on both rising and falling edges of the Echo signal, enabling precise

capture of timer values at each transition. By handling these events in interrupt service routines, the CPU remained free to execute other tasks concurrently with the distance measurement process.

A code sample for the firmware implementation of input capture interrupt are shown in Listing 7 in the Appendix. More information about input capture mode and its implementation can be found on this site, here. If the module sensed a distance smaller than 30cm, a basic GPIO pin would be set low to communicate to the Arduino of this status.

### 4.7.2 Color Sensing

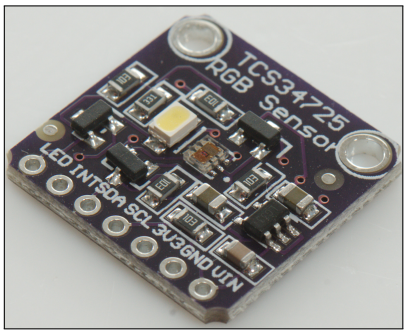


Figure 20: TCS34725 Light-to-Digital Sensor

We decided to integrate the **TCS34725** Color Sensor Module, responsible for detecting the color composition of a surface by measuring the intensity of red, green, blue, and clear light. It communicates via an $I^2C$ interface, and was implemented on the *STM32L051* microcontroller. The STM32 HAL libraries simplify $I^2C$ communication, as demonstrated in Listing 8.

The `ColorSensor_Handle` function is called within the main control loop. It reads the values from each color channel register (16 bits) and determines the most dominant color. If the red channel is the most prominent, the speaker is triggered to notify the user of the sensor's detection. When this would occur, a basic GPIO pin would be set low to communicate to the Arduino of this status.

### 4.7.3 RFID Security System

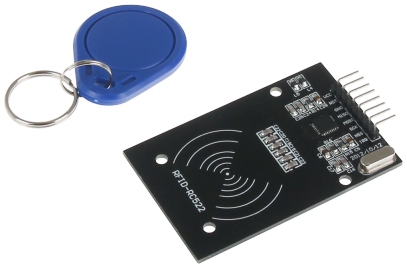

Figure 21: RC522 Contactless Reader IC

The **RC522** RFID Module was responsible for detecting and reading passive RFID tags using radio frequency communication at 13.56MHz. It operates over an SPI interface, allowing for fast and efficient data exchange with the *STM32L051* microcontroller. When a tag enters the RF field generated by the module's antenna, the RC522 initiates a handshake protocol and reads the tag's unique identifier (UID) stored in its internal memory.

In the case where the correct RFID tag would be scanned, a basic GPIO pin would be set low and the speaker would be triggered to communicate to the Arduino of this status. The remaining logic specific to our application is shown in Listing 9 in the Appendix.

#### 4.7.4 CEM-1302 Speaker/Buzzer

Figure 22: CEM-1302 Speaker/Buzzer

We used the CEM-1302 buzzer to indicate the following key events:

- RFID: Indicating successful and unsuccessful RFID card scans
- Ultrasonic Sensor: Alerting when the distance between the robot and an object is too small
- Color Sensor: Signaling when the color sensor detects a predominantly red color

When any of these events occurred, the speaker would either beep, or stay on, depending on the scenario. Code for the setup can be seen in Listing 10. The speaker was also developed on the *STM32L051*, using a square wave PWM running at 2.048kHz. GPIO pins for these events were used to communicate with the Arduino Nano for simplicity and speed.

### 4.8 Raspberry Pi Zero 2 W

The **Raspberry Pi Zero 2 W** is an affordable single-board computer designed for embedded systems and IoT applications. We leveraged its camera connector (CSI-2 interface) to interface with a **Freenove 5MP Camera**, as well as its Wi-Fi capabilities to wirelessly access it the **Raspberry Pi Connect** software.

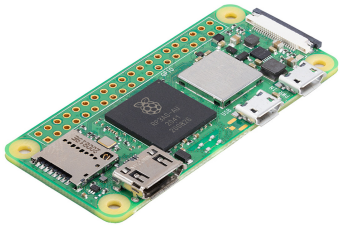

Figure 23: Raspberry Pi Zero 2 W

### 4.8.1 Camera App

We developed a **Python** application to display the live camera feed in a **Tkinter GUI** window, allowing the user to observe and record the operation of the robot. The user interface provided the ability to also take snapshots and store these in the **SD Card** storage.

The live feed was largely a result of scheduling an update function with **Tkinter**'s `after(...)` method, which allowed us to sample camera frames at a rate of 20 FPS.

Since the **Raspberry Pi Zero 2 W** had power and performance constraints (i.e. 512 MB of SDRAM), we ensured that the application was optimized for performance and iterated software versioning to minimize resource consumption. The camera was configured to capture frames at a resolution of 480 pixels to account for this.

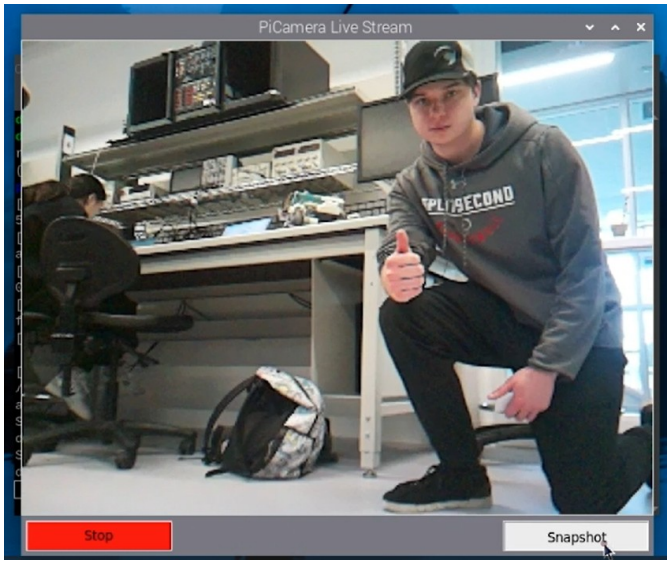


Figure 24: PiCamera Application

The class and methods for the camera display are shown in Listing 12 in the Appendix.

Early on in development, we switched from the **Kivy** framework. This provided smoother animations, but rendering was hardware accelerated. We were unconfident with the **Raspberry Pi Zero 2 W**'s ability to provide adequate performance with this, and ported the application to the current implementation.

### 4.8.2 AWS Bucket

Another constraint we considered during implementation was the inability to connect to the **Raspberry Pi** without powering on the robot. Having to access recording and snapshots would be quite cumbersome for the user.

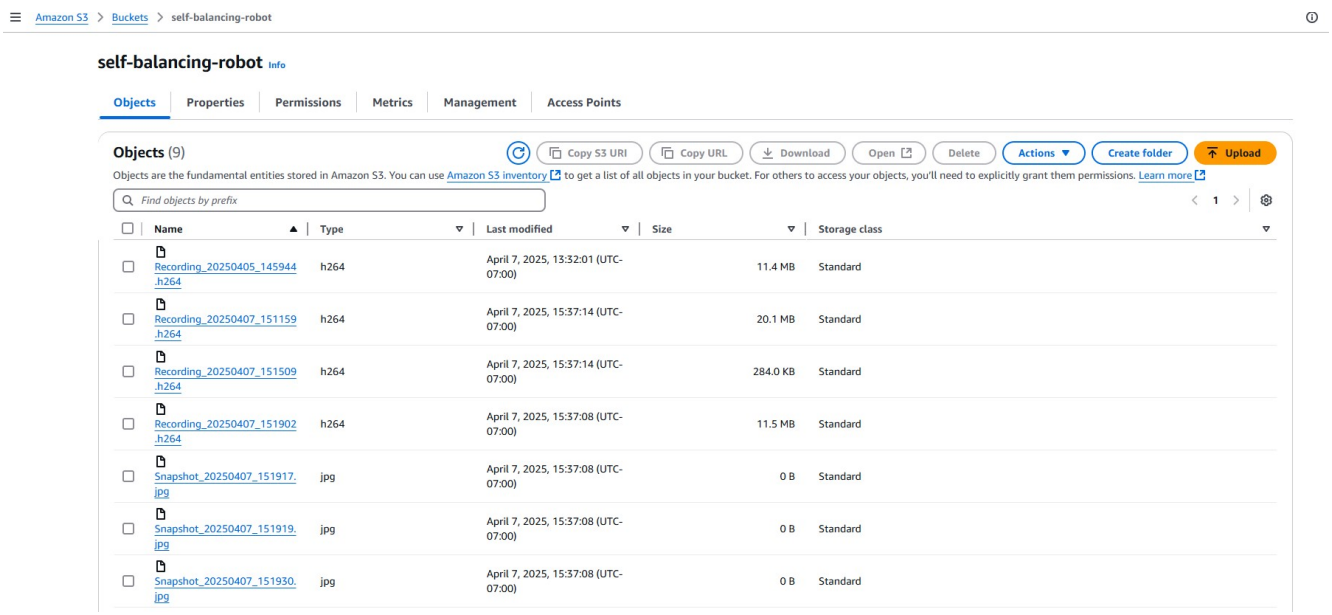


Figure 25: AWS S3 Bucket

We created an **AWS S3 Bucket** and uploaded the camera recordings and snapshots during operation. The **Boto** package enabled us to implement this in our **Python** application by leveraging the **AWS SDK**. We were able to download these files from the **AWS S3 Bucket** to our local machine for offline access.

# 5 Verification and Validation

## 5.1 Verification

To evaluate the core functionality of the robot, we leveraged the **Serial** and **BLE** capability of the **Arduino** microcontroller to both observe and modify **PID** I/O signals.

This was done to compare performance as we tweaked parameters.

### 5.1.1 Angle Measurement Accuracy

Early on in development, we realized that accurate angle measurements were critical to the robot's ability to successfully balance itself.

We developed a **Python** user interface to visualize the angle measurements from the **Arduino** in real-time, included in Listing 13 in the Appendix.

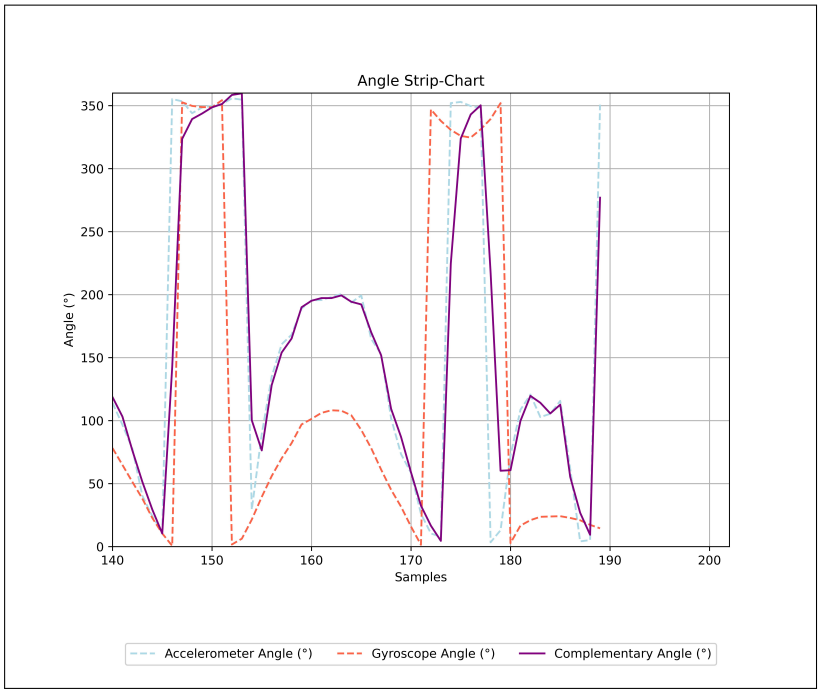


Figure 26: Angle Measurements (k = 0.6)

As shown in the example in Figure 26, we rotated the **Arduino** microcontroller within a predefined angle range and observed the complementary filter's performance with a given $k$ value.

From saving and comparing these logs, we observed trends in the angle measurements from the accelerometer and gyroscope. This enabled us to optimize the benefits of both sensor and was critical in ensuring that we were able to balance the robot successfully.

### 5.1.2 PID Tuning

Similarly, we developed a debugging tool in the **Arduino** firmware to tune **PID** parameters during operation. The source code for this is shown in Listing 6 in the Appendix.

When we were confident in the actual system level logic, we used the **BLE** capabilities to tune the following parameters.

- PID Gains ($K_p$, $K_i$, $K_d$)
- Complementary Filter Weight ($k$)
- Setpoint Angle ($\theta_{set}$)

During operation, we observed the effect of increasing / decreasing each of these on meeting the balancing requirements from Table 1.

| Parameter | Behavior When Increased | Behavior When Decreased |
|---|---|---|
| $K_p$ | Undershoot | Overshoot, Tipping over When $K_p$ Too High . |
| $K_i$ | Drifting, Undershoot | Degraded Stability, Oscillates Near Reference Angle. |
| $K_d$ | Drifting, Overshoot | Oscillates Near Reference Angle |
| $k$ | Faster, Smooth Response. Accumulates Drift. | Increases Noise Influence, Experiences Jitter. No Drift. |
| $\theta_{set}$ | Drifting in Forward Direction. | Drifting in Backward Direction. |

Table 3: PID Parameters and System Behavior

### 5.2 Validation

To evaluate the balancing performance, we placed the robot on the lab floor and allowed it to balance without external assistance for one minute. A ruler was placed on the ground to measure any drift over time. During testing, the robot was able to maintain balance for over five minutes and remained relatively stationary, drifting less than five centimeters. These results indicate that the system meets the balancing requirement.

To test angle management, we applied a force to the top of the pendulum to tilt the robot approximately 15 degrees and observed whether it could recover without tipping over. The robot successfully recovered from tilts of up to 20 degrees without oscillation. However, when the tilt angle exceeded 20 degrees, it was unable to recover from the disturbance. These results indicate that the system meets the angle management requirement within the expected operating range.

## 6 Conclusions and Future Work

The implementation of the PID controller is effective, allowing the robot to balance with reasonable resistance to external disturbances. However, one improvement would be to utilize the encoder and implement an additional, cascaded PID controller. This could help reduce oscillations and improve the robot's ability to remain stationary. It may also enhance the angle management — since the current system requires a relatively high I-term to maintain balance, it has a degraded stability under significant external disturbances. By introducing a cascaded PID controller, the I term in the main controller could be reduced, leading to improved stability and better angle management.

In terms of structural design, we raised the third layer to increase the linearity of the system's step response within its controllable region. While this improves linearity, it also narrows the linear region and makes the robot more prone to external disturbances. One possible improvement is to replace the current motors with ones that have less mechanical slack, which would provide a more responsive system. This would allow us to lower the third layer, enhancing angle management while maintaining linear behavior.

Additionally, although the distance sensors are mounted using L-stands, we did not cut the mounting holes on the second layer. Instead, the stands were taped in place, which led to noticeable shakiness during testing. This affected the stability and observability of the system. If more time were available, cutting proper mounting holes in the second layer would help makes the sensors more secure and improve the overall system performance.

## 7 References

[1] T. N. Soe and A. S. Khaing, "Mathematical model for two wheeled balancing robot using pid control algorithm," *International Journal of Latest Technology in Engineering, Management & Applied Science (IJLTEMAS)*, vol. VIII, no. VIII, pp. 1–5, Aug. 2019, Technological University (Mawlamyine), ISSN: 2278-2540. [Online]. Available: `https://www.ijltemas.in`.

# 8 Appendices

Include sections that do not fit well in the main body of the report but help to show the development progress and justify your decisions. These might include (but may not be restricted to) the following:

## 8.1 Appendix: Budget

| Part | Manufacturer PN | Digikey PN | Link | Datasheet | Description | Quantity | Cost/Unit | Total Cost |
|---|---|---|---|---|---|---|---|---|
| Distance Sensor | SNS-HCSR04 | 1188-SNS-HCSR04-ND | Link | Datasheet | SENSOR OPTICAL 20-100CM TTL | 2 | 2.81 | 5.62 |
| Raspberry Pi Zero 2 | SC1176 | 2648-SC1176-ND | Link | Datasheet | SBC 1.0GHZ 4 CORE 512MB RAM | 1 | 22.75 | 22.75 |
| RFID Reader Module | CN0090 | 4411-CN0090-ND | Link | Datasheet | RC522 RFID MODULE | 1 | 8.49 | 8.49 |
| High Current Voltage Regulator 5V | LD1085V50 | 497-3436-5-ND | Link | Datasheet | IC REG LINEAR 5V 3A TO220 | 1 | 2.55 | 2.55 |
| | | | | | | | | |
| Color Sensor I2C | TCS34725 | N/A | Link | Datasheet | TCS34725 Color Sensor Recognition Modul | 1 | 3.85 | 3.85 |
| Raspberry Pi Camera | N/A | N/A | Link | None | Freenove 5MP Camera for Raspberry Pi 5 4 | 1 | 16.95 | 16.95 |
| | | | | | | | | 60.21 |

Figure 27: Extra Features Budget Rundown

## 8.2 Appendix: Technical Calculations and Simulations

| Sensor | Pros | Cons |
|---|---|---|
| **Accelerometer** | • accurate gravitational readings<br>• zero-mean noise | • high noise variance (especially with motor vibrations) |
| **Gyroscope** | • lower noise over short time periods | • accumulates bias over time |

Table 4: IMU Sensors: Pros and Cons

## 8.3 Appendix: Drawings, Schematics, and Blueprints

## 8.4 Appendix: Datasheets / Product Specifications

Supporting Datasheets and Technical Documentation for Core Features (does not include battery related components, mechanical components, or generic electrical components):

1. Arduino Nano 33 IoT (ABX00069) Datasheet
2. Pololu 37D Metal Gearmotor Specifications
3. Texas Instruments DRV8833 Dual H-Bridge Motor Driver Datasheet

Supporting Datasheets and Technical Documentation for Additional Features:

1. HC-SR04 Ultrasonic Distance Sensor
2. TCS34725 Color Sensor
3. Raspberry Pi Zero 2 W
4. CN0090 Accelerometer Board
5. L293D Motor Driver IC
6. Freenove 5MP Camera (FNK0056)
7. CEM-1302 Speaker/Buzzer

## 8.5 Appendix: Prototypes

## 8.6 Appendix: Sample Code

```
ANGLES Angles = {0, 0, 0}; // Accelerometer, Gyroscope, Complementary
void getAngles(ANGLES &Angles) {
  float currAccel, currGyro, currComplementary;
  float sampleTime;

  if (!IMU.gyroscopeAvailable()) return;
  IMU.readGyroscope(gx, gy, gz);

  if (!IMU.accelerationAvailable()) return;
  IMU.readAcceleration(ax, ay, az);

  currAccel = atan2(az, ay) * (180 / PI);
  currAccel = (currAccel - 90) + ACCELEROMETER_OFFSET;

  sampleTime = 1.0 / IMU.gyroscopeSampleRate();

  currGyro = prevGyro + gx * sampleTime;

  prevAngle = prevComplementary;
  currComplementary = k * (prevComplementary + gx * sampleTime) + (1 - k) * currAccel;

  /* Update Time Variables */
  t_n = millis(); // Current Time in Milliseconds
  dt = (t_n - t_n1) / 1000.0; // Time Difference in Seconds
  t_n1 = t_n; // Assign Current Time to Previous Time

  /* Update Angles */
  Angles.Accelerometer = currAccel;
  Angles.Gyroscope = currGyro;
  Angles.Complementary = currComplementary;

  /* Assign Previous Angles */
  prevGyro = currGyro;
  prevComplementary = currComplementary;
}
```

Listing 1: Arduino IMU Complementary Filter Firmware Implementation

```
void balanceRobot(int bleDirection) {
    // Get Measured Angle
    measuredAngle = Angles.Complementary; // Complementary Filter
    errorAngle = setpointAngle - measuredAngle; // e_t = r_t - y_t
    errorDifference = (errorAngle - prevErrorAngle) / dt; // e_t - e_(t-1) / dt
    errorAccumulation += (errorAngle * dt);

    // Reset Accumulated Error Value ∑e_t
    if (digitalRead(DISABLE_INTEGRAL_BUTTON) == LOW) errorAccumulation = 0;

    // Calculate Control Signal : u_t = Kp * e_t + Ki * ∑e_t + Kd * (e_t - e_(t-1) / dt)
    u_t = (Kp * errorAngle) + (Ki * errorAccumulation) + (Kd * errorDifference);
    drive(u_t, errorAngle);

    prevErrorAngle = errorAngle; // Update Previous Error Value
    return;
}
```

Listing 2: Arduino PID Firmware Implementation

```
/*
 * Fast Decay: Hold Primary Input at PWM Duty Cycle, Secondary Input Low.
 */
void moveFastDecay(XIN &motor, DirPWM dir, float dutyCycle) {
    if (dir == CW) {
        motor.Pin1->write(dutyCycle);
        motor.Pin2->write(0);
    } else if (dir == CCW) {
```

```
        motor.Pin1->write(0);
        motor.Pin2->write(dutyCycle);
    }
}

/*
 * Slow Decay: Hold Primary Input High, Secondary Input at PWM Duty Cycle.
 */
void moveSlowDecay(XIN &motor, DirPWM dir, float dutyCycle) {
    if (dir == CW) {
        motor.Pin1->write(1);
        motor.Pin2->write(1-dutyCycle);
    } else if (dir == CCW) {
        motor.Pin1->write(1-dutyCycle);
        motor.Pin2->write(1);
    }
}
```

Listing 3: Arduino Motor PWM Firmware Implementation

```
void drive(float u_t, float errorAngle) {
    currDutyCycle = normalizePWM(u_t, 0);

    if (millis() - startTime >= 1500) {
        startTime = currTime; // Reset Start Time
        setpointAngle = SETPOINT_0;
        bleDirection = IDLE; // Stop Robot
    }

    if (redAlert) {
        startTime = currTime; // Reset Start Time
        setpointAngle = SETPOINT_0 + 1.5 * ANGLE_TILT;
        bleDirection = REVERSE;
    }

    switch (bleDirection) {
        case FORWARD:
            if (forwardAlert) setpointAngle = SETPOINT_0 + 0.5 * ANGLE_TILT;

            if (u_t > 0) moveForward(currDutyCycle);
            else moveReverse(currDutyCycle);
        break;
        case REVERSE:
            if (reverseAlert) setpointAngle = SETPOINT_0 - ANGLE_TILT;

            if (u_t > 0) moveForward(currDutyCycle);
            else moveReverse(currDutyCycle);
        break;
        case LEFT:
            if ((directionCount++ % 3 == 0) || (abs(errorAngle) > MAX_ERR_ANGLE)) {
                if (u_t > 0) moveForward(currDutyCycle);
                else moveReverse(currDutyCycle);
            } else {
                turnLeft(currDutyCycle, 0.6);
            }
        break;
        case RIGHT:
            if ((directionCount++ % 3 == 0) || (abs(errorAngle) > MAX_ERR_ANGLE)) {
                if (u_t > 0) moveForward(currDutyCycle);
                else moveReverse(currDutyCycle);
            } else {
                turnRight(currDutyCycle, 0.5);
            }
        break;
        default:
            if (u_t > 0) moveForward(currDutyCycle);
            else moveReverse(currDutyCycle);
        break;
    }
    return;
}
```

Listing 4: Arduino Movement Firmware Implementation

```
void setupBLE() {
  pinMode(LED_BUILTIN, OUTPUT); // Init Built-in LED to Indicate Connection Status

  if (!BLE.begin()) {
    Serial.println("Starting BLE Failed!");
    while (1);
  }

  // Set Local Name and Device Name
  BLE.setLocalName("WALL-E");
  BLE.setDeviceName("WALL-E");

  BLE.setAdvertisedService(customService);
  customService.addCharacteristic(customCharacteristic);  // Add the Characteristic to the
      Service
  BLE.addService(customService); // Add the Service to the BLE Device

  // Callback Event Handlers
  BLE.setEventHandler(BLEConnected, connectBLE);
  BLE.setEventHandler(BLEDisconnected, disconnectBLE);
  customCharacteristic.setEventHandler(BLEWritten, rxBLE);

  BLE.advertise(); // Advertising the BLE Device

  while (!BLE.connected()) BLE.poll(); // Wait for Connection
  customCharacteristic.writeValue("Enter Code:"); // Send Pairing Prompt
}

void connectBLE(BLEDevice central) {
  digitalWrite(LED_BUILTIN, HIGH);
  pairPrompted = false; // Reset Pairing Prompt Flag
}

void disconnectBLE(BLEDevice central) {
  digitalWrite(LED_BUILTIN, LOW);
  isAuthenticated = false; // Reset Authentication Status
}

void authenticateBLE() {
  if (strcmp(buffBLE, CODE) == 0) {
    isAuthenticated = true;
    customCharacteristic.writeValue("Auth Success!");
    isAuthenticated = true; // Set Authentication Status
  } else {
    customCharacteristic.writeValue("Auth Fail!");
    BLE.disconnect(); // Disconnect if Authentication Fails
  }

}

void rxBLE(BLEDevice central, BLECharacteristic characteristic) {
  int length = characteristic.valueLength();
  const unsigned char* receivedData = characteristic.value();

  memcpy(buffBLE, receivedData, length);
  buffBLE[length] = '\0'; // Null-Terminated

  if (isAuthenticated) {
    // updateParamBLE(buffBLE); // Update PID Parameters Based On BLE Input
    changeDirection(buffBLE); // Change Direction Based On BLE Input
  } else {
    authenticateBLE(); // Authenticate Device
  }
  return;
}
```

Listing 5: Arduino BLE Firmware Implementation

```
void updateParamBLE(const char* bleBuff) {
  char paramType[STD_BUFFSIZE] = {0};
  char valueStr[STD_BUFFSIZE] = {0};
  float newValue = 0.0f;

  char returnStr[STD_BUFFSIZE] = {0};

  char cmd[STD_BUFFSIZE] = {0};
  strcpy(cmd, bleBuff); // Copy Command to Local Buffer

  int start = 0;
  while ((cmd[start] == ' ') || (cmd[start] == '\t')) start++; // Skip Leading Whitespace

  if (
    strncmp(&cmd[start], "k=", 2) == 0 ||
    strncmp(&cmd[start], "set=", 4) == 0 ||
    strncmp(&cmd[start], "Kp=", 3) == 0 ||
    strncmp(&cmd[start], "Ki=", 3) == 0 ||
    strncmp(&cmd[start], "Kd=", 3) == 0
  ) {
    char* equalsPos = strchr(&cmd[start], '='); // Find '=' Position
    if (!equalsPos) return;

    int paramLength = equalsPos - &cmd[start]; // Length of Parameter Name
    strncpy(paramType, &cmd[start], paramLength); // Copy Parameter Name
    paramType[paramLength] = '\0'; // Null-Termination

    strcpy(valueStr, equalsPos + 1); // Copy Value String
    newValue = atof(valueStr); // Convert Value String to Float

    if (strcmp(paramType, "k") == 0) k = newValue;
    else if (strcmp(paramType, "set") == 0) SETPOINT_0 = newValue;
    else if (strcmp(paramType, "Kp") == 0) Kp = newValue;
    else if (strcmp(paramType, "Ki") == 0) Ki = newValue;
    else if (strcmp(paramType, "Kd") == 0) Kd = newValue;
    else return; // Invalid Command

    sprintf(returnStr, "NewVal: %s=%s", paramType, valueStr);
    customCharacteristic.writeValue(returnStr);
  }
  return; // Exit Function
}
```

Listing 6: Arduino BLE PID Tuning

```
void DistanceSensor_InputCaptureInterrupt(distancesensor* sensor)
{
    if (HAL_GPIO_ReadPin(sensor->icGPIOPort, sensor->icGPIOPin)) {
        sensor->IC_Value1 = HAL_TIM_ReadCapturedValue(sensor->timer, TIM_CHANNEL_1); // First
            rising edge
        HAL_TIM_PWM_Stop(sensor->timer, TIM_CHANNEL_2);
    }
    else {
        sensor->IC_Value2 = HAL_TIM_ReadCapturedValue(sensor->timer, TIM_CHANNEL_1); // Second
            rising edge
        if (sensor->IC_Value2 > sensor->IC_Value1) {
            sensor->timeDifference = sensor->IC_Value2 - sensor->IC_Value1;
        }
        else {
            sensor->timeDifference = (TIM_PERIOD + 1 - sensor->IC_Value1) + sensor->IC_Value2;
                // Handle overflow
        }

        HAL_TIM_PWM_Start(sensor->timer, TIM_CHANNEL_2);
        __HAL_TIM_SetCounter(sensor->timer, 65535);

        DistanceSensor_Handle(sensor);
    }
}

```

```
// On main.c:

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM21) {
        DistanceSensor_InputCaptureInterrupt(&Front);
    }
    else if (htim->Instance == TIM22) {
        DistanceSensor_InputCaptureInterrupt(&Back);
    }

}
```

Listing 7: STM32 HC-SR04 Firmware Implementation

```
#include "colorsensor.h"

#define TCS3472_SLAVE_ADDRESS 0x29 << 1
#define TCS3472_EXPECTED_ID 0x44

#define TCS3472_ID_REG 0x12

#define TCS3472_WTIME_REG 0x03
#define TCS3472_ATIME_REG 0xF6
#define TCS3472_ENABLE_REG 0x00

#define TCS3472_CDATAL_REG 0x14
#define TCS3472_RDATAL_REG 0x16
#define TCS3472_GDATAL_REG 0x18
#define TCS3472_BDATAL_REG 0x1A

extern speaker Speaker;

void ColorSensor_Init(colorsensor* sensor, I2C_HandleTypeDef* i2c_handle) {
    sensor->i2c = i2c_handle;
    sensor->slave_address = TCS3472_SLAVE_ADDRESS;

    memset(sensor->rgb_data, 0, sizeof(sensor->rgb_data));  // Clear RGB data
    sensor->enabled = true;
    sensor->count = 0;

    // Define initialization sequence
    uint8_t init_data[][2] = {
        { TCS3472_WTIME_REG  | 0x80, 0xFF },  // Wait time
        { TCS3472_ATIME_REG  | 0x80, 0xFF },  // Integration time
        { TCS3472_ENABLE_REG | 0x80, 0x0B },   // Power ON & enable RGBC
    };

    // Loop through initialization commands and send them
    for (size_t i = 0; i < sizeof(init_data) / sizeof(init_data[0]); i++) {
        if (HAL_I2C_Master_Transmit(sensor->i2c, sensor->slave_address, init_data[i], 2,
            HAL_MAX_DELAY) != HAL_OK)
            while(1);
    }

    HAL_GPIO_WritePin(COLOR_SENSOR_STATUS_GPIO_Port, COLOR_SENSOR_STATUS_Pin, GPIO_PIN_SET);
}


void ColorSensor_EnableStatus(colorsensor* sensor, bool enable)
{
    sensor->enabled = enable;
}


uint16_t ColorSensor_Read16(colorsensor* sensor, uint8_t reg) {
    uint8_t buffer[2];
    uint8_t command = reg | 0xA0;  // Add auto-increment bit

    // Send register address
    if (HAL_I2C_Master_Transmit(sensor->i2c, sensor->slave_address, &command, 1, HAL_MAX_DELAY)
        != HAL_OK) {
```

```
56          return 0; // Handle error
57      }
58
59      // Read 2 bytes from the sensor
60      if (HAL_I2C_Master_Receive(sensor->i2c, sensor->slave_address, buffer, 2, HAL_MAX_DELAY) !=
            HAL_OK) {
61          return 0; // Handle error
62      }
63
64      // Combine two bytes into a 16-bit value (LSB first)
65      return (uint16_t)(buffer[0] | (buffer[1] << 8));
66  }
67
68  void ColorSensor_ReadAll(colorsensor* sensor) {
69      sensor->rgb_data[0] = ColorSensor_Read16(sensor, TCS3472_CDATAL_REG); // TCS3472_CDATAL
70      sensor->rgb_data[1]   = ColorSensor_Read16(sensor, TCS3472_RDATAL_REG); // TCS3472_RDATAL
71      sensor->rgb_data[2] = ColorSensor_Read16(sensor, TCS3472_GDATAL_REG); // TCS3472_GDATAL
72      sensor->rgb_data[3]  = ColorSensor_Read16(sensor, TCS3472_BDATAL_REG); // TCS3472_BDATAL
73  }
74
75
76  color ColorSensor_CalculateColor(colorsensor* sensor)
77  {
78      uint16_t max_val = sensor->rgb_data[1];
79      color detected_color = RED;
80
81      if (sensor->rgb_data[2] > max_val) detected_color = GREEN;
82      if (sensor->rgb_data[3] > max_val) detected_color = BLUE;
83
84      return detected_color;
85  }
86
87  void ColorSensor_Handle(colorsensor* sensor)
88  {
89      ColorSensor_ReadAll(sensor);
90      color detected_color = ColorSensor_CalculateColor(sensor);
91
92      if (detected_color == RED)
93      {
94          HAL_GPIO_WritePin(COLOR_SENSOR_STATUS_GPIO_Port, COLOR_SENSOR_STATUS_Pin,
                GPIO_PIN_RESET);
95          if (!Speaker.hasFault)
96              Speaker_Start(&Speaker, COLOR_SENSOR_ID);
97      }
98      else
99      {
100         HAL_GPIO_WritePin(COLOR_SENSOR_STATUS_GPIO_Port, COLOR_SENSOR_STATUS_Pin, GPIO_PIN_SET);
101         if (Speaker.hasFault)
102             Speaker_Stop(&Speaker, COLOR_SENSOR_ID);
103     }
104
105
106 }
```

Listing 8: STM32 TCS34725 Firmware Implementation

```
1   #include "rfid.h"
2
3   #include "speaker.h"
4
5   #define SERIALNUM_1_IDLE 32
6
7   #define SERIALNUM_0_CORRECT_CARD 170
8   #define SERIALNUM_1_CORRECT_CARD 205
9   #define SERIALNUM_2_CORRECT_CARD 47
10  #define SERIALNUM_3_CORRECT_CARD 3
11  #define SERIALNUM_4_CORRECT_CARD 75
12
13  extern speaker Speaker;
14  extern UART_HandleTypeDef huart1;
15  extern char Data;
```

```
16
17 void RFID_Init(rfid* sensor) {
18     MFRC522_Init();
19     memset(sensor->prevSerialNum, 0, 5);
20     sensor->status = CARD_IDLE;
21
22     sensor->botEnabled = false;
23     sensor->initialSuccessfulCardTap = true;
24     sensor->initialFailedCardTap = true;
25
26     HAL_GPIO_WritePin(RFID_STATUS_GPIO_Port, RFID_STATUS_Pin, GPIO_PIN_SET);
27 }
28
29 rfid_card_status RFID_ValidateCard(rfid* sensor)
30 {
31     uint8_t serialNum[5];
32     MFRC522_Request(PICC_REQIDL, serialNum);
33     MFRC522_Anticoll(serialNum);
34
35     sensor->status = CARD_IDLE;
36
37     if ((serialNum[0] == 170 && serialNum[1] == 205 && serialNum[2] == 47 && serialNum[3] == 3
           && serialNum[4] == 75) ||
38             (sensor->prevSerialNum[0] == 170 && sensor->prevSerialNum[1] == 205 &&
                   sensor->prevSerialNum[2] == 47 && sensor->prevSerialNum[3] == 3 &&
                   sensor->prevSerialNum[4] == 75))
39     {
40         sensor->status = CARD_SUCCESS;
41
42     }
43     else if (!(serialNum[0] | (!(serialNum[1] == 32 || serialNum[1] == 0)) | serialNum[2] |
           serialNum[3] | serialNum[4]))
44     {
45         if (!(sensor->prevSerialNum[0] | sensor->prevSerialNum[1] | sensor->prevSerialNum[2] |
               sensor->prevSerialNum[3] | sensor->prevSerialNum[4]))
46             sensor->status = CARD_IDLE;
47         else
48         {
49             sensor->status = CARD_FAIL;
50         }
51
52     }
53     else
54     {
55         sensor->status = CARD_FAIL;
56     }
57
58
59     for (uint8_t i = 0; i < 5; i++)
60     {
61         sensor->prevSerialNum[i] = serialNum[i];
62     }
63
64     return sensor->status;
65 }
66
67 void RFID_SecurityLogic(rfid* sensor)
68 {
69     rfid_card_status cardStatus = RFID_ValidateCard(sensor);
70
71     switch (cardStatus) {
72         case CARD_SUCCESS:
73             if (sensor->initialSuccessfulCardTap)
74             {
75
76                 Speaker_SetAutoReload(&Speaker, 488);
77                 Speaker_Beep(&Speaker, 150, 0, 1);
78                 sensor->initialSuccessfulCardTap = false;
79                 sensor->initialFailedCardTap = true;
80                 sensor->botEnabled = !sensor->botEnabled;
81                 HAL_GPIO_WritePin(RFID_STATUS_GPIO_Port, RFID_STATUS_Pin, (GPIO_PinState)
```

```
                    !sensor->botEnabled);
            }
            break;

        case CARD_FAIL:
            if (sensor->initialFailedCardTap)
            {
                HAL_Delay(200);
                if (RFID_ValidateCard(sensor) != CARD_FAIL)
                    break;
                Speaker_SetAutoReload(&Speaker, 488 * 4);
                Speaker_Beep(&Speaker, 150, 50, 4);
                sensor->initialSuccessfulCardTap = true;
                sensor->initialFailedCardTap = false;


            }
            break;

        case CARD_IDLE:
            sensor->initialSuccessfulCardTap = true;
            sensor->initialFailedCardTap = true;
            break;

        // Optional: Default case if no case matches
        default:
            // Code to execute if none of the above cases match
            break;
    }

}
```

Listing 9: STM32 RC522 Firmware Implementation

```
#include "speaker.h"


#define CLK_SPEED 32000000
#define DEFAULT_AUTORELOAD 488

void Speaker_Init(speaker* speaker, rfid* rfid_struct, TIM_HandleTypeDef* timer)
{
    speaker->rfid_sensor = rfid_struct;
    speaker->timer = timer;

    speaker->hasFault = false;
    speaker->beepLengthOn = 0;
    speaker->beepLengthPeriod = 0;
    speaker->wantedNumBeeps = 0;
    speaker->currentNumBeeps = 0;
    speaker->timerCounter = 0;

    for (uint8_t i = 0; i < sizeof(speaker->featureFault) / sizeof(speaker->featureFault[0]);
        i++)
    {
        speaker->featureFault[i] = false;
    }


}

void Speaker_Start(speaker* speaker, uint8_t ID)
{

    speaker->featureFault[ID] = true;
    if ((speaker->featureFault[0] || speaker->featureFault[1] || speaker->featureFault[2]) &&
        speaker->rfid_sensor->botEnabled)
    {
        speaker->hasFault = true;
        Speaker_SetAutoReload(speaker, DEFAULT_AUTORELOAD);
        HAL_TIM_PWM_Start(speaker->timer, TIM_CHANNEL_1);
```

```
36      }
37
38  }
39
40  void Speaker_Stop(speaker* speaker, uint8_t ID)
41  {
42      speaker->featureFault[ID] = false;
43      if (!(speaker->featureFault[0] || speaker->featureFault[1] || speaker->featureFault[2]))
44      {
45          speaker->hasFault = false;
46          HAL_TIM_PWM_Stop(speaker->timer, TIM_CHANNEL_1);
47      }
48  }
49
50
51
52  bool Speaker_Beep(speaker* speaker, uint16_t length_on_ms, uint16_t length_off_ms, uint8_t
        numBeeps)
53  {
54
55      if (speaker->hasFault)
56          return false;
57
58
59      speaker->timerCounter = 0;
60      speaker->currentNumBeeps = 0;
61
62
63
64      speaker->beepLengthOn = length_on_ms * 1000 / __HAL_TIM_GET_AUTORELOAD(speaker->timer);
65      speaker->beepLengthPeriod =speaker->beepLengthOn + length_off_ms * 1000 / __
            HAL_TIM_GET_AUTORELOAD(speaker->timer);
66      speaker->wantedNumBeeps = numBeeps;
67
68      //speaker->beepLength = length_ms * CLK_SPEED / (speaker->timer->Instance->PSC);
69
70      HAL_TIM_Base_Start_IT(speaker->timer);
71      HAL_TIM_PWM_Start(speaker->timer, TIM_CHANNEL_1);
72      return true;
73  }
74
75  void Speaker_BeepInterrupt(speaker* speaker)
76  {
77      if (speaker->currentNumBeeps < speaker->wantedNumBeeps)
78      {
79
80          if (speaker->timerCounter == speaker->beepLengthOn)
81          {
82              HAL_TIM_PWM_Stop(speaker->timer, TIM_CHANNEL_1);
83              __HAL_TIM_ENABLE(speaker->timer);
84          }
85          else if (speaker->timerCounter >= speaker->beepLengthPeriod)
86          {
87              speaker->currentNumBeeps++;
88              speaker->timerCounter = 0;
89
90              if (speaker->currentNumBeeps < speaker->wantedNumBeeps)
91              {
92                  HAL_TIM_PWM_Start(speaker->timer, TIM_CHANNEL_1);
93
94              }
95
96          }
97          speaker->timerCounter++;
98      }
99      else
100     {
101         HAL_TIM_PWM_Stop(speaker->timer, TIM_CHANNEL_1);
102         HAL_TIM_Base_Stop_IT(speaker->timer);
103     }
104 }
```

```
105
106 void Speaker_SetAutoReload(speaker* speaker, uint16_t value)
107 {
108     __HAL_TIM_SET_AUTORELOAD(speaker->timer, value);
109     __HAL_TIM_SET_COMPARE(speaker->timer, TIM_CHANNEL_1, value / 2);
110 }
```

Listing 10: STM32 Speaker Firmware Implementation

```
1     import asyncio
2     import threading
3
4     from bleak import BleakScanner, BleakClient, BleakError
5     from flask import Flask, request, jsonify, render_template
6
7     class BLEManager:
8         # Define UUIDs for BLE Service and Characteristic
9         SERVICE_UUID = "00000000-5EC4-4083-81CD-A10B8D5CF6EC"
10        CHARACTERISTIC_UUID = "00000001-5EC4-4083-81CD-A10B8D5CF6EC"
11        CODE = "0095" # Authentication Code
12
13        def __init__(self):
14            self.devices = []
15            self.client = None
16            self.is_connected = False
17
18            self.loop = asyncio.new_event_loop()
19            self.start_loop()
20
21        def start_loop(self):
22            threading.Thread(target = self._run_loop, daemon = True).start()
23
24        def _run_loop(self):
25            asyncio.set_event_loop(self.loop)
26            self.loop.run_forever()
27
28        def run_async(self, coro):
29            return asyncio.run_coroutine_threadsafe(coro, self.loop)
30
31        async def scan_devices(self):
32            self.devices = []
33            scanned = await BleakScanner.discover()
34
35            self.devices = [
36                {"name" : device.name, "address" : device.address} for device in scanned if
                      device.name and ("WALL-E" in device.name)
37            ] # Filter BLE Devices
38            return self.devices
39
40        async def connect_device(self, address):
41            """Connect to BLE Device."""
42            if self.client and self.client.is_connected:
43                self.is_connected = True
44                return True
45
46            self.client = BleakClient(address)
47            try:
48                await self.client.connect()
49                await self.client.write_gatt_char(BLEManager.CHARACTERISTIC_UUID,
                      BLEManager.CODE.encode("utf-8"))
50                self.is_connected = self.client.is_connected
51                return self.is_connected
52            except BleakError:
53                self.is_connected = False
54                return False
55
56        async def disconnect_device(self):
57            """Disconnect from BLE Device."""
58            if self.client and self.client.is_connected:
59                await self.client.disconnect()
60            self.is_connected = False
```

```
61
62          async def send_cmd(self, cmd):
63              """Schedule Send Command Coroutine Using Button's Custom Command."""
64              if not (self.client and self.client.is_connected):
65                  return False, "Not Connected to BLE"
66              try:
67                  await self.client.write_gatt_char(BLEManager.CHARACTERISTIC_UUID,
                        cmd.encode("utf-8"))
68                  return True, f"Sent Command: {cmd}"
69              except BleakError as e:
70                  return False, f"Error Sending Command: {e}"
71
72      class RobotDriverApp:
73          def __init__(self):
74              self.app = Flask("Robot Driver App")
75              self.ble_manager = BLEManager()
76              self._register_routes()
77
78          def _register_routes(self):
79              @self.app.route("/")
80              def index():
81                  return render_template("index.html")
82
83              @self.app.route("/scan", methods = ["GET"])
84              def scan():
85                  future = self.ble_manager.run_async(self.ble_manager.scan_devices())
86                  devices = future.result(timeout = 30) # Wait for 30 Seconds
87                  return jsonify(devices)
88
89              @self.app.route("/connect", methods = ["POST"])
90              def connect():
91                  data = request.get_json()
92                  if not data or "deviceAddress" not in data:
93                      return jsonify({"error": "No Address Provided"}), 400
94                  future =
                        self.ble_manager.run_async(self.ble_manager.connect_device(data["deviceAddress"]))
95                  success = future.result(timeout = 15) # Wait for 15 Seconds
96                  return (jsonify({"status": "Connected"}) if success
97                          else (jsonify({"status": "Failed"}), 400))
98
99              @self.app.route("/disconnect", methods = ["GET"])
100             def disconnect():
101                 future = self.ble_manager.run_async(self.ble_manager.disconnect_device())
102                 future.result(timeout = 10) # Wait for 5 Seconds
103                 return jsonify({"status": "Disconnected"})
104
105             @self.app.route("/move", methods = ["POST"])
106             def move():
107                 data = request.get_json()
108                 if not data or "command" not in data:
109                     return jsonify({"error": "No Command Provided"}), 400 # Bad Request
110
111                 future = self.ble_manager.run_async(
112                     self.ble_manager.send_cmd(data["command"])
113                 )
114                 success, message = future.result(timeout = 10) # Wait for 5 Seconds
115
116                 if success:
117                     return jsonify({"status": "OK", "msg": message})
118                 else:
119                     return jsonify({"status": "Error", "msg": message}), 400 # Bad Request
120
121         def run(self):
122             self.app.run(host = "0.0.0.0", port = 5000, debug = False)
123
124     if __name__ == "__main__":
125         RobotDriverApp().run()
```

Listing 11: Robot Driver App

```
1 from picamera2 import Picamera2
```

```
from picamera2.encoders import H264Encoder
from libcamera import Transform

class CameraDisplay(tk.Frame):
    SAMPLE_RATE = 1.0 / 50.0 # Update Delay ~20 FPS (1000ms/20 = 50ms)
    def __init__(self, parent, *args, **kwargs):
        super().__init__(parent, *args, **kwargs)

        self.filename = ""
        self.camera = Picamera2() # Init Camera

        self.snapshot_dir: str = os.path.join(os.path.dirname(os.path.abspath(__file__)),
            "Snapshots")
        self.video_dir: str = os.path.join(os.path.dirname(os.path.abspath(__file__)), "Videos")

        os.makedirs(self.snapshot_dir, exist_ok = True)
        os.makedirs(self.video_dir, exist_ok = True)

        # Force RGB888 Output
        self.config = self.camera.create_video_configuration(
            main = {
                "size": (640, 480), # Default Size
                # "size": (1280, 720), # HD
                # "size": (1920, 1080), # Full HD
                "format": "RGB888"
            },
            transform = Transform(hflip = False, vflip = False) # For Tkinter Display
        )
        self.image_label = tk.Label(self)
        self.image_label.pack()

    def start_recording(self):
        self.camera.configure(self.config)
        self.camera.start()

        self.filename = f"Recording_{dt.datetime.now().strftime('%Y%m%d_%H%M%S')}"
        input_file = os.path.join(self.video_dir, f"{self.filename}.h264")

        self.camera.start_recording(H264Encoder(bitrate = 10000000), input_file)

    def stop_recording(self):
        if self.filename == "":
            return

        self.camera.stop_recording()
        self.camera.stop()
        self.filename = ""

    def take_snapshot(self):
        if self.filename == "":
            self.camera.configure(self.config)
            self.camera.start()

        # Capture Current Frame
        frame = self.camera.capture_array()
        if frame is None or frame.size == 0:
            print("Failed to Capture Snapshot.")
            return

        # Swap Color Channels (BGR -> RGB)
        frame = frame[..., ::-1]

        snapshot_name = f"Snapshot_{dt.datetime.now().strftime('%Y%m%d_%H%M%S')}.jpg"
        snapshot_path = os.path.join(self.snapshot_dir, snapshot_name)

        # Convert to PIL Image
        image = PILImage.fromarray(frame)
        image.save(snapshot_path) # Save Image
        print(f"Snapshot Saved to {snapshot_path}")

        if self.filename == "":
```

```
            self.camera.stop()

    def update(self):
        if self.filename == "":
            return

        frame = self.camera.capture_array()
        if (frame is not None) and (frame.size != 0):
            frame = frame[..., ::-1] # BGR -> RGB

            # Convert the PIL Image to an ImageTk.PhotoImage
            self.image_label.current = ImageTk.PhotoImage(
                image = PILImage.fromarray(frame) # Convert Numpy Array to PIL.Image
            )
            self.image_label.configure(image = self.image_label.current)

    def stop(self):
        self.camera.close()
```

Listing 12: PiCamera Display

```
    class StripChart:
    # SAMPLE_RATE = 0.250 # 250 ms
    SAMPLE_RATE = 1 # 1000 ms
    def __init__(self, master, conn = None, data_size = 25, ylim = 30):
        self.master = master
        self.conn = conn
        self.fig, self.ax = plt.subplots(figsize = (900 / 100, 755 / 100))

        self.data_size, self.ylim = data_size, ylim

        # Angle Data (Additional Storage Preferred Over DataFrame for Speed)
        self.sample_data = []
        self.accelerometer_data = []
        self.gyroscope_data = []
        self.complementary_data = []

        # DataFrame for Data Export
        self.start_time = None
        self.data_df = pd.DataFrame(
            columns = [
                'Time (PST)',
                'Accelerometer Angle (Deg)',
                'Gyroscope Angle (Deg)',
                'Complementary Angle (Deg)'
            ]
        )

        self.ax.set_title('Real-Time Angle Strip-Chart')
        self.ax.grid(True)

        self.ax.set_xlim(
            0, # Start at 0s
            # 1.25 * self.data_size * StripChart.SAMPLE_RATE * 0.1
            1.25 * self.data_size * StripChart.SAMPLE_RATE
        )
        self.ax.set_xticks(
            np.arange(
                0,  # Start at 0s
                # 1.25 * self.data_size * StripChart.SAMPLE_RATE * 0.1,
                1.25 * self.data_size * StripChart.SAMPLE_RATE,
                5  # Set Ticks to 5s Intervals
            )
        )

        self.ax.set_ylim(-self.ylim, self.ylim) # Angle Expected to be Between -180 Deg and 180
            Deg
        self.ax.set_yticks(np.arange(-self.ylim, self.ylim + 1, 5))

        self.ax.tick_params(axis = 'both', labelsize = 8)

```

```
50          self.ax.set_xlabel('Time (s)')
51          self.ax.set_ylabel('Angle (Deg)')
52
53          self.accelerometer_line, = self.ax.plot([], [], linestyle = 'dashed', lw = 1.5, label =
                'Accelerometer Angle (Deg)', color = 'lightblue')
54          self.gyroscope_line, = self.ax.plot([], [], linestyle = 'dashed', lw = 1.5, label =
                'Gyroscope Angle (Deg)', color = 'tomato')
55          self.complementary_line, = self.ax.plot([], [], lw = 1.5, label = 'Complementary Angle
                (Deg)', color = 'purple')
56
57          self.fig.subplots_adjust(bottom = 0.2)
58          self.ax.legend(
59              loc = 'upper center',
60              bbox_to_anchor = (0.5, -0.15), # Position Above Plot with Padding
61              ncol = 3, framealpha = 0.9,  # 3 Columns, Transparent Background
62              fontsize = 10,
63          )
64
65          self.canvas = FigureCanvasTkAgg(self.fig, master=self.master)
66          self.canvas_widget = self.canvas.get_tk_widget()
67
68          self.figures_dir: str = os.path.join(os.path.dirname(os.path.abspath(__file__)),
                "Figures")
69          self.logbook_dir: str = os.path.join(os.path.dirname(os.path.abspath(__file__)),
                "Logbook")
70
71          os.makedirs(self.figures_dir, exist_ok = True)
72          os.makedirs(self.logbook_dir, exist_ok = True)
73
74      def start(self, conn):
75          """Start StripChart."""
76          self.conn = conn
77          self.animation = animation.FuncAnimation(
78              fig = self.fig, # Figure
79              func = self.update, # Update Function
80              frames = self.data_size, # Number of Frames
81              blit = False, # Prevent Re-rendering Entire Plot
82              interval = StripChart.SAMPLE_RATE * 1000, # Delay in ms
83          )
84          self.canvas = FigureCanvasTkAgg(self.fig, master = self.master)
85          self.canvas_widget = self.canvas.get_tk_widget()
86
87      def update(self, frame):
88          """Update StripChart with New Data."""
89          self.rx_angle()
90
91          # Update Plot Data
92          # t_data = [t * 0.1 for t in self.sample_data]
93          t_data = [t for t in self.sample_data]
94          self.accelerometer_line.set_data(t_data, self.accelerometer_data)
95          self.gyroscope_line.set_data(t_data, self.gyroscope_data)
96          self.complementary_line.set_data(t_data, self.complementary_data)
97
98          # Adjust x Limits to Scroll Forward
99          if (len(self.sample_data) > 0) and (self.sample_data[-1] > self.data_size):
100             self.ax.set_xlim(
101                 t_data[0], # Start at First Data Point
102                 # t_data[0] + 1.25 * self.data_size * StripChart.SAMPLE_RATE * 0.1
103                 t_data[0] + 1.25 * self.data_size * StripChart.SAMPLE_RATE
104             ) # Display 25% More Data
105             self.ax.set_xticks(
106                 np.arange(
107                     t_data[0], # Start at First Data Point
108                     # t_data[0] + 1.25 * self.data_size * StripChart.SAMPLE_RATE * 0.1,
109                     t_data[0] + 1.25 * self.data_size * StripChart.SAMPLE_RATE,
110                     5  # Set Ticks to 5s Intervals
111                 )
112             )
113         else:
114             self.ax.set_xlim(
115                 0, # Start at 0s
```

```
116                 # 1.25 * self.data_size * StripChart.SAMPLE_RATE * 0.1
117                 1.25 * self.data_size * StripChart.SAMPLE_RATE
118             )
119             self.ax.set_xticks(
120                 np.arange(
121                     0,  # Start at 0s
122                     # 1.25 * self.data_size * StripChart.SAMPLE_RATE * 0.1,
123                     1.25 * self.data_size * StripChart.SAMPLE_RATE,
124                     5  # Set Ticks to 5s Intervals
125                 )
126             )
127 
128     def stop(self):
129         """Set Connection to None and Stop Updating."""
130         if self.conn is not None:
131             self.conn.close()
132             self.conn = None
133 
134             fig_name = f"Angle_Data_{dt.datetime.now().strftime('%Y%m%d_%H%M%S')}"
135             self.save_logs(fig_name)
136             self.save_fig(fig_name)
137 
138     def rx_angle(self):
139         """Read Angle Data from Serial Connection."""
140         def read_serial() :
141             """Read Data from Serial Connection."""
142             if (self.conn is not None) :
143                 return str(self.conn.readline().decode('ascii').strip('\r').strip('\n'))
144             else :
145                 return None
146 
147         if self.conn is None:
148             return
149 
150         try:
151             self.conn.write(b'A')
152         except serial.SerialException:
153             self.conn.reconnect()
154 
155         try:
156             arduinoStream: list = str(read_serial()).rstrip('\r').split(' ')
157             if len(arduinoStream) != 3:
158                  raise ValueError
159 
160             accelerometer_angle = float(arduinoStream[0])
161             gyroscope_angle = float(arduinoStream[1])
162             complementary_angle = float(arduinoStream[2])
163 
164             print(f"Accelerometer Angle: {accelerometer_angle} Deg")
165             print(f"Gyroscope Angle: {gyroscope_angle} Deg")
166             print(f"Complementary Angle: {complementary_angle} Deg")
167 
168             # Append Data
169             new_sample = (self.sample_data[-1] + 1) if self.sample_data else 0
170 
171             self.sample_data.append(new_sample)
172             self.accelerometer_data.append(accelerometer_angle)
173             self.gyroscope_data.append(gyroscope_angle)
174             self.complementary_data.append(complementary_angle)
175 
176             self.start_time = dt.datetime.now(pytz.timezone('US/Pacific')) if self.start_time is
                 None else self.start_time
177 
178             if self.data_df.empty:
179                 self.data_df = pd.DataFrame({
180                     'Time (PST)': self.start_time,
181                     'Accelerometer Angle (Deg)': accelerometer_angle,
182                     'Gyroscope Angle (Deg)': gyroscope_angle,
183                     'Complementary Angle (Deg)': complementary_angle
184                 }, index = [0])
185             else:
```

```
                self.data_df = pd.concat([
                    self.data_df,
                    pd.DataFrame([{
                        'Time (PST)': self.start_time + dt.timedelta(seconds = new_sample),
                        'Accelerometer Angle (Deg)': accelerometer_angle,
                        'Gyroscope Angle (Deg)': gyroscope_angle,
                        'Complementary Angle (Deg)': complementary_angle
                    }])], ignore_index = True
                )

        except serial.SerialException:
            self.conn.reconnect()
        except ValueError as value_error: # Parse Error
            display("Parse Error:", str(value_error))

        # Limit Size to Trailing Data
        self.sample_data = self.sample_data[-self.data_size:]
        self.accelerometer_data = self.accelerometer_data[-self.data_size:]
        self.gyroscope_data = self.gyroscope_data[-self.data_size:]
        self.complementary_data = self.complementary_data[-self.data_size:]

    def save_fig(self, fig_name) :
        """Save Figure to File."""
        file_path = os.path.join(self.figures_dir, f"{fig_name}.jpg")

        self.fig.savefig(file_path, format = 'jpg', dpi = 800)
        print(f"Figure Saved to {file_path}")

    def save_logs(self, file_name):
        """Save Data to CSV File."""
        file_path = os.path.join(self.logbook_dir, f"{file_name}.csv")

        csv_df = self.data_df.copy()
        csv_df['Time (PST)'] = self.data_df['Time (PST)'].dt.strftime('%Y-%m-%d %H:%M:%S')
        csv_df.to_csv(file_path, index = False)

        print(f"Data Exported to {file_path}\n\nData Preview:\n")
        display(self.data_df.head(2))
        display("---------------------")
        display(self.data_df.tail(2))
```

Listing 13: Python Stripchart