Balance Robot Report

ELEC50015 - Electronics Design Projects 2

Team Bonus

Word Count: 9653

17 June 2024

Submitted to: Dr. Edward Stott

Maximilian Adam
CID: 02286647

Xiaoyang Xu
CID: 02030033

Fedor Dakhnenko
CID: 02228543

Yueming Wang
CID: 02061452

Maxim Kelly
CID: 02049056

Yingkai Xu
CID: 01848959

IMPERIAL

Abstract

This project focuses on designing and developing a two-wheeled self-balancing robot with autonomous navigation capabilities. The main objectives were to create a stable balancing mechanism, enable autonomous movement based on sensor inputs, and provide a user-friendly interface for manual control and monitoring. Our methodology involved integrating multiple subsystems, including balance control, movement control, image processing, and communication.

The balance control system was implemented using a cascaded loop PID structure, with an inner loop for tilt control and an outer loop for speed regulation. Image processing was achieved using a Raspberry Pi and camera module, enabling the robot to detect and move towards specified objects. The communication subsystem allows remote control and real-time monitoring via a WiFi connection.

Testing demonstrated the robot's ability to maintain balance and execute movement commands accurately. The robot successfully navigates autonomously, identifying and approaching target objects, while the user interface provides feedback and control options.

Overall, the project achieved its goals, and alongside we have improved our ability in software and hardware design. Future options of improvement are refining the control algorithms and improving the robustness of the image processing and communication systems.

Contents

1 Introduction

1.1 Project Objective

The objective of this project is to design and develop a two-wheeled self-balancing robot, able to move autonomously. We aim to present the robot being able to identify and navigate to certain objects specified beforehand. Our final product should include a robust control system and the incorporation of various subsystems for autonomous navigation and interaction with surrounding environment.

Our main objectives for the project are as follows:

- Design a balancing mechanism robust in nature, formed by a combination of sensors and control algorithms.
- Ensure that the robot can navigate in an autonomous manner through processing inputs from cameras and other sensors.
- Implement a user interface that allows for manual controls to be implemented, as well as live monitoring of the robot state.
- Develop additional features like object identification and audio input handling.
- Design a modular system architecture that allows for easy enhancements, modifications and integration.

Our goals can only be achieved by implementing knowledge from control systems, sensor integration, mechanical design, and software development, requiring a diverse multidisciplinary team. Providing an excellent opportunity to learn and use as experience when developing projects in the future.

1.2 Project Requirements

Taking from the initial requirements [1], we integrated our specific goals and outlined them as follows:

1. The robot shall demonstrate a form of autonomous behaviour based on inputs from a camera and/or other sensors.
    - **Goal**: Identify specific objects with a camera attached to the chassis and move towards them.
    - **Stretch goal**: Receive audio input and perform actions based on these inputs.
2. The robot shall balance on two wheels, with a centre of gravity above the axis of rotation of the wheels.
    - **Goal**: Implement a cascaded loop PID control structure to allow balance and easy movement
3. There shall be a remote control interface that allows a user to enable or control the autonomous behaviour and to move the robot manually in two dimensions around a flat surface.
    - **Goal**: Control the robot manually via a user interface (UI). And switch between autonomous and manual control.
4. The user interface shall display useful information about the power status of the robot, such as power consumption and remaining battery energy.
    - **Goal**: Display key information on the UI, including battery life, speed, PID values, and the robot's current view.
5. The robot should augment the provided chassis with a head unit that suits the chosen demonstrator application.
    - **Goal**: Attach the camera unit and all other sensing equipment effectively.

6. The user interface shall provide inputs and display information pertinent to the demonstrator application.

    - **Goal**: Provide the user with a tool to monitor and influence the autonomous behaviour of the robot.

1.3 Design Approach

Due to initial complications when forming our group, we adopted an incremental design approach to make the most of the limited time available. The design approach focused on simplicity and functionality to facilitate easier integration of all sub-modules.

Initial Planning and Team Formation
At the outset, we faced challenges in forming a cohesive team. To address this, we allocated tasks based on individual interests and strengths. This ensured that each team member was engaged and could contribute effectively to their assigned tasks.

Incremental Design Strategy
An incremental design strategy was chosen to manage the complexity of the project and to facilitate easier debugging and testing. We began with the core balancing mechanism, as it is the most critical aspect of the robot. Once a stable balance was achieved, we incrementally added other functionalities such as object identification and manual control.

Research and Preliminary Analysis
We researched extensively to understand the project requirements and to analyze existing solutions. This research included studying balance control algorithms, sensor integration, and user interface design. Based on this research, we performed a preliminary analysis to estimate the hours required for each goal, which informed our planning and task allocation.

Design Principles and Objectives
Our design was guided by principles of simplicity and functionality. These principles were chosen to ensure that the system would be easy to integrate, debug, and maintain. The main objectives were to create a robust and scalable system that could be easily tested and upgraded.

Integration and Testing Approach
Integration was carried out incrementally, starting with the core balancing components. Each functionality was added and tested individually to ensure stability and performance before moving on to the next. This approach allowed us to identify and resolve issues early in the process, leading to a more reliable overall system.

Once fully integrated, a final testing run was conducted to optimize the design. This final test ensured that all components worked seamlessly together and met the project requirements.

1.4 Report Structure

The following sections of this report provide an in-depth analysis of the project:

- **System Design**: Specifies the top-level design and integration process.
- **Subsystem Implementation**: Delves into the creation of each component.
- **Integration and Testing**: Shows how components were assembled and tested.
- **Project Management**: Details how time and other resources were managed throughout the project.

Finally, the **Conclusion** section summarizes the project's achievements and suggests potential directions for future research.

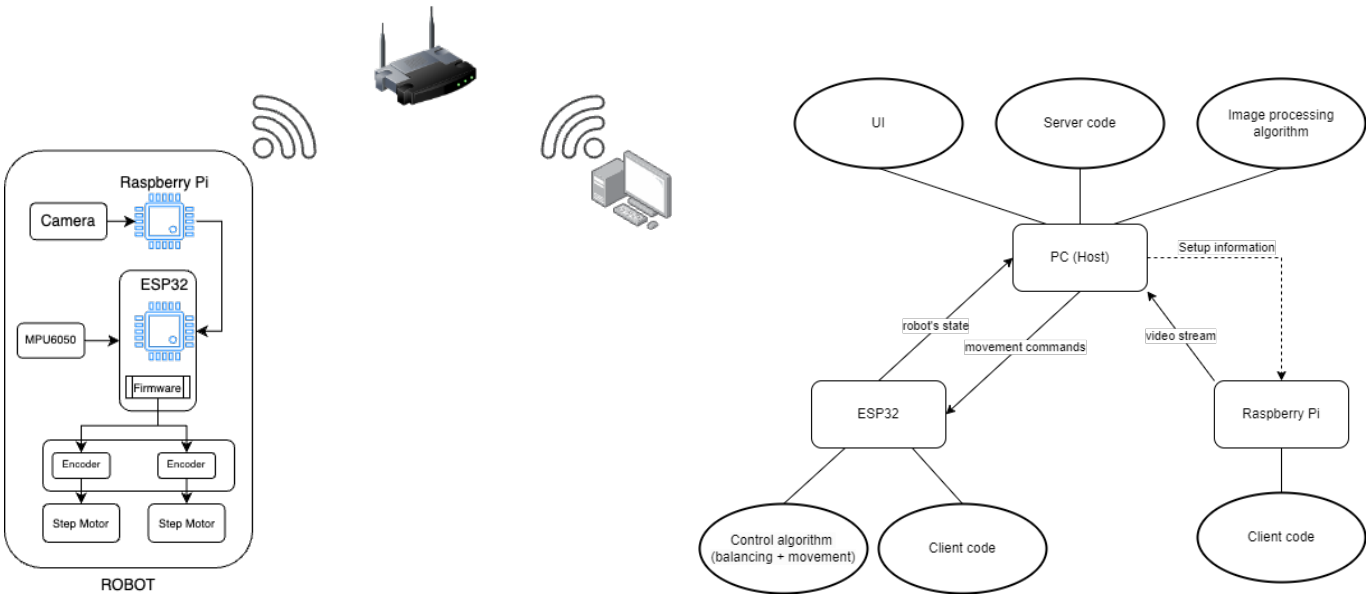
2 System Design

Our balance robot's system design is a comprehensive integration of several key subsystems, each responsible for critical functionalities. The primary subsystems we defined are as follows:

- **Balancing**
- **Movement**
- **Image Processing**
- **Communication**
- **User Interface**
- **Power**
- **Audio**

And finally **Integration** to put all the subsystems together and compose our final design.

2.1 System Diagrams



(a) Hardware diagram

(b) Software diagram

Figure 1: System diagrams

The hardware system diagram (figure 1a) illustrates the connections between the microcontroller (ESP32), sensors (MPU6050), actuators (stepper motors), image processing unit (Raspberry Pi and Camera), and other peripheral devices we used.

The software system diagram (figure 1b) outlines our architecture, highlighting key modules such as the control algorithm, sensor data processing, data routing, and user interface management.

2.2 Key Modules and Principles

ESP32 Microcontroller
Role: Acts as the central processing unit, managing sensor data, executing control algorithms, and handling communication with peripheral devices.
Justification: The ESP32 was chosen within this project for its processing power, built-in WiFi, and versatility in handling various I/O operations.

MPU6050 Sensor
Role: Provides real-time motion sensing by measuring acceleration and gyroscopic data.
Justification: This sensor is crucial for maintaining balance of our robot by providing feedback on the robot's orientation and movement.

Raspberry Pi and Camera
Role: Used for image processing tasks such as object identification and tracking.
Justification: The Raspberry Pi offered sufficient processing power and compatibility with the OpenCV library, making it ideal for complex image processing tasks that we require of it.

Stepper Motors
Role: Enable precise control of the robot's movement and positioning.
Justification: Stepper motors were selected for their accuracy and ease of control, essential for maintaining balance and executing movement commands. We also could use them to accurately measure speed.

PID Control Algorithm
Role: Adjusts the robot's position dynamically to ensure balance.
Justification: PID controllers provide a robust method for maintaining balance by continuously adjusting the motor outputs based on feedback from the MPU6050 sensor.

WiFi Communication
Role: Allows for remote control and monitoring of the robot.
Justification: WiFi communication enhances our robot's functionality and user interaction by enabling remote command and data transmission. Required by our projects specification

User Interface (UI)
Role: Provides a platform for manual control and displaying key information such as battery life, speed, and PID values.
Justification: Web-based UI offers accessibility and ease of use, allowing us to interact with the robot through a familiar and informative interface.

Power Management
Role: Ensures we know stable and sufficient power is being supplied to all components.
Justification: Effective power measurement is extremely useful in ensuring the reliable operation of our robot, ensuring we know the robot is operating correctly.

3 Subsystem Implementation

3.1 Balancing

The balancing control system of the robot comprises of two loops: an inner loop dedicated to managing the tilt and an outer loop focused on regulating the speed. These work in concert to maintain the robot's stability and desired movement.

3.1.1 Inner Loop: Tilt Control

The inner loop of the balancing robot's control system is crucial for managing the tilt. This loop uses a PID controller to dynamically adjust the motor outputs based on the robot's tilt angle, with the main aim of keeping the robot upright by continuously correcting any deviations from the setpoint.

Measuring Tilt
Tilt measurement is done using the MPU6050 sensor, which has both an accelerometer and a gyroscope. The accelerometer measures the direction of gravity to give an estimate of the tilt angle, but this can be inaccurate due to longitudinal acceleration. To fix this, the gyroscope measures the rate of change of the tilt angle, offering short-term accuracy.

Complementary Filter
A complementary filter combines the accelerometer and gyroscope data to give a more accurate tilt angle measurement. The filter uses a low-pass filter on the accelerometer data and a high-pass filter on the gyroscope data. With the formula for the complementary filter being:

$$\Theta_n = (1 - C)\Theta_a + C\left(\frac{d\Theta_g}{dt}\Delta t + \Theta_{n-1}\right) \tag{1}$$

where:

- Θ_n is the calculated tilt at iteration n
- Θ_{n-1} is the tilt from the previous iteration
- Θ_a is the tilt angle from the accelerometer
- $\frac{d\Theta_g}{dt}$ is the rate of change of the tilt angle from the gyroscope
- Δt is the time interval between iterations
- C is a factor between 0 and 1, typically close to 1.

We implement the complementary filter as follows:

```
pitch = atan2(a.acceleration.z, sqrt(a.acceleration.x * a.acceleration.x + a.acceleration.y
    * a.acceleration.y)) - 0.08837433;
float gyroPitchRate = g.gyro.y; // Pitch rate
double dt = LOOP_INTERVAL / 1000.0;  // Convert LOOP_INTERVAL to seconds
filteredAngle = (1 - alpha) * pitch + alpha * (previousFilteredAngle + gyroPitchRate * dt);
previousFilteredAngle = filteredAngle;
```

Where 0.08837433 was obtained from testing the error from 0 when the robot was vertical:

Calibration	Value
Calibration 1	0.093061
Calibration 2	0.101147
Calibration 3	0.070915
Average	0.08837433

Table 1: Calibration Values

3.1.2 Outer Loop: Speed Control

The outer loop is responsible for regulating the robot's speed. This loop uses another PID controller to manage the setpoint of the inner loop based on the desired speed. By adjusting the tilt angle setpoint, the outer loop ensures the robot moves at the desired speed while staying balanced.

Speed Measurement
Speed is measured using the stepper motors, which provide feedback on the rotational speed. The robot calculates the speed in cm/s by converting the stepper motor's steps to the distance traveled using the wheel circumference. The details are summarized in Table 2.

Parameter	Value
Diameter of wheels	6.6 cm
Wheel circumference	20.736 cm
Steps per revolution	200 steps, with 16 microsteps
Distance per step	0.00648 cm
Distance between the centers of the two wheels	11.9 cm

Table 2: Speed Measurement Parameters

Filters for Speed Measurement
To get a smooth speed measurement, two filters are used:

1. **Exponential Moving Average (EMA):** Smooths out the raw speed readings to reduce noise. The EMA is calculated as:

$$\text{EMA} = \alpha_{\text{EMA}} \times \text{rawSpeed} + (1 - \alpha_{\text{EMA}}) \times \text{previousEMA} \tag{2}$$

   This helps reduce the impact of sudden changes in speed, providing a more stable value.

2. **Butterworth Low-Pass Filter:** Further filters the EMA output to eliminate high-frequency noise. The Butterworth filter is effective at removing noise without distorting the signal. The filter coefficients are calculated based on the desired cutoff frequency and sampling frequency.

The Butterworth filter is defined as:

```
float butterworthFilter(float *x, float *y, float input) {
  for (int i = order; i > 0; i--) {
    x[i] = x[i - 1];
    y[i] = y[i - 1];
  }
  x[0] = input;

  y[0] = 0;
  for (int i = 0; i <= order; i++) {
    y[0] += b[i] * x[i];
    if (i > 0) {
      y[0] -= a[i] * y[i];
    }
  }
  return y[0];
}
```

With the method of calculating coefficients specified in appendix A.1.

We implement the filtering as follows:

```
float rawSpeed1 = step1.getSpeed() / 2000.0;
float rawSpeed2 = step2.getSpeed() / 2000.0;

emaSpeed1 = alphaEMA * rawSpeed1 + (1 - alphaEMA) * emaSpeed1;
emaSpeed2 = alphaEMA * rawSpeed2 + (1 - alphaEMA) * emaSpeed2;

float butterSpeed1 = butterworthFilter(x1Butter, y1Butter, emaSpeed1);
float butterSpeed2 = butterworthFilter(x2Butter, y2Butter, emaSpeed2);
```

It can be seen that we initially divide the speed obtained from the getSpeed() step function, this is due to the fact that speed units are provided in steps per 2000 seconds. Requiring us to convert it for our measuring method.

Finally speed is measured like so:

```
float averageSpeedSteps = (butterSpeed1 - butterSpeed2) / 2.0;
float distancePerStep = wheelCircumference / stepsPerRevolution;
speedCmPerSecond = averageSpeedSteps * distancePerStep;
```

3.1.3 Cascaded Loop Architecture

The cascaded loop architecture combines the inner and outer loops to achieve both balancing and movement. The outer loop PID controller sets the desired tilt angle (setpoint) for the inner loop based on the speed error. The inner loop then adjusts the motor outputs to maintain the set tilt angle, thereby achieving the desired speed while keeping the robot balanced - see figure 2 fore reference.

```
speedControlOutput = speedPid.compute(speedCmPerSecond);
TargetTiltAngle = speedControlOutput * 0.001;

balancePid.setSetpoint(TargetTiltAngle);
balanceControlOutput = balancePid.compute(filteredAngle);
```

The output from the speed control PID is scaled when setting the tilt angle setpoint for the inner loop. This scaling factor is used because tilt is approximately proportional to acceleration, and the proportional gain (K_p) roughly scales the difference between speed and tilt angle. By scaling the output, the system ensures that the adjustments to the tilt angle are within a reasonable range, preventing excessive tilt and maintaining stability.

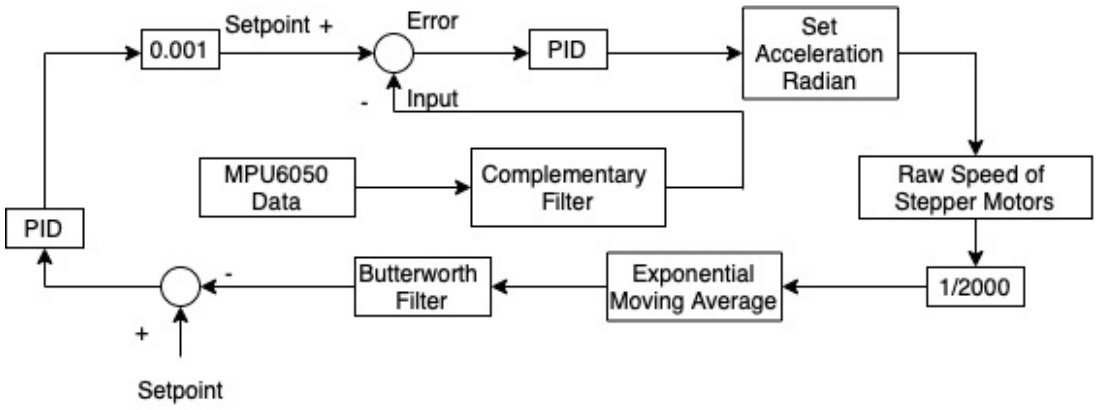


Figure 2: Cascaded loop

3.1.4 Relationships Between Tilt and Movement

Initially we desired to find a linear relationship between tilt and a variable related dimensionally to the output. With $K_p = 1000$, $K_i = 15$, and $K_d = 91$. This resulted in finding a linear relationship between tilt and jerk which through differentiating the outerloop output twice and finding the proportional parameter would result in a relationship between desired speed and corresponding tilt angle. However due to multiple differentiation the system became too unresponsive. Therefore we chose to approximate by the dominant variable K_p setting K_i and K_d to 0, finding a linear relationship between tilt and acceleration that could be used as described in section 3.1.3.

Tilt	Jerk
0.1	0.15
0.02	0.3
0.04	0.6

Tilt	Acceleration
0.01	10
0.02	20
0.04	40

Table 3: Relationship Between Tilt and Jerk (left) and Relationship Between Tilt and Acceleration (right)

The relationships for both tables can be summarized by the following equations:

$$\text{Tilt} = \frac{1}{15} \times \text{Jerk} \tag{3}$$

$$\text{Tilt} = 0.001 \times \text{Acceleration} \tag{4}$$

By integrating all of these components, the balance robot can effectively maintain its upright position and navigate based on the specified speed commands.

3.1.5 Tuning the PID Controller

Balancing a robot is similar to an inverted pendulum, which requires precise control to maintain stability. Tuning a PID (Proportional-Integral-Derivative) controller for such a system is challenging due to the dynamic nature of the system and the multible variables required to be tuned. To achieve stability we utilised the following method to tune the PID controller.

Firstly we specified what each parameter corresponds to and the effects of altering it within the context of a balance robot.

Proportional (P)
The proportional term determines the reaction to the current error. A higher P gain can cause the system to respond faster but can also lead to increased oscillations.
Integral (I)
The integral term accounts for the accumulation of past errors. A higher I gain can reduce steady-state error but can introduce lag and lead to instability.
Derivative (D)
The derivative term predicts future error based on its rate of change. A higher D gain can dampen oscillations but too much can cause noise amplification.

We initially started with P-Only Control to simplify the process and achieve a base level of stability. Our process is as follows:

1. Set I and D gains to zero.
2. Increase the P gain until the system oscillates steadily. This is known as the ultimate gain (K_u).
3. Note the period of oscillation (T_u).

Through experimentation K_u was determined to be 495.

With T_u being calculated as follows:

Run	Time of Oscillations (s)	Number of Oscillations
1	20.26	20
2	22.55	20
3	19.72	20
Average	20.843	-

Table 4: Calculation of T_u

$$T_u = \frac{\text{Average Time of Oscillations}}{\text{Number of Oscillations}} = \frac{20.843}{20} = 0.956$$

Now that we have determined K_u and T_u, the Ziegler-Nichols method can be applied to set the initial PID values:

Controller Type	K_p	K_i	K_d
P Controller	$0.5 \times K_u$	-	-
PI Controller	$0.45 \times K_u$	$\frac{K_p}{1.2 \times T_u}$	-
PID Controller	$0.6 \times K_u$	$\frac{2 \times K_p}{T_u}$	$\frac{K_p \times T_u}{8}$

Table 5: Ziegler-Nichols Tuning Parameters

Now it is required we fine-tune the parameters, keeping track of the current issues associated with the stability of the system. Through how we specified the effects of the parameters above, we can tune the system as follows:

To Reduce Oscillations:

- If oscillations are too high, decrease the P gain slightly.
- Increase the D gain to dampen oscillations.

To Eliminate Steady-State Error:

- Gradually increase the I gain to eliminate steady-state error. Being cautious as too much I can cause instability.

Using this information, we now tested and adjusted the PID values accordingly:

- Make small adjustments to the PID gains and observe the system's response.
- Ensuring each change is tested for a sufficient period to understand its impact fully.
- Balance between gains to achieve desired performance:
  - If the system responds too sluggishly, increase P gain.
  - If there is residual steady-state error, increase I gain.
  - If the system is too noisy or oscillates, increase D gain or slightly reduce P gain.

Throughout this, we needed to keep in mind the fundamental mathematical implications of each parameter corresponding to our PID control calculations:

$$\begin{aligned} \text{error} &= \text{setpoint} - \text{measured\_value} \\ \text{integral} &= \text{integral} + \text{error} \times dt \\ \text{derivative} &= \frac{\text{error} - \text{previous\_error}}{dt} \\ \text{output} &= K_p \times \text{error} + K_i \times \text{integral} + K_d \times \text{derivative} \end{aligned}$$

Finally here are our experimentally obtained results for our inner loop:

- Ultimate Gain (K_u) = 495
- Average Period of Oscillation (T_u) = 0.956

Round	K_p	K_i	K_d	Comments
1	297	621.3	35.492	Slow to respond
2	326.7	559.17	31.942	Oscillates with increasing amplitude
3	294.03	503.253	35.1362	Oscillates with increasing amplitude
4	264.627	452.9277	38.64982	Oscillates with increasing amplitude
5	211.702	362.34216	46.379784	Oscillates with increasing amplitude
6	169.3616	181.17108	69.569676	Oscillates with increasing amplitude
7	152.42544	90.58554	76.526644	Oscillates with increasing amplitude. More oscillations needed to topple
8	137.182896	72.468432	91.831973	Oscillates a little around centre then quickly speeds up and topples
9	123.4646064	36.234216	110.1983676	Stays oscillating around centre for a little while then quickly speeds up and topples
10	111.11814576	18.117108	132.23804112	Oscillates for a while about the centre point then quickly loses stability, speeds up, and topples
11	99.960331184	16.3053972	105.7904329	Lots of little vibrations while balancing, system eventually speeds up and topples
12	89.9642980656	14.67485748	95.21138961	A few little vibrations, has small oscillations about centre point, if disturbed greatly overcorrects
13	80.96786825904	13.207371732	76.169111688	Few vibrations, small disturbance leads to oscillations of increasing amplitude then toppling
14	80	15	95	Stable although unable to properly respond to disturbances

Table 6: PID Tuning Rounds and Results for the Inner Loop

After achieving desirable stability, the system was unable to respond properly to disturbances. Therefore, we chose to scale K_p through further trial and error to achieve effective responsiveness.

Finally, the following PID values were found to provide the most stable performance.

- $K_p = 1000$
- $K_i = 15$
- $K_d = 95$

After the completion of the Inner Loop testing, and once speed had begun to be measured correctly and smoothly, we needed to test the outer loop to find its optimal PID values. However this would have to be carried out differently:

1. Speed setpoint needs to be offset from 0 otherwise all balance would be attributed solely to the inner loop
2. Ziegler-Nichols method is no longer appropriate with a graphical approach and analysis allowing faster and more accurate testing
3. Initially aim for a system with minimal oscillations using only K_p then attune K_i and K_d accordingly.

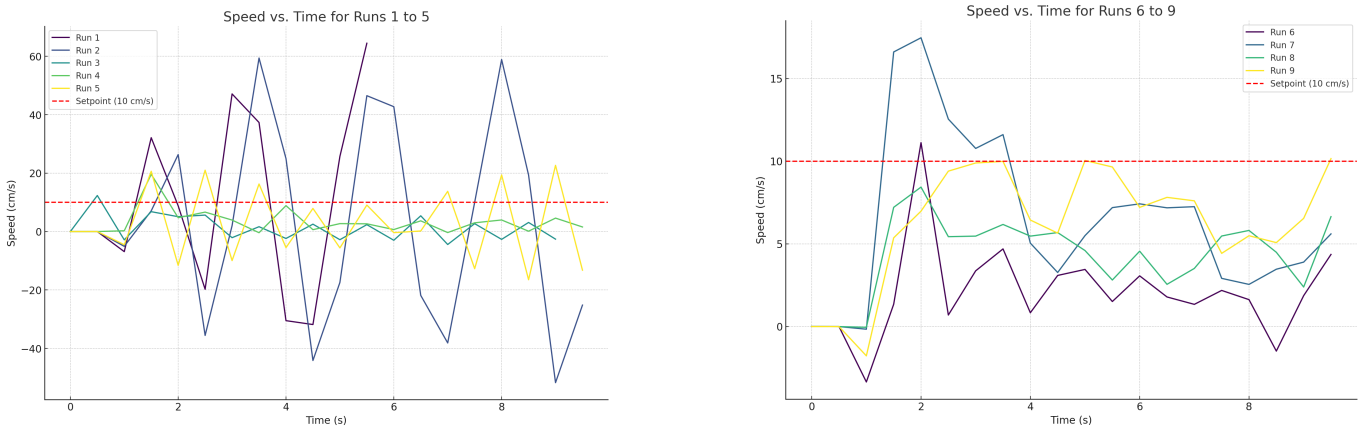
For the setpoint we chose 10 cm/s from trial and error, finding it to be a good compromise between being offset from 0 but avoiding extreme operating conditions where the inner loop would be unable to maintain stability.

Testing Process

1. **Initial Setup**: set PID parameters, and measure the system's speed response. Each speed data point was recorded at 0.5-second intervals.
2. **Data Collection**: For each test run, collect speed data and plot it against time with a reference line at the setpoint (10 cm/s) to visualize the target speed.

3. **Analysis and Adjustment**: After analyzing the speed response, adjust the PID parameters to improve stability and reduce oscillations. And repeat across multiple runs.

Using this process the following graphs (figure 3) and observations (table 7) were obtained:



(a) Runs 1 to 5

(b) Runs 6 to 9

Figure 3: Speed vs. Time for Different Runs

Run	PID Values	Learnings
1	$K_p = 250$, $K_i = 0.0$, $K_d = 0.0$	Large oscillations and instability indicate need to reduce K_p.
2	$K_p = 100$, $K_i = 0.0$, $K_d = 0.0$	Reduced oscillations but still unstable, suggesting further K_p reduction needed.
3	$K_p = 1$, $K_i = 0.0$, $K_d = 0.0$	Improved stability but still some oscillations.
4	$K_p = 3$, $K_i = 0.0$, $K_d = 0.0$	Stability improved, oscillations persisted but reduced.
5	$K_p = 5$, $K_i = 0.0$, $K_d = 0.0$	Great increase in oscillations.
6	$K_p = 3$, $K_i = 0.1$, $K_d = 0.1$	Improved stability and reduced fluctuations, but significant steady state error.
7	$K_p = 3$, $K_i = 0.15$, $K_d = 0.15$	Improved accuracy but slight increase in overshoot, suggesting a need to balance K_p and K_d.
8	$K_p = 3$, $K_i = 0.2$, $K_d = 0.15$	Achieved better balance, minimal fluctuations, and closer adherence to setpoint.
9	$K_p = 3$, $K_i = 0.35$, $K_d = 0.2$	Significant improvement, stable response, minimal deviations, and close to setpoint.

Table 7: Summary of PID Tuning Runs

Further runs and adjustment just led to greater oscillations, so the PID parameters for the outer loop were chosen to be:

$$K_p = 3 \quad K_i = 0.35 \quad K_d = 0.2$$

Additionally, we wrote an auto-tune Python script to brute force the tuning of PID parameters seen in appendix A.2. However, we found this method to be too slow and inefficient, leading us to abandon it in favor of our manual tuning as described above.

3.1.6 Testing Results and Discussion

Testing Methodology

The balancing robot was tested under various controlled conditions to ensure stability and responsiveness. Consistent testing conditions were maintained using a self-induced impulse method to minimize human-induced variability. The primary metrics for evaluation were stability, responsiveness, and oscillations.

Metrics for Evaluation

- **Stability**: Measured by the duration the robot can maintain balance without toppling.
- **Responsiveness**: Evaluated by how quickly the robot corrects its position after a disturbance.
- **Oscillations**: Frequency and amplitude of oscillations around the balanced state.

Test Results

- **Stability**: The robot was able to remain upright for 5 minutes during continuous testing, leading us to believe it is stable.
- **Responsiveness**: The robot corrects its tilt within 1-2 seconds after a disturbance.
- **Oscillations**: Amplitude of oscillations was within $[-0.121305, 0.318978]$ radians with an average tilt of 0.052 radians, when stationary over a period of 60 seconds. Speed was within $[7.007236, 12.646814]$ cm/s for a randomly selected 10 second run out of 5, with an average of 8.654 cm/s.

Discussion
Our tuning process highlights the trade-offs between responsiveness and stability. Higher K_p values increased responsiveness but also the risk of oscillations. Incremental adjustments and thorough testing were crucial in achieving optimal performance. And the implementation of the filters (EMA and Butterworth) for speed measurement made it possible for us to have a stable outer loop.

Future improvements could include implementing advanced techniques like gain scheduling or model-based tuning for varying operating conditions. Although the current setup clearly demonstrates the effectiveness of our cascaded loop architecture in maintaining the robot's balance.

3.2 Movement

When implementing movement initially we identified that if speed sensing inaccuracies could not be resolved we would have to rely only on the inner loop of our control system. We therefore decided to initially allocate someone to produce movement functions which only rely on a stable inner loop.

3.2.1 Inner Loop Only Movement

The robot has to balance continuously while accepting commands to move. We chose to implement this behaviour using separate threads for balancing and input detection. This approach facilitates non-blocking input since only the movement thread is stalled and creates a clear code structure, which is easy to debug and upgrade, without including any extra libraries.

The threads are initialized in setup(); the content of both threads is included in an infinite loop. This way, the functionality and the sequence of the code are the same as in the default setup() and loop() framework.

In order for threads to communicate with each other and to keep track of how the robot is moving at every moment several flags are instantiated globally. The movement thread accepts an input (from the UI or processed image) and changes these flags accordingly as well as making an impulse, so that the robot starts moving.

Moving Forwards and Backwards

```
void moveForwards() {
  stop();
  movingForwards = true;
  balancePid.setSetpoint(setpoint - offset);

  input = true; // Indicate that we are handling input and block balancing
  step1.setAccelerationRad(15);
  step2.setAccelerationRad(-15);
  step1.setTargetSpeedRad(20);
  step2.setTargetSpeedRad(-20);
  delay(50);  // Duration of the impulse
```

```
  resetSteppers(); // Stop any ongoing movements before applying new command
  input = false;
}
```

Consider the code above. Initially the robot stops (this functionality is explained later). Then the setpoint is put at a small offset, which was determined experimentally[1] to be 0.08 (see table 8). This forces the robot to stay slightly off balance, so that it moves forward, but doesn't fall as the offset isn't big enough to cause significant disturbance in balancing. In addition, the corresponding flag is set to *true*.

Offset value	result
≤ 0.06	the robot didn't move at all or moved very slowly
0.07	the robot moved with occasional stopping
0.08	the robot moved with rare stopping and appropriate speed
0.09	the robot moved too fast and toppled within 5 seconds
≥ 0.1	the robot toppled within 2 seconds

Table 8: Offset tuning

After that the robot inflicts an impulse on itself, which insures that the robot starts moving forwards. During this impulse balancing is disabled using the input flag.

Moving backwards is achieved with an almost identical piece of code, which has opposite flag, numbers for motors, and the offset is added to the sepoint, not subtracted.

Turning Left and Right

Turning is implemented similarly with the exception of not interacting with the setpoint. In this case the impulse is all it takes for the robot to turn, as it produces rotational speed (yaw), which decreases very slowly due to small friction.

However, this is still a hazard, which should be taken into account when operating the robot. If the turning isn't interrupted soon enough, the counter impulse will cause an overshoot in acceleration, thus some rotational speed will be left unchecked.

```
void turnLeft() {
  stop();
  turningLeft = true;

  input = true; // Indicate that we are handling input
  step1.setAccelerationRad(-40);
  step2.setAccelerationRad(-40);
  step1.setTargetSpeedRad(-20);
  step2.setTargetSpeedRad(-20);
  delay(50);  // Duration of the impulse
  resetSteppers(); // Stop any ongoing movements before applying new command
  input = false;
}
```

Stopping

This functions checks if the robot is moving and stops that movement by either resetting the setpoint or applying a counter-impulse to negate the rotational speed.

```
void stop() {
  if (movingForwards){
    //reset setpoint
    balancePid.setSetpoint(setpoint);
  } else if (movingBackwards){
    //reset setpoint
    balancePid.setSetpoint(setpoint);
  } else  if (turningLeft){
    //counter impulse right
    input = true;
    step1.setAccelerationRad(40);
```

[1] the experiments were carried out with an unfinished version of the balancing algorithm, therefore the initial conditions and disturbances affected the outcomes of the tests

```
    step2.setAccelerationRad(40);
    step1.setTargetSpeedRad(20);
    step2.setTargetSpeedRad(20);
    delay(50);
    resetSteppers();
    input = false;
  } else  if (turningRight){
    //counter impulse left
    input = true;
    step1.setAccelerationRad(-40);
    step2.setAccelerationRad(-40);
    step1.setTargetSpeedRad(-20);
    step2.setTargetSpeedRad(-20);
    delay(50);
    resetSteppers();
    input = false;
  }
  movingForwards = false;
  movingBackwards = false;
  turningLeft = false;
  turningRight = false;
}
```

3.2.2 Movement enabled by Cascaded Loop Control

The final movement control subsystem leverages the existing balance control structure, incorporating the additional PID control loop to manage speed and direction. The main functions implemented include moving forward, backward, and turning left and right. These functions are much simplified from before, relying only on altering the speed setpoint for linear movement and direction of motor rotation for turning.

Movement Functions

The primary movement functions are defined as follows:

```
void stop() {
  speedPid.setSetpoint(0);
}

void moveForward(double speed) {
  speedPid.setSetpoint(speed);
}

void moveBackward(double speed) {
  speedPid.setSetpoint(-speed);
}

void turnLeft(double speed) {
  step1.setTargetSpeedRad(-speed);
  step2.setTargetSpeedRad(-speed);
}

void turnRight(double speed) {
  step1.setTargetSpeedRad(speed);
  step2.setTargetSpeedRad(speed);
}
```

Speed and Direction Control

Speed and direction are controlled through the stepper motors, which receive commands based on the outputs of the PID controllers. The PID controllers adjust the motors acceleration to achieve the desired movement while maintaining balance.

```
// Outer loop: Speed control
speedControlOutput = speedPid.compute(speedCmPerSecond);
TargetTiltAngle = speedControlOutput * 0.001;

// Inner loop: Balance control with speed control output as setpoint
balancePid.setSetpoint(TargetTiltAngle);
```

```
balanceControlOutput = balancePid.compute(filteredAngle);

// Apply dead-band
if (abs(balanceControlOutput) < deadBand) {
  balanceControlOutput = 0;
}

step1.setAccelerationRad(-balanceControlOutput);
step2.setAccelerationRad(balanceControlOutput);

if (balanceControlOutput > 0) {
  step1.setTargetSpeedRad(-20);
  step2.setTargetSpeedRad(20);
}
if(balanceControlOutput < 0) {
  step1.setTargetSpeedRad(20);
  step2.setTargetSpeedRad(-20);
}
```

As can be seen in the code we also utilised a dead-band, so that when the output is negligible we stop the movement of the motors. This allows for smoother movement and balancing, removing unnecessary over-corrections.

3.2.3 Testing Results and Discussion

Testing Methodology
The movement control subsystem was tested under various conditions to ensure accurate and responsive performance. The primary metrics for evaluation were precision in movement, responsiveness to commands, and stability during motion.

Metrics for Evaluation

- **Precision**: Measured by the accuracy of the robot's movements relative to commanded positions.
- **Responsiveness**: Evaluated by the time taken for the robot to start and stop movements upon command.
- **Stability**: Assessed by the robot's ability to maintain balance during and after executing movement commands.

Test Results

- The robot demonstrated a relatively high precision in executing movement commands, maintaining a straight path and accurate, on the spot turns.
- Responsiveness was good, with the robot starting and stopping movements within 0.4 seconds of command input.
- Stability was well-maintained during movement, with manageable oscillations and no instances of toppling.

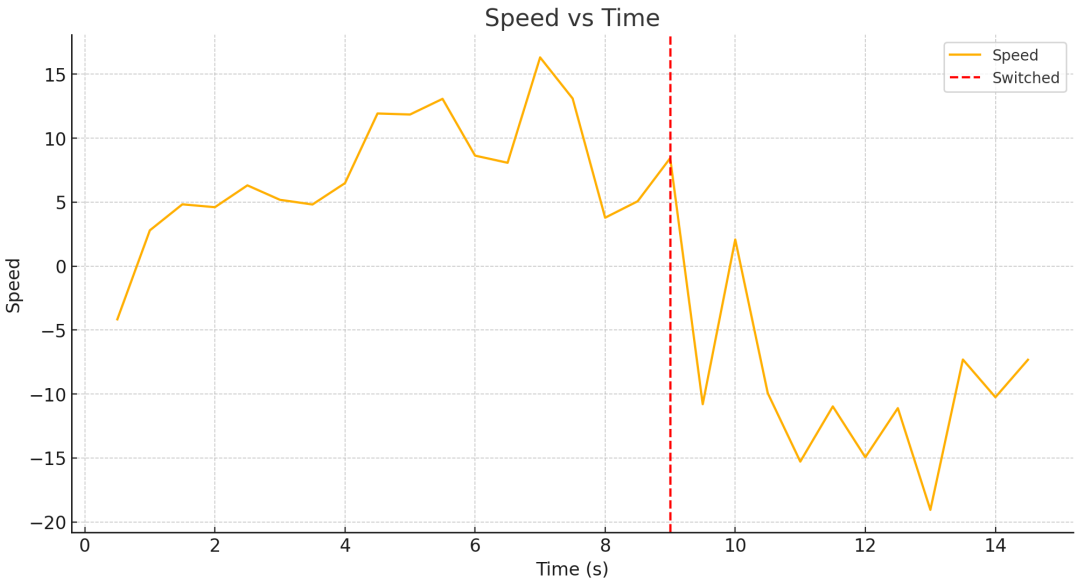


Figure 4: Speed vs Time

Discussion
The testing revealed that the movement control subsystem effectively integrates with the balancing system to provide smooth and precise control. Our PID controllers for speed and direction worked well together tandem ensuringaccurate movement while maintaining balance. Fine-tuning our PID parameters was helpful in achieving our current performance, with further slight adjustments made based on observed oscillations and response times.

Future improvements could include implementing the adaptive PID tuning as stated in the control section or incorporating additional sensors for enhanced navigation capabilities. The current setup demonstrates the effectiveness of our chosen control algorithm and the robustness of our subsystem design in the real-world.

3.3 Image Processing

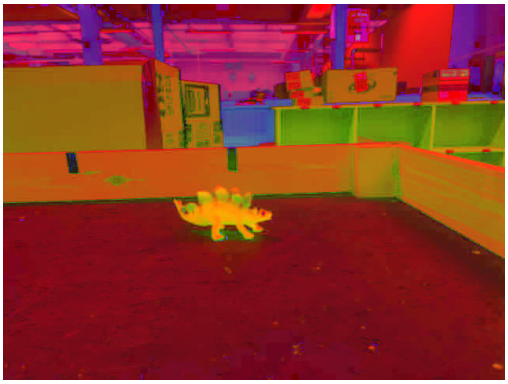
The robot's function is to identify and move to certain objects specified by a human observer. In order to do so the robot is equipped with a Raspberry Pi and its camera module. The Raspberry Pi sends a video stream to a server, where the image processing takes place.

Before the robot can start performing its task, the pilot has to specify what object the robot should follow. This is done in the UI, which displays the video stream, a mask (explained later) and several sliders, which allow the user to choose a range of colours, which characterise the required object.

When the setup is completed, the main algorithm activates, which heavily relies on OpenCV library (see appendix B.1). When an image is received it is converted to from a RGB to a HSV format - this lessens the impact of shades on colour identification, since they don't affect the hue channel: figure 5b. After that if a pixel in the image is within the range specified during the setup, then this pixel becomes white, otherwise it become black. The resulting image is the mask, which was used for calibration in setup: figure 5c. Then, the mask is converted into a binary image (it had 3 channels before, but since it is black and white it can be encoded with only 1 channel). This binary image is used to perform a morphological opening, which eliminates artifacts in the image and makes the later parts of the code faster, since they have less pixels to work with: figure 5d. After the morphology is applied, the algorithm creates an array of connected components (white and black areas) and information about them. The array is sorted by area in descending order. It is common sense that the biggest component is black background, the second biggest component is white area, which corresponds to the object the algorithm has to identify, and other components are noise.

Thus, the object was identified, so the robot has to start moving towards it. Using the component's centroid (provided in the array of the components and their characteristics) the algorithm identifies if the

(a) initial picture



(b) HSV representation

(c) Mask

(d) Morphology open

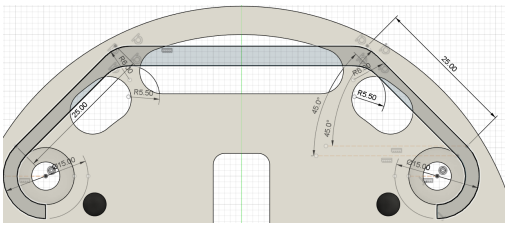
Figure 5: Image processing example

object is in the left, middle or right third of the image and sends a command to the esp32 to go straight, start or stop turning. If the robot moves close enough to the object, the latter will appear at the bottom of the captured image, touching the side. When that happens, the robot stops, so that it doesn't lose the tracked object. Additionally, the coordinates of the edges of the component are used to create a box around the object in the initial image to indicate to the observer what the robot is following: figure 5a.

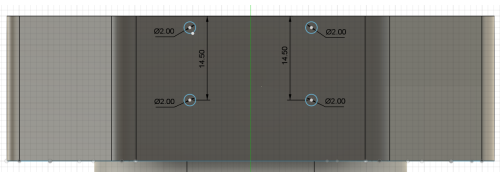
3.3.1 Camera Mount

The camera used for object detection is a Raspberry Pi camera module, which had to be installed on the robot with a custom support. It was designed using the official datasheet[2] and 3D printed in the lab.

The effect of the robot's oscillations on the camera should be minimized to prevent the object from getting out of the camera's field of view. With that in mind we decided to put the camera mount on the rods surrounding the battery module - this way the camera is as close to the wheels (the turning point) as possible, and there is no obstruction for the movement. The mount is designed in way that it clips on the rods, which facilitates an easy removal if the batteries have to be taken out: see figures 6a and 6b.



(a) Top-down sketch



(b) Side sketch

Figure 6: Camera mount

3.3.2 Testing Results and Discussion

Testing Methodology

The image processing subsystem was tested with different objects (size, shape, colour) and in different environments (lighting, density of objects) to ensure accurate object detection in a wide range of conditions.

Metrics for Evaluation

- **Accuracy**: How often does the algorithm detect the correct object?
- **Alignment**: How close is the detected object's outline to the actual one.
- **Setup sensitivity**: How much does a small deviation in the setup affect the other metrics?

Test Results

- For the majority of objects accuracy and alignment were high. However, if something thin (like a pencil) was far away from the robot, the robot failed to detect it correctly.
- The environments mostly affected the precision of the setup. For example, in an environment with many objects of a similar colour the "saturation" value had to be adjusted much more accurately to ensure that only one object is tracked.
- Some very small objects caused an issue, where the robot lost them when it got close enough to stop, as the lowest point of the camera's field of view rose higher then the highest point of the object.

Discussion
The algorithm performed as intended for almost every test scenario, since the setup process was specifically designed to allows great customization for a variety of cases. Hence, the robot should perform its task correctly for most of the usual objects. However, the algorithm failed at some edge cases, which shows that it isn't perfect.

The error rate can be improved nonetheless. The problem with thin or small objects can be solved by adding one more variable in the setup process, which controls the morphological opening. This variable should show the kernel applied to the mask, so that the user can choose how much they want to "blur" the image. The issue of losing the object can be resolved by writing an algorithm to search for an object. If the robot doesn't detect anything for a set amount of time, it can start moving in an outward spiral, which ensures that sooner or later the robot will find the object.

3.4 Communication

The image processing and manual control are done remotely on a server, so a communication subsystem is needed to link the server and the robot. During the design process we developed two communication systems with their own aims. The first one was made for quick testing of balancing and movement algorithms. The other one provides a more integrated and user friendly experience: it comes with a better-looking UI and an all in one control panel.

In both subsystem, WiFi is used to link the two ends and all information is be sent via an HTTP protocol. In our demonstration we will use a PC as server and an access point (AP) to provide WiFi connection. For the more integrated system, the client end will be only one Raspberry Pi, and for fast iteration system, the clients will be one Raspberry Pi with one esp32 .

The following table illustrate the integrated system use only raspberry as client.

Source	channel	Destination	Information
Camera	Mipi wire	Raspberry Pi	video stream
Raspberry Pi	Wi-Fi (HTTP, POST, image/jpeg)	Server	video stream
Server	Wi-Fi (websocket, text/JSON)	Raspberry Pi	movement control
Raspberry Pi	Wired (uart, 'wasd' letters)	ESP32	movement control

Table 9: Overview of communication information and channels for integrated system

An alternating approach: The communication section of this project establishes a connection between the ESP32 microcontroller and a client device, such as a laptop, via a WiFi network such as personal hotspot. This connection allows users to control a balance robot through a web-based interface by sending commands via HTTP requests. The following report explains the detailed working of the communication section, including the setup of the WiFi conection, handling HTTP requests, and integrating the user interface

3.4.1 Integrated System

Server This section will outline the functionality and implementation of a communication system server designed to run on a PC. In general, the server receives a video stream, displays the video on a PC using HTML served by flask, and captures 'wasd' key presses from the PC to send back to the client. We breakdown the goal into four functions and implement it individually.

- Firstly, network connection. The server is implemented in Python by creating a Flask app to serve HTML on PC's IP address, handling HTTP requests and use Flask-SocketIO library to provide real-time communication.

```
from flask import Flask, request, Response, render_template
from flask_socketio import SocketIO, emit

app = Flask(__name__)
if __name__ == '__main__':
    app.run(host='192.168.0.1', port=8080, debug=True)
```

- Secondly, receiving video stream. The server listens for incoming video data via a POST request and decodes the image data. Then stores the latest frame decoded data in server for displaying. The '/stream' route our app is the API we designed for client stream video to which will be use in next section.

```
# Global variable to store the latest frame
latest_frame = None

@app.route('/stream', methods=['POST'])
def stream():
    global latest_frame
    nparr = np.frombuffer(request.data, np.uint8)
    img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
    if img is not None:
        latest_frame = img
    return 'Frame received', 200
```

- Thirdly, serving HTML page and displaying Video. The server file system is organised as follow.

```
project_folder_root/
│
├── server.py
├── templates/
│   └── index.html
├── static/
    ├── css/
    └── js/
```

Figure 7: server file system

The Flask server will server our UI using index.html under templates folder. Then the HTML will displays the video stream by periodically fetching the latest frame which being stream to a endpoint '/video_feed'. The video is streamed to the endpoint by a video_feed() function. In the video_feed() function, we can change the source of lasted_frame to processed frame done by imaging subsystem. So that we can show the result on our UI.

```python
@app.route('/')
def home():
    return render_template('index.html')

@app.route('/video_feed')
def video_feed():
    global latest_frame
    if latest_frame is not None:
        _, buffer = cv2.imencode('.jpg', latest_frame)
        return Response(buffer.tobytes(), mimetype='image/jpeg')
    return 'No frame available', 404
```

In HTML, fetching lastest frame is done by a JavaScript as follow.

```javascript
        function fetchFrame() {
            const videoElement = document.getElementById('video');
            videoElement.src = '/video_feed?' + new Date().getTime();
        }
        // Fetch a new frame every 10ms
        setInterval(fetchFrame, 10);
```

- Lastly, handling key presses. The HTML page includes JavaScript code to listen for key press events. When a WASD key is pressed, the key press event is sent to the server via a WebSocket connection.

```javascript
var socket = io();

document.addEventListener('keydown', function(event) {
    var key = event.key.toLowerCase();
    if (['w', 'a', 's', 'd'].includes(key)) {
        socket.emit('keypress', {key: key});
    }
});
socket.on('key_update', function(data) {
    console.log('Key pressed:', data.key);
});
```

The server then broadcasts this key press event to all connected clients. The key press message are incoded in JSON format. At client end, it can be decode by indexing 'key' in JSON.

```python
@socketio.on('keypress')
def handle_keypress(data):
    key = data.get('key')
    if key in ['w', 'a', 's', 'd']:
        emit('key_update', {'key': key}, broadcast=True)
```

Client This section will outline the functionality and implementation of a client in your communication system designed to run on a Raspberry Pi. In general, the client system streams video to the server, receives JSON messages containing 'wasd' key press information, and forward the key presses to the UART port on Raspberry Pi. We breakdown this goal into four functions and implement it individually. The client side code is purely in Python.

- Firstly, streaming video. The client captures video from the camera on Raspberry Pi and streams it to the server. Video capturing is done by using opencv library or use picamera library. The choose of library use is depend on the hardware being used. During our development progress some of the camera can not be successfully detected. therefore Raspberry Pi official library picamera might being used.
  For the opencv version, we use cv2.VideoCapture(0) to get the frame to a buffer 'cap' in a while loop to get each frame.

```python
while True:
        # Capture frame-by-frame
        ret, frame = cap.read()
        if not ret:
            break
        # Send the frame to the server
        response = send_frame(frame)
```

For the picamera version, capture_continuous() will be used in a for loop to get each frame.

```
for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
    image = frame.array
    # Send the frame to the server
    response = send_frame(image)
    rawCapture.truncate(0)
```

The streaming is done by using request library and send 'POST' request with encoded frame to the sever.

```
def send_frame(frame):
    _, img_encoded = cv2.imencode('.jpg', frame)
    response = requests.post(server_url, data=img_encoded.tobytes(), headers={'Content-Type': 'image/jpeg'})
    return response
```

The server_url is the server IP address followed by '/stream', which is the flask server API designed in the last section.

- Secondly, receiving key press information. The key press information will be broadcast by flask_socketio. To get the message, client side code will run a websocket service. This websocket service will catch decode the broadcast from server and mean while run a message forward function describe later. The websocket service also include other supportive function such as on_error(), on_close() and on_open().

```
def on_message(ws, message):
    try:
        data = json.loads(message)
        if 'key' in data:
            key = data['key']
            send_uart(key)
            print(f"Key pressed: {data['key']}")
        else:
            print(f"Unexpected message: {data}")
    except json.JSONDecodeError as e:
        print(f"Failed to decode JSON: {e}")

def on_error(ws, error):
    print(f"Error: {error}")

def on_close(ws):
    print("Connection closed")

def on_open(ws):
    print("Connection opened")

def run_websocket():
    websocket.enableTrace(True)
    ws = websocket.WebSocketApp("ws://"+server_ip+":5000/socket.io/?EIO=4&transport=websocket",
                                on_open=on_open,
                                on_message=on_message,
                                on_error=on_error,
                                on_close=on_close)
    ws.run_forever()
```

- Lastly, forward key press message. The client send key press message using GPIO port of the Raspberry Pi and the message is send through uart communication. The clinet use RPi.GPIO and serial library to setup GPIO ports and start uart communication. The key press message will be sent are 'wasd' letters in ASCII code.

```
GPIO.setmode(GPIO.BCM)
ser = serial.Serial('/dev/ttyS0', 9600)

def send_uart(key):
    # Send the received key to UART
    if key in ['w', 'a', 's', 'd']:
```

```
            ser.write(key.encode())  # Send the key over UART
            print(f"Sent {key} over UART")
```

3.4.2 Fast Iteration System

WiFi Connection The ESP32 module connect to a WiFi network using the provided SSID and password, this is achieved by using the WiFi library, which handles the network connection. Once successful connection is setup, the local IP address of the ESP32 would be prompted up.

```
// Connect to WiFi
WiFi.begin(ssid, password);
Serial.print("Connecting to WiFi");
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
Serial.println("Connected!");

// Start the server
server.begin();
Serial.println("Server started");
Serial.print("IP Address: ");
Serial.println(WiFi.localIP());
```

HTTP Server The ESP32 board is treated as a server while the user's laptop is treated as a client. The server and the client must be in the same local area network. The ESP32 now listening for incoming client request. When the request is received. the server processes the request and sends an appropriate response.

Command handling Commands are sent from the UI to the ESP32 via HTTP Get request. The ESP32 processes these requests and executes the corresponding functions to control the robot.

3.4.3 Testing results and discussion

Testing Methodology
The communication system is aim to provide low delay connection and high quality video streaming. The system is been tested in different network environment and different devices. The environment we tested includes: a public Wi-Fi in a student accommodation with at least Hundreds of users and a personal AP with only server and client connected. The devices we tested are different computer with different computation power.

Metrics for Evaluation

- **Delay**: If the communication system produce perceptible delay that might affect robot control?
- **Video Quality**: How clear the video in slow and fast movement?
- **Signal strength**: How far the robot can go with good video quality?

Test Results

- The delay dose depend on the environment. For the test run in public Wi-Fi the delay can be over 100ms, which result in perceptible delay. However for the test using personal AP, the delay is usually less than 30ms. Sometimes the delay can be as low as 10ms which is even faster than the frame rate of the camera. In that case the internet delay can be ignore.
- The video quality is usually clear enough for image processing. Only when we use very poor computer such as another raspberry pi 3 as the server, the UI will sometimes be freeze. This might be due to limited RAM for the server app.
- For all environment, the video keeps clear and fluent when server and client are separated within 8-9 meters. For the public Wi-Fi, server and client can even communicate between floors.

Discussion
According to our testing, during the demonstration we will use our personal AP as priority choice for the network to avoid large delay and use a laptop as server to provide sufficient hardware for our server. In order to have greater range of movement we should place our AP away from the laptop, so that signal can be relay by the AP and allow our robot move further.
The communication system has more potential to transmit more data, such as gyroscope data. Therefore the communication system can be improve according to other subsystems' needs.

3.5 User Interface

The User Interface (UI) for the balance robot control system provides an intuitive and interactive method to manage the robot's movements via a web browser. This interface is served by the ESP32's web server and leverages HTML and JavaScript for its implementation. The following report details the functionalities available in the UI, how they are implemented, and the interaction between the user interface and the robot.

3.5.1 Layout and Styling

The UI is structured to be user-friendly and visually appealing. The layout is divided into two main sections: a sidebar and a main content area. The sidebar contains controls and status displays, while the main area is reserved for additional outputs such as maps or other visual feedback.

- Side bar: The sidebar includes sections for battery life, movement controls, and control parameters. Each section is styled to be distinct and easy to navigate.
- Main content area: This area is designed to display video stream and contains a switch to toggle image processing.

3.5.2 Interactive Elements

- Camera Output and Battery Life: Displays for real-time camera feed and battery status.
- Movement Control: Allowing the user to type "W" "A" "S" "D" on the keyboard to let the robot go around the arena.
- Control parameters: Allowing the user to adjust the PID parameters on a real-time basis, which allows the robot can adapt to various scenario.

3.5.3 JavaScript for Interaction

JavaScript is used extensively to handle user interactions and send commands to the robot, all of which are contained in the folder js folder that is within the UI folder. The key functionalities implemented with JavaScript include:

- Handling keyboard inputs: The script listens for events to capture W, A, S, and D key pressed, corresponding to forward, left, backward, and right movements, respectively. When a key is pressed, an HTTP GET request is sent to the ESP32 server with the appropriate command.
- Sending Commands: The JavaScript fetch API is used to send HTTP requests to the ESP32 server. Each command triggers a specific action on the robot, such as moving forward or turning left.
- Retrieving data and real-time display: Shows the live-stream captured by the camera as well as coordinates within the arena. These are shown in the camera scene in side-bar and main content area.

The UI for the balance robot control system is a well-organized and interactive interface that allows users to manage the robot's movements efficiently. By integrating HTML, CSS, and JavaScript, the interface provides a seamless experience for users to control the robot in real-time. The use of an ESP32 web server ensures that the interface is accessible from

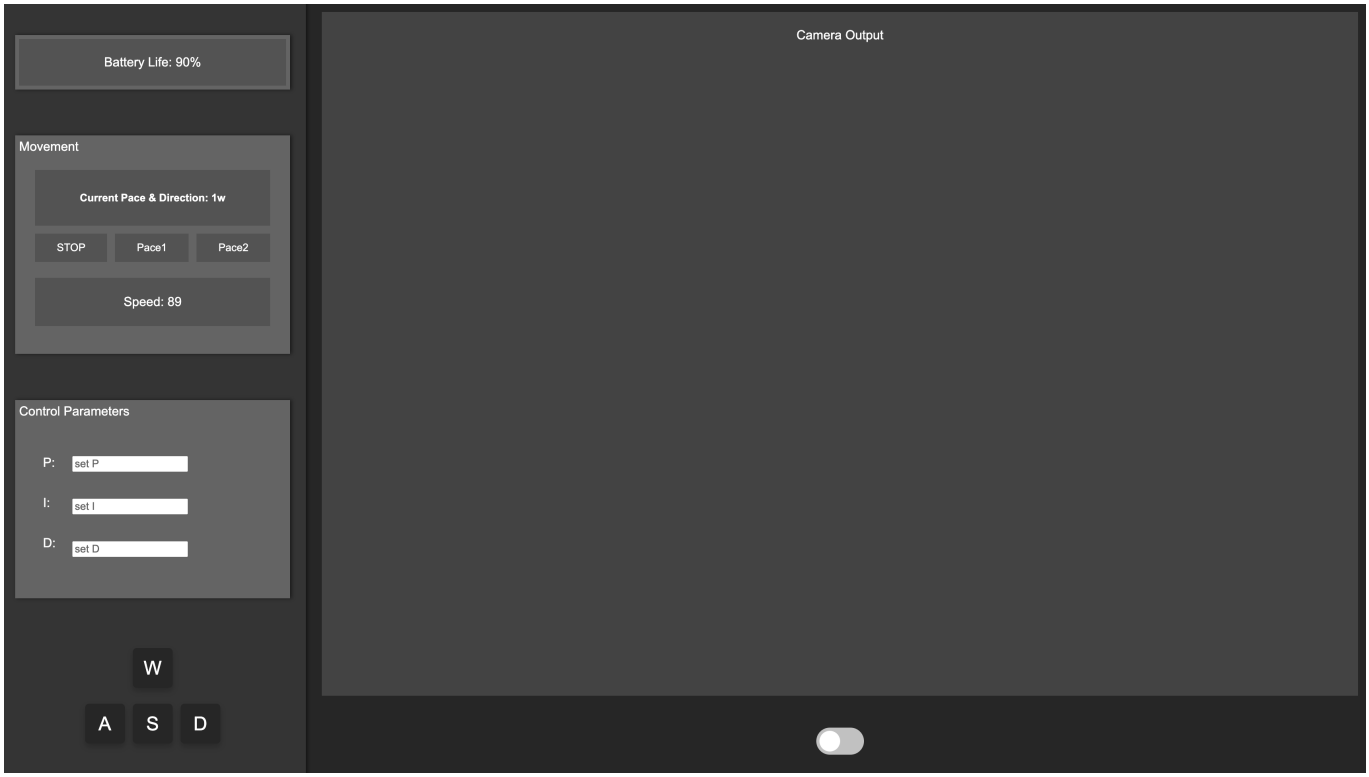


Figure 8: Our UI

3.6 Power

A key aspect for observing the robot's performance is analysing its power consumption: a robot with a low power consumption operates for longer periods of time before recharging, has a better battery life, and overall costs less to run. To observe the power consumption of the robot and the remaining charge in the battery, a series of sensing circuits are installed on the robot, connected to a Raspberry Pi via an ADC which processes the data to produce values for power dissipation in the logic and motor subsystems, and the remaining charge in the battery packs.

3.6.1 Potential Divide Circuit

The project utilises a set of 12 Overlander 2000mAh 1.2V batteries[3] in series to give a total rating of 14.4V. However, when testing the voltage output of 6 fully charged batteries, the volt meter measured 8.46V, giving a combined total voltage output for the 12 fully charged batteries of 16.92V. To reduce the possibility of damaging the circuitry (MCP3008 ADC and Raspberry Pi), the larger tested value will be used for calculations.

- The potential divider is made up of one upper resistor and two lower resistors in parallel. There are unusually 3 resistors in this subsystem (including the MCP3008 protective resistor), but the same principals apply for calculating the gain of the subsystem:
  - Equations used: $\frac{V_{out}}{V_{in}} = \frac{R_2}{R_1+R_2}$, $R_t = \frac{R_1 \times R_2}{R_1+R_2}$

Given the recommended value of the protective resistor is 100kΩ, implementing a resistor of value 33KΩ in parallel gives a combined resistance of 25kΩ. Based upon the aim to obtain a gain of ≈ 0.295 (16.92V input to $\approx$ 5V output), R_1 should be valued at 4.15MΩ. Rounding up to the next value in the E24 series, we will implement a 4.3MΩ resistor.

3.6.2 Volt-Meter Circuit

In the base design of the robot, two sensing resistors of 0.01Ω is in series between the power supply and each of the motor/logic subsystems. The resistor is high-side, so in order to isolate the voltage across the resistor, a comparator circuit is implemented as shown below. Given the small value resistance of the

sensing resistor, the output from the comparator needs to be boosted in order to increase the resolution of the data. A non-inverting op-amp circuit can solve this issue.

- The logic and motor subsystems both have current-limiting features to help protect the electronics within the subsystem. The motor subsystem contains a fuse rated at 2A, so we can assume that no more than 2 amps will pass through the motor subsystem's sensing resistor. The logic subsystem has a voltage regulator, which limits the current in the subsystem to 3A. To remove complexity when coding the processing algorithms for the Raspberry Pi, the maximum current for both subsystems will be assumed to be 3 amps: this will lower the resolution of the voltage sensing for the motor subsystem from its maximum possible resolution, but it will not have a major effect on the data.
  - Equation used: $\frac{V_{out}}{V_{in}} = 1 + \frac{R_f}{R_{in}}$

Assuming a maximum current flow in each subsystem as being 3A, we can deduce that the maximum voltage across the sensing resistors would be 0.03V. To utilise the MCP3008 ADC to the best of its ability, we aim to boost this 0.03V to 5V: a gain of ≈ 166. To achieve this gain, we can choose the following resistor values: $R_f = 510\text{k}\Omega$; $R_{in} = 3\text{k}\Omega$. This combination will produce a gain of 171 which is 5 more than what was proposed, but we can rectify this inconsistency during the data processing stage and proceed under the assumption that the current flow in the logic subsystem will not reach 3A.

3.6.3 Analogue-to-Digital Conversion

The raw data from the potential divider and volt-sense subsystems is in analogue form, and must be converted to digital form for numerical processing. The most efficient method for this project would be to utilise an ADC chip due to their compact size and ease of use. The chip chosen for the power aspects of the project was a single MCP3008 10-bit 8-channel ADC device.

- Pros: compact and easy to implement; includes SPI serial interface; 200ksps max sampling rate at VDD = 5V; 8 channels; high resolution (10-bit).
- Cons: only one channel can be converted at a time; costly.

The channel selection is controlled automatically by the Raspberry Pi, and data is pulled from each channel in a cyclical nature, ensuring that each reading stays as up-to-date as possible.

3.6.4 Data-Processing

Certain elements of the processing involved in converting the digital data from the MCP3008 ADC to their respective values have previously been mentioned, so a methodical breakdown of each process will help to better understand the code:

- Channel 1: Remaining Battery Charge:
  - The data outputted from the MCP3008 chip is converted from 10-bit binary to a numerical voltage by multiplying by the reference voltage and dividing by the resolution.
  - The voltage is divided by the voltage divider ratio 4.3MΩ/25kΩ to give the total voltage outputted by the battery.
  - The program uses a look-up table which has assigned percentage values for their corresponding voltage level. It selects the value the calculated voltage is closest to and returns the percentage charge remaining.
- Channel 2 and 3: Power Dissipated in the Logic and Motor Subsystems:
  - The data outputted from the MCP3008 chip is converted from 10-bit binary to a numerical voltage by multiplying by the reference voltage and dividing by the resolution.
  - The voltage is divided by the product of the gain of the non-inverting op-amp (171) and the sensing resistor resistance (0.01Ω) to give the current flowing through the subsystem.
  - The calculated current value is multiplied by the regulated supply voltage (5V) to calculate the power dissipated in the subsystem.

These three values are then sent to an external flask server with a UI to view the data.

3.6.5 User Interface

To properly test the user interface, the UI for the power will be a stand-alone UI adjacent to the UI for the rest of the project. The design for the UI is simple, but conveys the data clearly. A graphical representation of the power dissipation for both the motor and logic systems has been included to help track the robot's operation and more easily evaluate its performance.

3.7 Audio

3.7.1 Speech Recognition

Speech recognition is a an extension of the presentation task the robot perform and is a stretch goal, as it is still under development. When this subsystem is finshed, the robot should starts moving forward when an observer says "hi" to the robot. In order to achieve this target, a speaker is connected to raspberry pi

and the input voice message is transferred, using an online service, such as Google's Web Speech API, to convert the audio input into text. This involves sending the audio data to the API, which then returns the recognized text. In the coding, we introduce the speech_recognition library that listens for and processes audio input.

3.7.2 Noise Filter

Initially, when we tried to test, we found the system barely get the correct words especially with the noise. After that, we introduce a noise filter part which is adjust_for_ambient_noise method and it helps in reducing the background noise, making it easier to recognize the actual speech. The working principle of the system is the system captures the noise and adjusts the recognition process to filter out background sounds. The coding includes mechanisms to handle cases where the speech is not recognized or if there is

an error in the request to the speech recognition service as well. Moreover, if the text matches "hi", the

system recognizes it as a valid command to perform a specific action and we can adjust the action in the control part for example sending command to the motors to finish the action

3.7.3 Future Development

Based on the coding, we can also make the robot have different interesting actions with different voice commands, for example linking "Liam" to move backward. Finally, we added the part for exit so that if we say "exit", the script will stop and exit completely.

3.8 Integration

Integrating the various subsystems of our balance robot correct was a must to ensure smooth operation and overall functionality. Our approach was methodical and incremental, allowing us to troubleshoot and optimize each component before combining them into the final system.

Balancing Subsystem
The balancing subsystem was the first to be integrated, as maintaining the robot's upright position is fundamental to its operation. As described In 3.1 we implemented a cascaded loop PID control system, with an inner loop dedicated to tilt control and an outer loop for speed regulation. This setup ensured that the robot could adjust its tilt in real-time to maintain balance while responding to speed commands.

Movement Subsystem
Next, we integrated the movement subsystem. Initially, movement was controlled using only the inner loop of our balancing system to ensure stability. Once we achieved reliable balancing, we incorporated the outer loop to manage speed. This allowed the robot to move forward, backward, and turn accurately, with precise adjustments made through the stepper motors.

Image Processing Subsystem
Image processing was crucial for enabling the robot's autonomous navigation. Using a Raspberry Pi and its camera module, we implemented an algorithm to identify and track specific objects. The image data was processed to generate commands for the robot, such as moving towards an object or adjusting its path. This subsystem required careful tuning to handle various lighting conditions and object types.

Communication Subsystem
The communication subsystem linked the robot with a remote server, allowing for both autonomous control and manual overrides. We used WiFi to facilitate real-time data transmission between the robot and the server. The server receives video streams from the robot and sends back control commands based on image processing results or user inputs from the web-based interface.

User Interface Subsystem
Finally, we integrated the user interface, which provided a platform for manual control and live monitoring. The interface displays essential information such as battery status, speed, and current view from the robot's camera. Users could switch between autonomous and manual modes, providing flexibility in controlling the robot.

3.8.1 Testing Results and Discussion

Testing Methodology
Integration testing was performed iteratively. Each subsystem was tested individually and then in combination with others. We faced several challenges, such as synchronization issues and communication delays, which were addressed through debugging and optimization. By incrementally adding and testing each component, we ensured that the final integrated system was stable and responsive.

Metrics for Evaluation

- **Robustness**: Can the whole system operate continuously, without any critical errors?
- **Functionality**: Do all of the components of the system successfully work together?
- **Ease of Use**: Is it easy to react to and interact with the system, allowing us to use it as we desire?

Test Results

- The integrated parts of the system for the majority of the time operate correctly with minor errors such as delayed communication occuring infrequently. However overall the system is relatively robust and can survive even in occasional extreme operating conditions, such as recovering from a collision.
- All the subsystems perform satisfactorily together, achieving our initial goals.
- The interaction with the robot is intuitive and effective, we can monitor all key data and switch easily between manual and autonomous control. There is also ability to expand the scope of the users interaction and control of the robot through our current architecture.

Discussion
The integration of our robot's subsystems was a complex but rewarding process. By following our structured approach, we successfully combined balancing, movement, image processing, communication, and user interface subsystems into a cohesive and functional robot. If we were to do it again we would aim to have greater team cohesiveness and communication as it became hard to track the progress of some of the members. Our final code is located in this GitHub repository:
`https://github.com/Arc-Cloud/BalanceRobot`.

4 Project Management

4.1 Role Management

Team Bonus is a group of six members. Each member was assigned a certain role within the project, although some necessary changes have be made along the way, all of which count as contributions to the project. These roles were mainly allocated from personal preference, as all the members volunteered to work on a certain aspect of the project where they have a stronger understanding and interest in. The table below displays the responsibilities of each member.

Name	Responsibilities	GitHub Username
Xiaoyang Xu	Balancing, Chassis assembling, Communication	X454XU
Maximilian Adam	Balancing, Movement, Communication and UI	Arc-Cloud
Yueming Wang	Communication and image processing	rrroooyyywang
Fedor Dakhnenko	Movement, Image processing, 3D printing	fedorrrd
Yingkai Xu	Audio	LiamXu423
Maxim Kelly	Battery life measurement	mjpk109

Table 10: Roles Assigned

This also meant that during the interim group presentations, as well as the report, that each member had a minimum responsibility for creating their own slides and explaining their assigned section of the project. Otherwise members could freely contribute to the shared sections within the presentation or the report, which include topics such as Design Specification, or even addressing the communication and teamwork within the group. It is important to note that while each member has been assigned a particular role in the project, all members were encouraged to help each other, ensuring that members have a degree of experience and contribution in all parts of the project.

4.2 Report Contribution

This report was developed through a collaborative effort, bringing together expertise and insights from various team members. Each individual contributed their specialized knowledge, ensuring that the report encompasses a wide range of perspectives and in-depth analysis. The following table describe how each team member contributed to the report.

Name	Main Contributions	Assisted with
Xiaoyang Xu	UI, Project Management and Appendix	Balancing, Communication
Maximilian Adam	Abstract, Introduction, System Design, Balancing, Movement (Cascaded), Integration, Conclusion	Project Management
Yueming Wang	Communication, UI	System Design
Fedor Dakhnenko	Movement (inner loop), Image Processing, Camera Mount, Proofreading	Project Structure, Communication
Yingkai Xu	Audio	
Maxim Kelly	Power	

Table 11: Contributions

4.3 Planning Management

First and foremost, all of our group meetings have been held in the laboratory. These meetings are held at least once a week, which allowed us to view the current work we have completed and possibly test/implement any new modifications that have been made to the final design. Aside from our regular meetings, we also agreed to work on our project in the laboratory usually 4 days each week for at least 6 hours a day. This meant we were able to work with full access to the equipment in the lab and allowed us to cooperate and communicate with each member if any issues or uncertainties arose.

The team's progress, deliverables and milestones have been summarised in technical documents which have been approved and regularly modified by every member. We effectively keep track of our work and progress through the following methods.

- Microsoft Excel: At the start of the project, we created a brief timeline of what we want to achieve, which gave us an overall understanding of our goals and aims.This allowed us to complete our tasks by the agreed timeline. Then, we created a more detailed planning system by using the Microsoft Excel. These are tables where we each write what we aim to do everyday for the upcoming week as a planer. Then after the wee, we write down what we actually accomplished over the week. This means that everyone in our team knows our responsibilities and who to communicate with in terms of a particular aspect of the project. Additionally, this sets a goal for each member to achieve the goals that they set, which ensures substantial progress in our project.
- GitHub: This is an platform which allows us to store and post any completed work, and is fully visible to any member for modification. This was an extremely useful tool for the organisation of our work because our project consists of a number of algorithms, with a minimum of one per each sub-system. in addition to a few more for the movement of the balance robot. These code also had to be combined together, as they would become functions that are called by the main code. Furthermore, GitHub has an incredibly useful function where users can quickly access all the connected files in the project in VSCode, through the GitHub. This saved us a significant amount of time and effort when accessing files and codes within the project.

Figure 9: Gantt Chart for Milestones

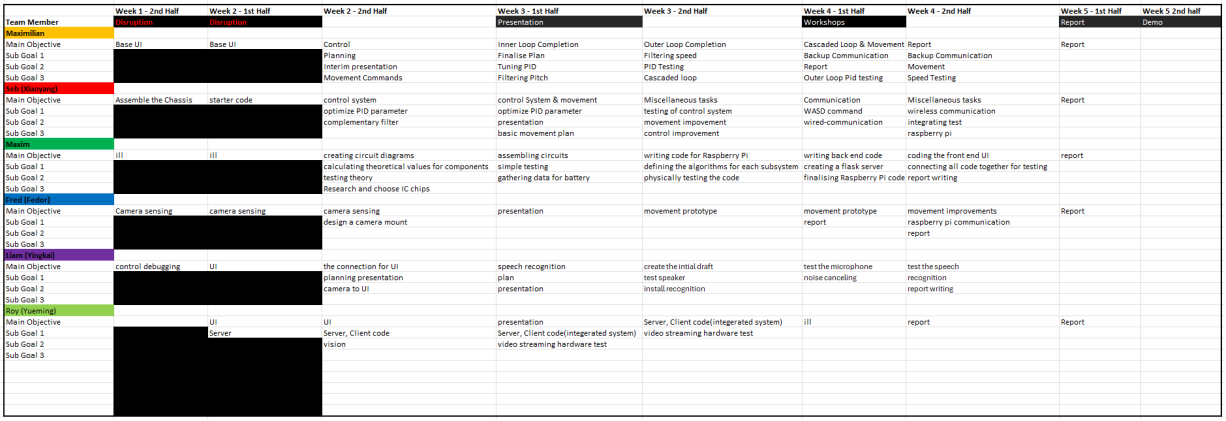


Figure 10: Gantt Chart for Roles

4.4 Budget Management

4.4.1 Actions Taken for Budget Efficiency

The permitted budge for the Balancing Robot project is 60 (GBP). Orders can only be made by the treasurer and must be approved by at least people who focus in the targeted area. To maximize cost efficiency of the project, the purchase of new components was carefully considered against a specific set of criteria, with a view to ensuring value for money.

- Careful consideration of hardware options available.
- Comparison of technical datasheet to ensure that the components meet the desired performance.
- Consider the Generality of the purchased product to make sure it fits as much device as possible
- ease of fixing technical issues, as the team is aware of the connections between each fundamental device.
- reduction of costs due to unnecessary components

4.4.2 Table of Item Purchased

The following table demonstrates a detailed version of which product was bought and how many has been purchased, as well as the total price.

Device Name	Number purchased	total price (£)
Raspberry Pi camera module	2	31.02
Operational amplifier	5	0.46
Analog to digital converter	3	7.47

Table 12: Purchased Items

5 Conclusion

In conclusion, the development and testing of our balancing robot has yielded positive results, demonstrating the robot's ability to balance effectively, identify objects, and execute movement commands. While there were challenges due to the imbalance between EEE and EIE students in our team, we managed to overcome these through collaboration and dedication of the members present.

The robot's performance in balancing and movement control met our expectations as evidenced by the successful long-term balancing and responsive movements observed during testing. However, there is still room for improvement, particularly in enhancing our robot's overall functionality. Future work would require us to refine both hardware and software components to improve their interaction and usability.

Overall, the project was a significant learning experience, offering a practical application of our theoretical knowledge and fostering our teamwork and problem-solving skills. With continued effort and development, our balancing robot has the potential to become a highly able and versatile autonomous system, and we aim to continue our development right up until the demonstration task.

6 Appendix

A Code Listings

A.1 Filtering Code

```
void calculateButterworthCoefficients() {
  double pi = 3.141592653589793;
  double sqrt2 = 1.4142135623730951;
  double normalizedCutoff = cutoffFrequency / (samplingFrequency / 2.0);
  double theta = 2.0 * pi * normalizedCutoff;
  double d = 1.0 / cos(theta);
  double beta = 0.5 * ((1 - (d / sqrt2)) / (1 + (d / sqrt2)));
  double gamma = (0.5 + beta) * cos(theta);
  double alpha = (0.5 + beta - gamma) / 2.0;

  b[0] = alpha;
  b[1] = 2.0 * alpha;
  b[2] = alpha;
  a[0] = 1.0;
  a[1] = -2.0 * gamma;
  a[2] = 2.0 * beta;
}
```

A.2 Auto Tuning Code

```
import time
import re
import subprocess
import os

# Define the path to your main.cpp file relative to the location of the Python script
main_cpp_path = os.path.join(os.path.dirname(__file__), '../src/main.cpp')

# Define the initial Kd value
initial_kd = 10
kd_increment = 5

# Define the number of increments
num_increments = 10

# Function to modify the Kd value in main.cpp
def modify_kd(kd_value):
    with open(main_cpp_path, 'r') as file:
        lines = file.readlines()
```

```
    with open(main_cpp_path, 'w') as file:
        for line in lines:
            if 'double kd =' in line:
                line = re.sub(r'double kd = \d+\.?\d*;', f'double kd = {kd_value};', line)
            file.write(line)

# Function to upload the code to the robot using PlatformIO
def upload_code():
    project_root = os.path.join(os.path.dirname(__file__), '../')
    result = subprocess.run(['platformio', 'run', '--target', 'upload'], cwd=project_root, capture_output=True)
    if result.returncode == 0:
        print("Upload successful!")
    else:
        print(f"Upload failed: {result.stderr.decode()}")

# Main loop to increment Kd and upload the code
kd_value = initial_kd
for i in range(num_increments):
    kd_value += kd_increment
    print(f"Setting Kd to {kd_value}")
    modify_kd(kd_value)
    upload_code()
    time.sleep(10)
```

```
import serial
import csv
import time

# Serial port settings
serial_port = '/dev/tty.usbserial-120'  # Replace with your port name, e.g., /dev/ttyUSB0 on Linux
baud_rate = 115200
timeout = 1

# Open serial connection
ser = serial.Serial(serial_port, baud_rate, timeout=timeout)

# File names
csv_file_name = 'output.csv'

# Open CSV file
with open(csv_file_name, mode='w', newline='') as csv_file:
    csv_writer = csv.writer(csv_file)

    # Write CSV header
    csv_writer.writerow(['Timestamp', 'Tilt (mrad)', 'Tilt (deg)', 'Step1 Speed', 'Step2 Speed', 'Setpoint'])

    try:
        while True:
            # Read line from serial port
            line = ser.readline().decode('utf-8').strip()

            if line:
                # Parse the line (assuming CSV format sent from Arduino)
                data = line.split(',')

                # Add a timestamp
                timestamp = time.strftime('%Y-%m-%d %H:%M:%S')
                data.insert(0, timestamp)

                # Write data to CSV file
                csv_writer.writerow(data)

                # Print to console (optional)
                print(data)

```

```
    except KeyboardInterrupt:
        print("Interrupted by user")

    finally:
        ser.close()

print(f"Data written to {csv_file_name}")
```

B Supplementary Materials

B.1 External Libraries

- OpenCV - open source Python library for image processing: `https://docs.opencv.org/4.x/`
- WiFi.h - A C++ library used to achieve wireless communication.
- Arduino.h, Adafruit_MPU6050.h, Adafruit_Sensor.h are all C++ libraries used for ESP32 manipulation.
- chrono.h - A C++ library used for time functions.

C Special Thanks

- A special thanks to Dr. Stott helped us assembled the robot and connect the circuit.
- A special thanks to Dr. Abd Al Rahman Abu Ebayyeh for the consulting session.

References

[1] D. E. Stott, *Balance Robot Project Brief*, 2024. [Online]. Available: `https://github.com/edstott/EE2Project/blob/main/doc/balance-guide.md`.

[2] R. Pi, *Raspberry pi camera module 3 standard mechanical drawing*, www.raspberrypi.com. [Online]. Available: `https://datasheets.raspberrypi.com/camera/camera-module-3-standard-mechanical-drawing.pdf` (visited on 06/17/2024).

[3] O. batteries, *Product specification: Subc 2000mah 1.2v*, overlander.co.uk. [Online]. Available: `https://www.overlander.co.uk/pub/media/datasheets/2000_SubC_1S.pdf` (visited on 06/17/2024).