

# **Documentation of Technical Details for Proposed Robot**

**by**

**Team Stabilize:**

**Team Members:**

**Avishikta Bhattacharjee**

**Tanushree Dutta**

**Md Adeeb Eqbal**

**Ayush Maity**

**Mentor Name: Dr. Nitin Sharma**

**July 2024**

**Kalinga Institute of Industrial  
Technology, Deemed to be University**

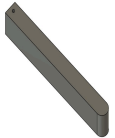
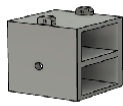
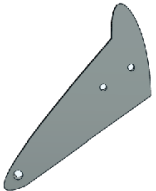
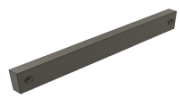

# **Table of Contents**


1. Type of Robot
2. CAD drawing
3. Hardware Identification and Realization
4. Proposed Methodology
5. Application of Proposed Robot
6. Dimension
7. Proposed Timeline
8. Proposed Outline of the Two Wheel Self Balancing Robot

# 1.Type of Robot

We propose a two-wheeled self-balancing robot (Fixed legs i.e the height of the bot will remain the same for small versions and movable i.e the height can be adjusted by folding of legs for the actual version). The robot exhibits a remarkable ability to maintain an upright posture on just two wheels, akin to a skilled cyclist.

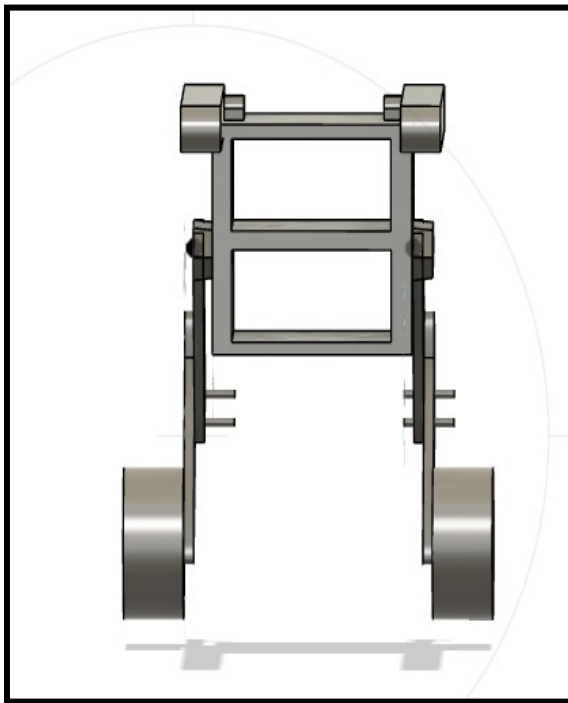
Dimensions of the different parts of the robot:

Parts	Images	Length	Breadth	Height/thickness	Use/Description
Arm		20 cm	2 cm	2 cm	This will help to give a self righting mechanism if the bot falls , as well as, help to lock onto the corners for support and
Box		10 cm	10 cm	10 cm	The box is divided into two segments. The microcontroller will be in the top part of the box, the gyroscope sensor on top of the box and motor driver and battery at the bottom part of the box.
Shin		12 cm	5.6 cm	0.5 cm	This part joins the two links and the wheel so that by giving motion to the servo at the box we can change the height to adjust its COG.
Leg Link 1		10 cm	1.5 cm	0.5 cm	This is the smaller link to join the servo at the box and shin.
Leg Link 2		11 cm	1.5 cm	0.5 cm	This is the longer link to join the servo at the box and shin.

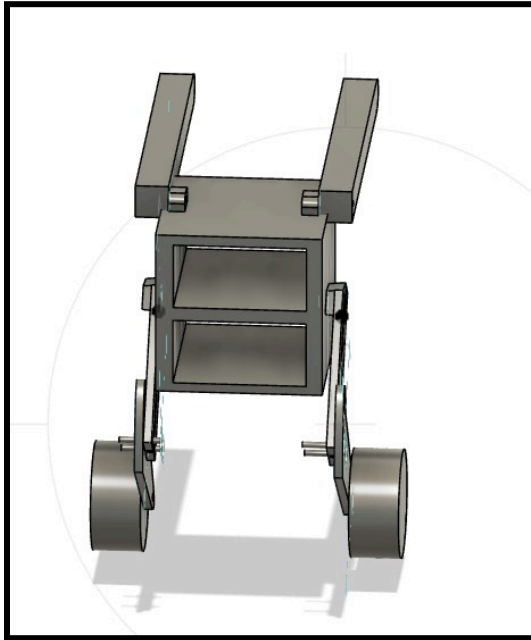
Wheels (for actual version)		8.5 cm	8.5 cm	3.9 cm	We have opted for broad-width wheels to enhance traction and stability, particularly on uneven surfaces.
-----------------------------	---	--------	--------	--------	--

**Table 1:** Dimensions and usage of the parts of the robot.

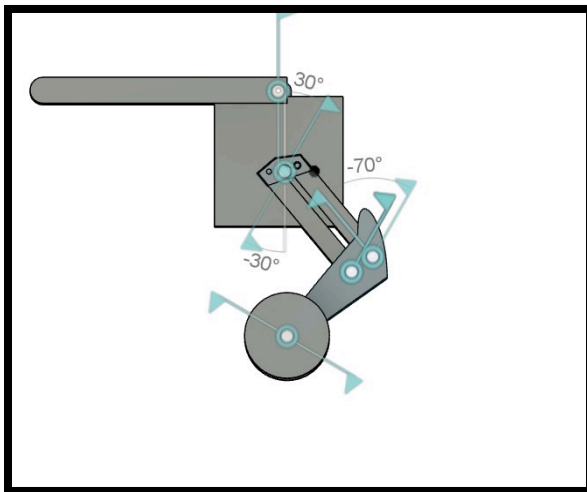
## 2.CAD drawings of Two Wheel Self-balancing Robot (TWSR)



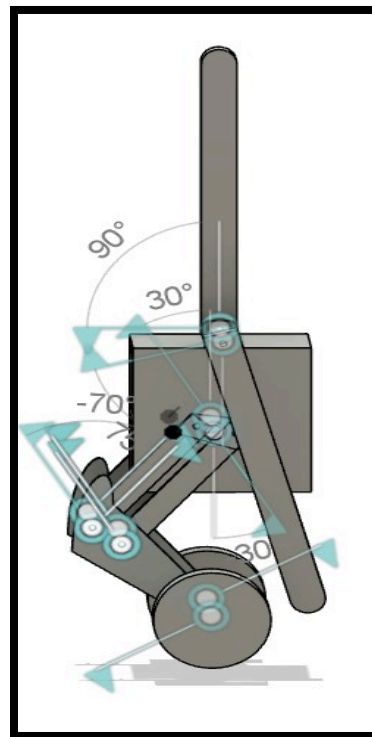
**Fig 1:** TWSR front view with arms



**Fig 2: TWSR back view with arms**



**Fig 3: TWSR side view with arms**



**Fig 4: TWSR side view with extended arms**

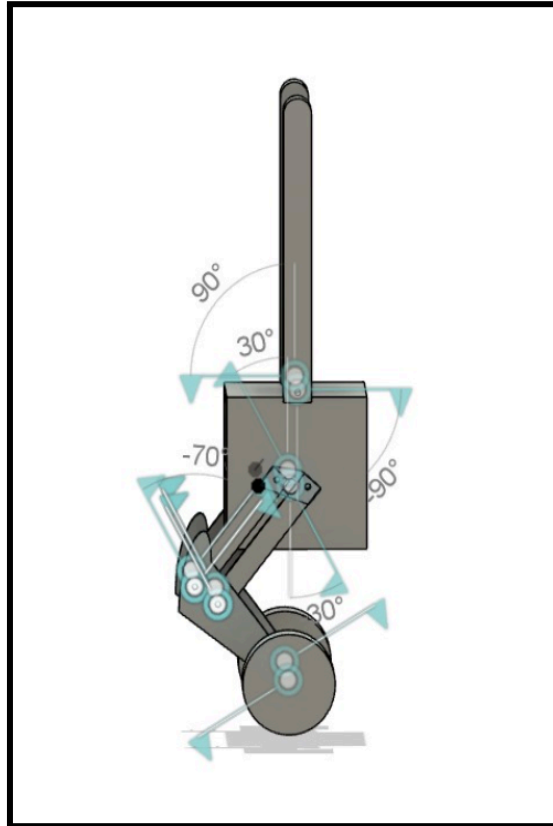


Fig 5: Resting (initial )state of TWSR side view with extended arms

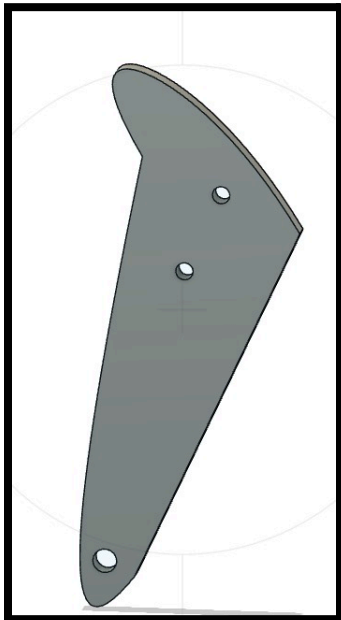


Fig 6: TWSR shin side view

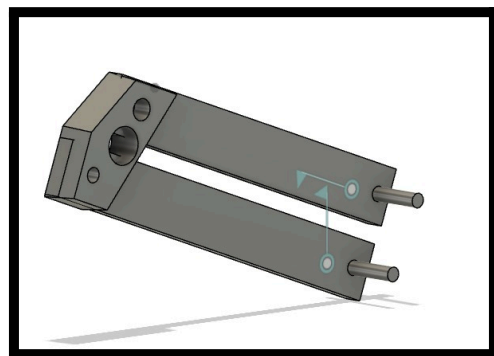
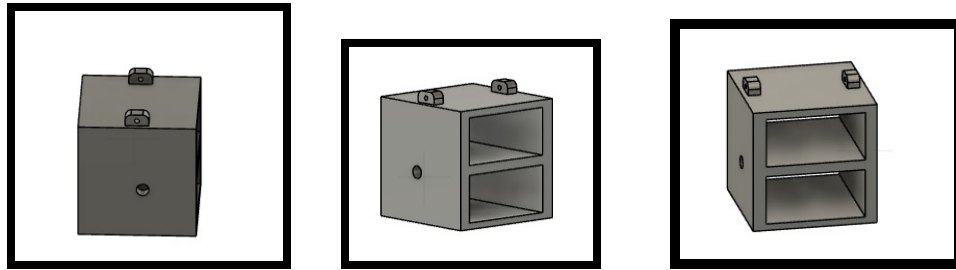
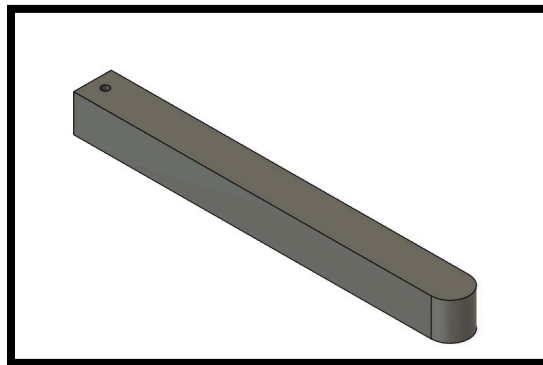


Fig 7: TWSR legs side view



**Fig 8: Body of TWSR**



**Fig 9: TWSR arm**

### 3. Hardware Identification And Realization

Sl No.	Hardware	Category	Quantity Needed	Justification	Estimated Cost
1	3D printed Arm	Structure Components	2	This will help to give a self righting mechanism if the bot falls , as well as, help to lock onto the corners for support .	As required for 3D printing
2	3D Printed Shin	Structure Components	2	This part joins the two links and the wheel so that by giving motion to the servo at the box we can change the height to adjust its COG.	As required for 3D printing
3	3D Printed Link 1	Structure	2	This is the smaller link to join the	As required for

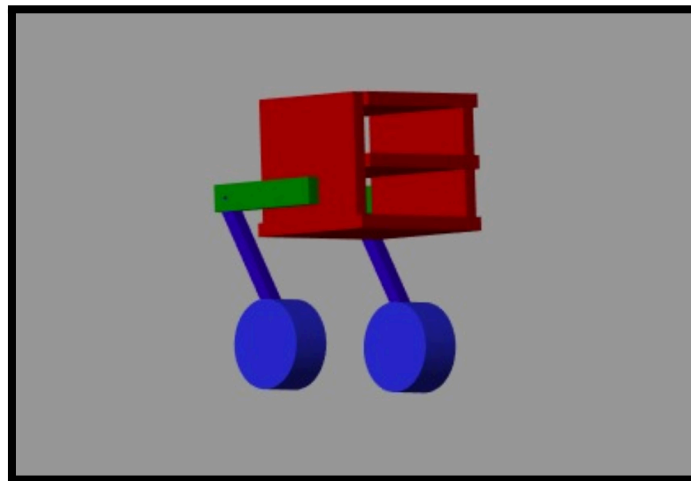
		Components		servo at the box and shin.	3D printing
4	3D Printed Link 2	Structure Components	2	This is the longer link to join the servo at the box and shin.	As required for 3D printing
5	3D Printed Body	Structure Components	1	The box is divided into two segments. The microcontroller will be in the top part of the box, the gyroscope sensor on top of the box and motor driver and battery at the bottom part of the box.	As required for 3D printing
6	Robotic Wheels (85 x 39 mm)	Motion Components	2	Due to their appropriate size, which offers exceptional traction, stability, and mobility, they are perfect for self-balancing robots..	
7	Pro-range NEMA17 PR42HS40-1204 AF-02 4.2kg-cm Stepper Motor-D Type Shaft	Electronic Components	2	Because of its dependable performance and 4.2 kg-cm holding torque, it is perfect for high-torque applications that need precision. It provides outstanding stability, accuracy, and interoperability with a range of control systems..	Rs. 649 each
8	TB6600 Stepper Motor Driver Controller 4A 9~42V TTL 16 Micro-Step CNC 1 Axis	Electronic Components	1	The large voltage range, precision micro-stepping capabilities, and high current capacity of the TB6600 Stepper Motor Driver Controller (4A, 9~42V, TTL, 16 Micro-Step, CNC 1 Axis) make it the perfect choice for CNC usage.	Rs. 249
9	MPU-6050 3-Axis Accelerometer and Gyro Sensor	Electronic Components	1	The combined accelerometer and gyroscope allows for precise and steady motion sensing, making it perfect for motion tracking applications.	Rs. 99
10	Double speed measuring, module encoder photoelectric, pulse output module	Electronic Components	1	For accurate speed measurement in robotics and automation, the Double Speed Measuring Module with Encoder Photoelectric Pulse Output is perfect. It offers precise pulse output for feedback and high-resolution speed sensing	Rs. 326
11	ESP-32	Electronic Components	1	It is renowned for its adaptability and comes with a wide range of peripherals, Bluetooth, Wi-Fi, and dual-core processors.	Rs. 359



12	TowerPro MG90S Mini Digital Servo Motor (180° Rotation)	Electronic Components	2	Because of its high torque, metal gears, and 180° rotation, it is perfect for small and precise applications.	Rs. 119 each
13	4S Lithium Polymer Battery (14.8 2500mAh)	Electronic Components	1	Because of its high energy density, steady voltage output, and lightweight construction, it is perfect for high-performance applications.	Rs. 3000
14	Breadboard	Electronic Components	1	Normal small breadboard to fit into the body	Rs. 39
15	Jumper Wires	Electronic Components	As needed	Connections	Rs. 1 each
16	Plastic Eyes	Other Accessories	2	Decoration	As required
17	Printed Paper	Other Accessories	As needed	Decoration	As required

**Table 2:Hardware Identification And Realization**

## 4.Proposed Methodology



**Fig10: Proposed structure of our two-wheeled self balancing robot in Simulink**

We are developing a two-wheeled self-balancing robot, with both a small and an actual version being crafted concurrently. To facilitate a comprehensive understanding of the project, we provide all essential technical specifications. These specifications are divided into two key components: Design Methodology and the Working Methodology. This structured approach ensures clarity and thoroughness, aiding in the effective realization of both robot versions.

## 4.1.DESIGN

### 4.1.1. Small Version

In a two-wheeled self-balancing robot, the center of mass (COM) plays a crucial role in maintaining stability and balance. The COM is the point where the mass of the robot is considered to be concentrated and around which all parts of the robot are balanced. For a self-balancing robot, precise control of the COM is essential to prevent tipping over and to respond effectively to changes in the environment.

For our two-wheeled self-balancing robot, we have identified three main Centers Of Mass (COM) based on the weighted average positions of individual components. Each COM is critical for the robot's stability and performance. By taking inspiration from the mechanics of Segways and hoverboards, which utilize gyroscopic sensors and accelerometers to maintain balance, we incorporate similar feedback and control systems. These systems dynamically adjust motor speeds to ensure the robot remains upright, even on uneven surfaces.

Our design focuses on a lower center of gravity by positioning heavier components, such as batteries and motors, near the bottom, and ensuring balanced weight distribution around the vertical axis.

#### 4.1.1.a. Design Principles

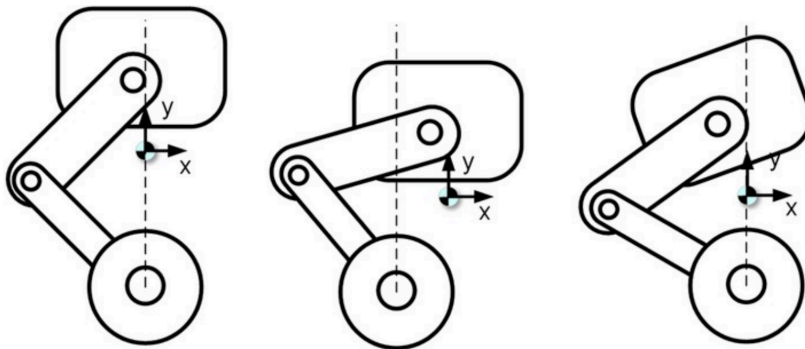


Fig11: Better balance for TWSR due to COM being higher relative to the wheel axis.


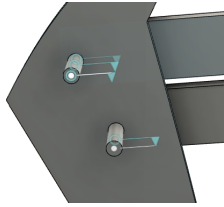
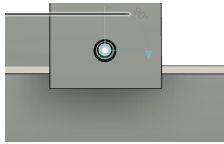
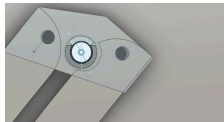
**Considered COM of TWSBR:**

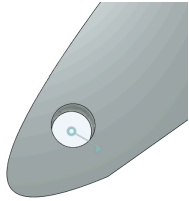
- a. Body COM
- b. Thigh-Shin COM
- c. Wheels-Axle COM

## Relationship Between Error from Motion Planning (MP) and Centre of Mass (COM)

- **Error from Motion Planning (MP):** Motion planning involves creating a path or trajectory for the robot to follow. Errors in motion planning can arise due to incorrect calculations, obstacles, or changes in the environment.
- **COM Relationship:** The COM of the robot needs to be managed in relation to the errors in motion planning. If the COM is not properly aligned or balanced, the robot might tip over or fail to follow the planned path accurately. Balancing the COM is critical to compensating for any discrepancies or errors from motion planning. A well-balanced COM ensures the robot remains stable even if there are slight deviations from the planned trajectory.

### 4.1.1.b. Joint Mechanics and Control

Part	Joint	Position	Image of joint	motion,,,,
Torsion spring damper	Spring joint	Between thigh and shin		Semi rigid will push the legs to its initial position (60°) as soon as the force is withdrawn.
M4 screws	Revolute	Between thigh and shin		M4 screws give free motion to the links but hold tightly to the shin.
Servo motors	Revolute	Between arm and body		Servo motors control the arm with freedom of 360°
Servo motors	Revolute	Between thigh and body		Servo motor control both the leg links (link 1 & link2) in a coupling motion of +5° and -5°.

Stepper motor	Revolute	Between wheel and shin		360° rotation by the stepper motor.
---------------	----------	------------------------	--	-------------------------------------

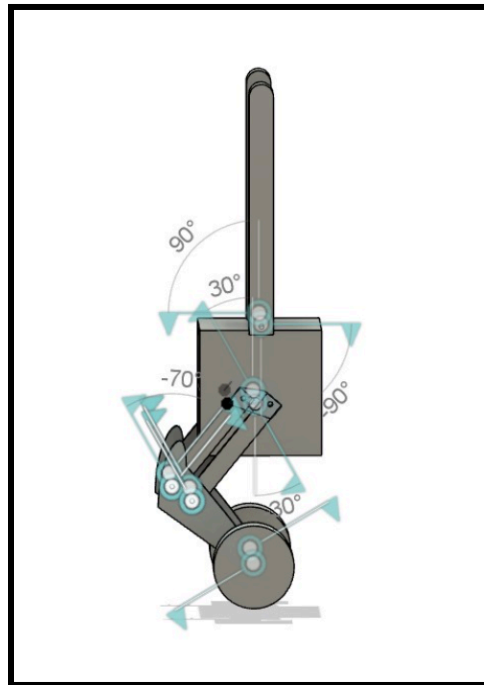
**Table 3:** Joint mechanism and control

## 4.1.2.Actual Version

When it comes to the actual version of the two wheeled self-balancing robot, there are just a few minor changes with respect to the smaller version. The necessary changes are all explained briefly.

### 4.1.2.a.Self-Righting Mechanism

Our self-balancing robot can handle extreme tilt angles up to 90 degrees using a robust control strategy. By adjusting wheel speeds incrementally with feedback from sensors, the PID controller brings the robot back to an upright position autonomously. This mechanism allows the robot to regain balance even from a horizontal position, showcasing its high resilience and adaptability in maintaining equilibrium.



**Fig12:** TWSR with extended arm.

### 4.1.2.b.Optimizing Joint Function

**Revolute Joint between Thigh and Shin:** In our actual version of the robot, opting for a revolute joint between the thigh and shin components proves to be a superior choice compared to incorporating a damper. By allowing rotational motion between the thigh and shin, the revolute joint facilitates dynamic adjustments in response to changes in terrain and external forces. This dynamic range of motion enables the robot to maintain stability while adapting to varying conditions, enhancing its agility and performance. Additionally, a revolute joint aligns with the

natural biomechanics of human limbs, promoting a more efficient and intuitive motion control mechanism for our robot.

### 4.1.2.c.Power Source

The 14.8V 2500mAh 25C 4S Lithium Polymer Battery Pack is ideal for a two-wheeled self-balancing robot, offering benefits like heavy-duty discharge leads, gold-plated connectors, and JST-XH balance connectors. Its high voltage ensures efficient motor and control system operation, while the 2500mAh capacity balances runtime and weight. The 25C discharge rate provides high currents for quick responses and stability during accelerations or corrections. Overall, this battery pack boosts the robot's performance, responsiveness, and endurance.

### 4.1.2.d.Position of Sensors

For optimal weight distribution and efficient management, we've strategically positioned our sensors on the top half of the robot. The ESP32 controller, coupled with an MPU6050 unit, is placed on the upper section, oriented along the Z-axis towards the Earth. This placement ensures precise sensing of the robot's orientation and motion. Additionally, on the lower half, we've allocated space for the L298 motor driver and batteries. This arrangement not only facilitates smooth weight distribution but also allows for convenient access and maintenance.

## 4.1.1.b.WORKING METHODOLOGY

Our control system for the two-wheeled self-balancing robot utilizes feedback control loops to maintain balance and control the heading direction. The feedback control loop continuously measures the robot's tilt angle and position, compares them to the desired values, and adjusts the wheel speeds to correct any errors. This system ensures the robot remains balanced and follows the intended path or position, dynamically responding to any disturbances or changes in the environment. The PID control algorithm offers a balanced approach to error correction, allowing the robot to maintain stability and follow the desired trajectory with high precision and responsiveness. This results in a more efficient and robust self-balancing system capable of adapting to various conditions and disturbances.

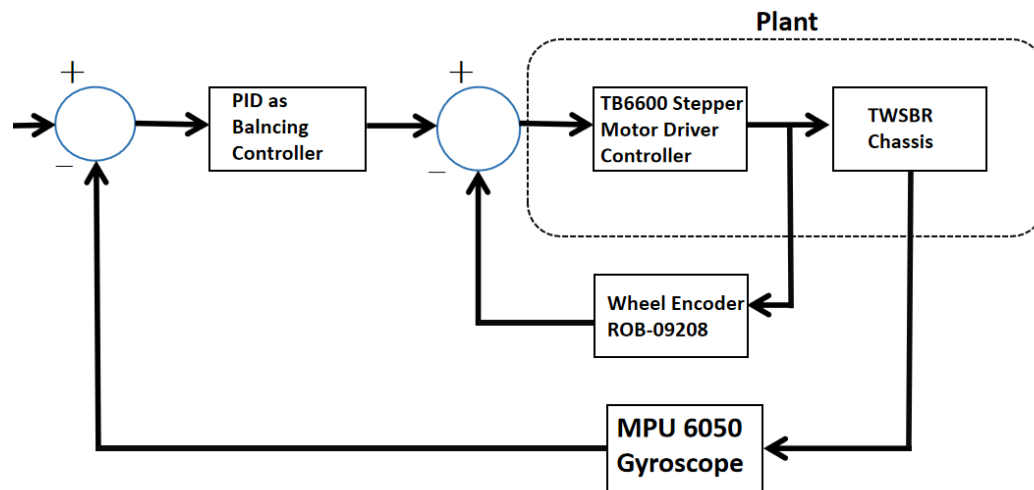


Fig13: Proposed working structure of our two-wheeled self-balancing robot for actual version if manually controlled

## **Components**

### 1.IMU (Inertial Measurement Unit)

**Function:** Measures the tilt angle and angular velocity of the robot.

**Output:** Provides tilt angle feedback to the balancing controller.

### 2.Tilt Angle Feedback

**Function:** Signals the current tilt angle of the robot, which is crucial for balance.

### 3. Controller

**Inner Loop PID:** Controls the motor speed (actuator) to maintain the desired wheel speed.

**Outer Loop PID:** Receives the robot's tilt angle as feedback and adjusts the reference speed for the inner loop PID, ultimately influencing the motor speed to keep the robot balanced.

### 4.Left Wheel and Right Wheel

**Function:** Actuated by the control signals to adjust the robot's balance and movement. Also Since there are two wheels, and each wheel independently contributes 1 DOF through its rotation, the wheels collectively provide 2 degrees of freedom in terms of rotational movement. This rotational freedom allows the robot to maneuver forward, backward, and to some extent, turn by varying the speed and direction of each wheel.

## **4.1.1.b.Software Realization**

The design and development of our two-wheeled self-balancing robot (TWSR) heavily relied on three critical software tools:

- CAD software was integral to the initial stages, allowing us to create a detailed model of the robot's design, including all mounting points for motors, sensors, and electronics. The finalized CAD model, with precise dimensions and specifications, ensured that all physical components would fit together seamlessly, minimizing potential issues during assembly.
- Simulink was then used to simulate the robot's behavior and control system, allowing us to test and refine control algorithms. By setting up a Simulink model representing the robot's dynamics, we could adjust parameters and optimize the control logic based on simulation results, predicting the robot's response to different inputs and conditions.

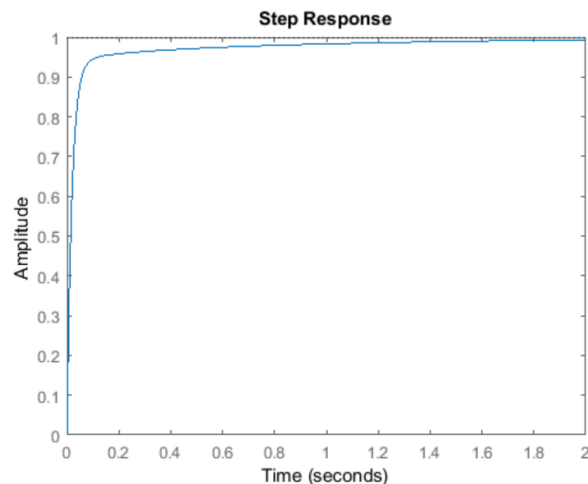
- **Arduino IDE** :The Arduino Uno software played a crucial role in programming the robot's microcontroller, which managed sensor inputs and motor outputs. We developed the control logic, focusing on PID control for balancing, and outlined the basic control flow for the robot's balancing algorithm.

### 4.1.1.c. Calibration

#### A. Step Plot of Amplitude

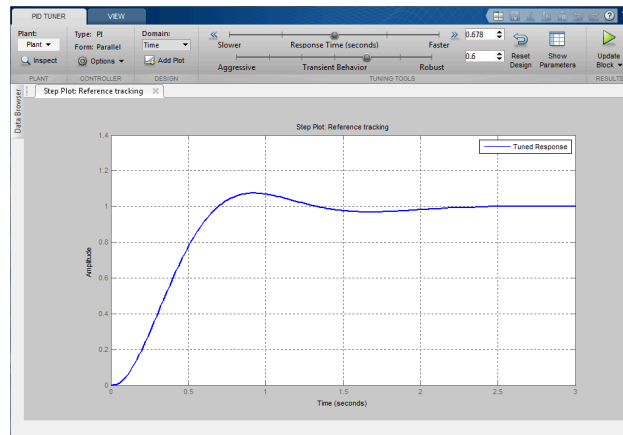
A step plot of amplitude with respect to time for the robot shows how the amplitude of the system's response changes over time when a step input is applied. Here the Tilt Angle Response, Position Response and Velocity Response of the robot is crucial to manage the balance simultaneously. The step plot is essential for evaluating the performance of the control system used in the robot. It helps in:

1. **Tuning Control Parameters:** Adjusting PID controller gains to achieve desired performance (minimal overshoot, quick settling time).
2. **Stability Analysis:** Ensuring the robot can return to a stable state after a disturbance.
3. **System Response Characterization:** Understanding how the robot responds to sudden changes, which is crucial for tasks requiring rapid adjustments.



**Fig 14.A:** Step Plot of Amplitude

## B.PID Tuning



**Fig 14.B:** Amplitude Time Graph

**Step :1.Design Simulink model with 2 PID controllers for wheels & gyro, simulate & analyze response (rise time, overshoot)**

**Step 2:Set initial PID gains and iterate based on simulation response.**

**1. Run Simulation:**

Simulate the model with the initial PID gains.

**2. Analyze Response:**

Observe the system's output response to the reference signal.

Focusing on two key aspects:

- **Rise Time:** Time taken for the output to reach (near) the reference value.
- **Overshoot/Undershoot:** Maximum deviation of the output from the reference value during settling.

**Step 3:.Evaluate Response:**

**Case1 :.Response Too Slow (Large Rise Time):**

Increase  $K_p$  to make the controller more responsive. This will reduce rise time but may also increase overshoot.

**Case 2:Excessive Overshoot/Undershoot:**

Decrease  $K_p$  to reduce the magnitude of the overshoot/undershoot. This will increase rise time.Consider increasing  $K_i$  for better disturbance rejection (if applicable) but be aware it can further slow down the response.If overshoot is the main concern, add a small amount of  $K_d$  for damping. Be cautious as too much  $K_d$  can cause oscillations.



#### Step 4: Refine Gains and Re-simulate:

Based on our evaluation in step 3, adjust the PID gains ( $K_p$ ,  $K_i$ ,  $K_d$ ) and re-run the simulation. If the response meets desired rise time and transient behavior (minimal overshoot/undershoot), we can stop here. Else we go back to step 4 and continue iteratively adjusting the gains and re-simulating until the required performance is achieved.

### C. Feedback Control Loop Operation

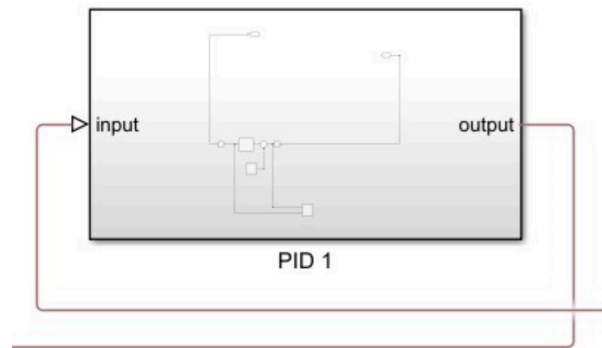


Fig 14.C: Proposed feedback loop for our two-wheeled self-balancing robot in simulink

**1. Tilt Angle Measurement:** The IMU measures the robot's tilt angle and sends this data as feedback to the balancing controller.

#### **2. Balancing Correction:**

1. The balancing controller compares the tilt angle feedback with the center of the balancing constant to determine the error.
2. It then computes the necessary corrections and sends control signals to the wheels to adjust their speeds, counteracting any imbalance.

#### **3. Distance Control:**

1. The distance PD controller calculates the error between the desired position and the current position of the robot.
2. Based on this error, it adjusts the speed of both wheels to move the robot forward or backward to minimize the distance error.

#### **4. Heading Control:**

1. The heading PID controller assesses the heading error, which is the difference between the desired and actual direction of the robot.
2. It sends differential signals to the left and right wheels to correct the heading, ensuring the robot follows the desired path or orientation.

## 5.Wheel Actuation:

1. The control signals from both the balancing and heading controllers adjust the speeds of the left and right wheels.
2. These adjustments help the robot maintain balance, correct its heading, and move towards the desired position.

Learning from Synchronization of Feedback and COM: The robot uses feedback from its sensors (like gyroscopes, accelerometer, encoders) to constantly adjust its balance and position. Synchronizing this feedback with the COM involves ensuring that the robot's control system can react in real-time to maintain stability.

## 5. Application of Proposed Robot in a Societal Context

**Inspection & Maintenance:** Offers a stable platform for inspecting machinery or overhead equipment, replacing ladders with increased maneuverability. Tilting ability provides unique inspection angles.

**Worker Transport:** Enables faster and more efficient movement within factories compared to walking, boosting worker productivity.

**Societal aspect:** Simple modifications to its arm transform it for diverse tasks. In homes, a small groove and tray allow it to hold utensils, aiding with daily chores.

## 6. Dimensions

### Size of Robot proposed for Proof of Concept (Small Version):

Dimension	Measurement (in cm)
Length	13.4
Width	12
Height	25

### Size of Robot proposed as prototype (Actual Version)

Dimension	Measurement (in cm)
Length	13.4
Width	12
Height	43

## 7. Timeline for Robot Making with milestones. (Divided in activities Vs. no. of days)

### **Project Proposal Phases: Two-Wheeled Self-Balancing Robot**

#### **1. Research Phase (15 Days) : Completed**

##### **I. Different Structures and Designs**

- Evaluate different frame materials and configurations
- Select the most promising design based on stability and practicality.

##### **II. Components**

- List essential components
- Ensure all components are available and budgeted.

##### **III. Understanding and Deciding on the Best Applicable Logic or Working of the Bot (Code)**

- Study existing control algorithms, focusing on PID control for balancing.
- Outline the basic control flow and logic for the robot's balancing algorithm.

#### **2. Initial Design Phase (10 Days) : Completed**

##### **I. Designing the CAD Model**

- Ensure the design includes all mounting points for motors, sensors, and other electronics parts.
- Finalize the CAD model with dimensions and specifications ready for fabrication.

#### **3. Submission of the Documentation**

The document was submitted on 30 June 2024.

#### **4. Simulation Phase (10 Days)**

##### **I. Using Simulink**

- Set up a Simulink model representing the robot's dynamics and control system.
- Adjust parameters and refine the control algorithm based on simulation results.

#### **5. Small Version (Proof of Concept) Construction (15 Days)**

##### **I. Making the Base with Cardboard**

- Ensure the base can hold all components securely.
- Ensure that all the joints and movable parts are working properly.

## **II. Calibrating the Gyro accelerometer**

- Connect the gyro accelerometer to the microcontroller.
- Perform calibration tests to ensure accurate angle readings.

## **III. Connecting All Necessary Parts**

- Mount the motors, wheels, microcontroller, and sensors on the cardboard base.
- Ensure all connections are secure and correctly wired.

## **IV. Programming the Microcontroller**

- Develop the control code, implementing the PID algorithm and sensor integration.
- Test basic functionality, such as motor control and sensor readings.

## **V. Debugging**

- Run tests to detect and resolve hardware glitches or software bugs.

## **VI. Changing the PID Values for Effective Balancing**

- Experiment with different PID values and use trial and error or automated tuning methods to optimize performance.

## **6. Discussion and Further Research (10 Days)**

### **I. Research on Additional Features**

- Investigate potential features such as remote control, obstacle avoidance, or self-charging to enhance functionality.

### **II. Finalizing All Aspects of the Bot**

- Review all design choices and ensure all systems are ready for integration and testing.

## **7. Large Version Construction (20 Days)**

### **I. 3D Printing the Base**

- Print the base frame and any other necessary parts using suitable materials.
- Assemble the printed components to form a robust structure.

### **II. Calibrating the Gyroaccelerometer**

- Repeat the calibration process to ensure accuracy with the new hardware setup

### **III. Connecting All Necessary Parts**

- Securely mount the motors, wheels, microcontroller, sensors, and power supply.

### **IV. Programming the Microcontroller**

- Test the code to ensure all functionalities are working as expected.

### **V. Debugging**

- Perform comprehensive testing to identify and fix any software or hardware issues.

### **VI. Changing the PID Values for Effective Balancing**

- Fine-tune the settings based on real-world testing and performance feedback.

## 8. Final Testing, Review, and Changes (10 Days)

### I. Checking and Debugging Any Errors That May Have Arisen

- Run detailed tests to check for any remaining issues or bugs.

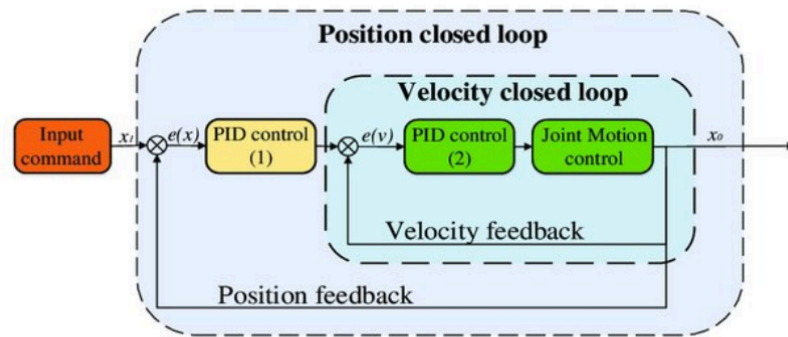
### II. Final Testing

- Conduct stress tests to ensure robustness and reliability under various conditions.

## 9. Implementation of further objectives that is to be provided later

# 8. Proposed Outline (photography) Of TWSBR

## 8.1. Two-Wheeled Balancing Robots: Balancing and Maneuvering



**Fig15:** Block diagram of pid control

A two-wheeled self-balancing robot relies on its two wheels to maintain its upright position. However, it only has four degrees of freedom: one for tilting (angular motion) and the other three for movement on a flat surface (planar motion).

This robot's dynamics are non-linear because the two wheels that drive it are positioned underneath the robot's body. Unlike a cart-and-pendulum system, the actuators (motors) of a two-wheeled self-balancing robot are directly mounted on the pendulum itself, which is essentially the robot's body.

## 8.2. Maneuvering Mechanism

### Possible Solutions

Linear Quadratic Regulator (LQR)	Dual PID Controllers for a Two-Wheeled Inverted Pendulum	Cascaded PID Control for a Two-Wheeled Inverted Pendulum
This technique builds upon state-feedback control by designing a controller that minimizes a cost function penalizing deviations from the desired state and control effort. It offers an optimal control strategy within the confines of the linearized model.	Employ two independent PID controllers, one for each leg (motor) of the robot. Each PID controller receives a separate control signal based on the desired behavior (e.g., maintaining balance) and adjusts the motor power accordingly.	Cascading allows for coordinated control of both wheels. The outer loop ensures both wheels respond appropriately to maintain balance based on the tilt angle. This coordinated response significantly improves the system's ability to handle disturbances and maintain balance.
<p>Modeling Challenges: Linearization often involves simplifying the system dynamics. This can lead to inaccurate control behavior, especially when dealing with larger deviations or complex maneuvers.</p> <p>Tuning Complexity: Extracting a linear model and selecting appropriate gains for state-feedback control or cost function weights for LQR can be challenging.</p>	<p>Limited Coordination: With separate controllers, the system lacks a coordinated response to complex situations. Significant tilt or disturbances might overwhelm one leg's controller, leading to instability.</p> <p>Error Propagation: Errors from one controller can propagate to the other leg, potentially amplifying the issue and causing a balancing failure</p>	The effectiveness of the cascaded system relies heavily on accurate sensor data (e.g., pendulum angle). Sensor noise or biases can lead to inaccurate control signals and potential instability.

Table4: Possible solutions

## Chosen Mechanism

### 8.2.1. Cascaded PID Controller

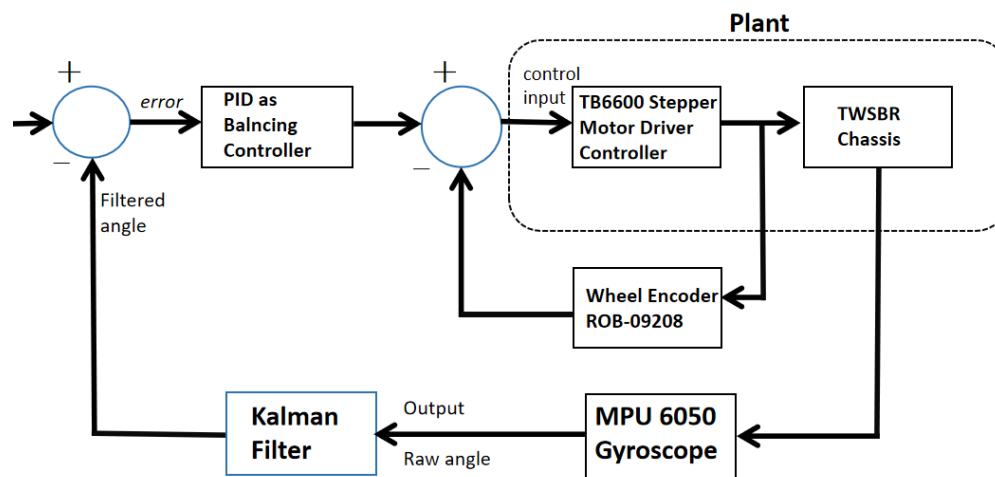


Fig16: Block diagram of Cascaded PID Controller

### 8.2.1.i. Components:

#### 1. Sensors:

- MPU 6050 (gyroscope): Measures the robot's tilt angle (pitch) and angular velocity.
- Wheel encoder ROB-09208: Measure the rpm of the wheels .

#### 2. Controllers:

- Inner Loop PID: Controls the motor speed (actuator) to maintain the desired wheel speed.
- Outer Loop PID: Receives the robot's tilt angle as feedback and adjusts the reference speed for the inner loop PID, ultimately influencing the motor speed to keep the robot balanced.

### 8.2.1.ii.Data Acquisition

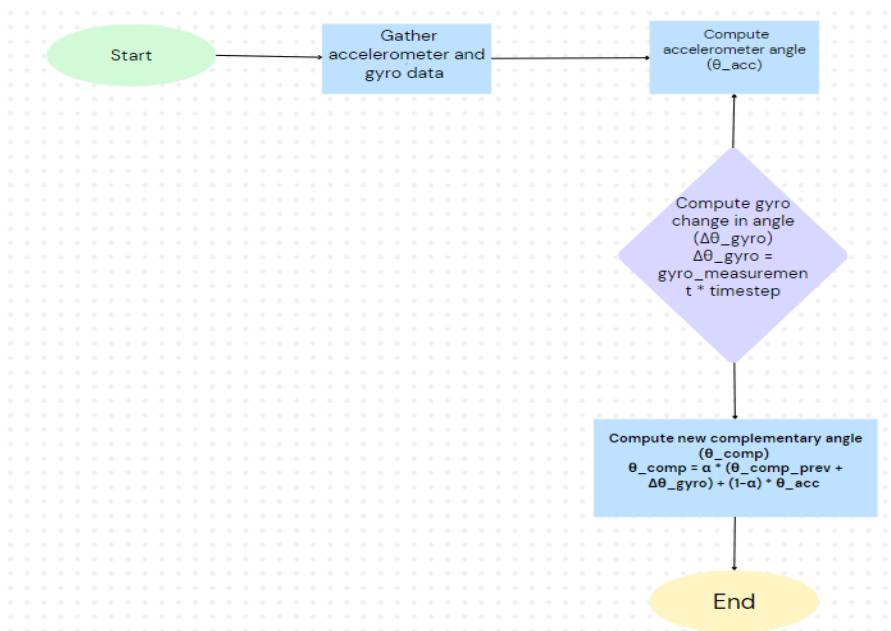


Fig17: Algorithm flowchart

## A.Finding Tilt with an Accelerometer:

An accelerometer senses up/down (z) and forward/backward (x) movement. By comparing these accelerations to the perfect vertical line, the robot calculates its tilt angle. A positive angle means it's leaning forward, while negative indicates backward tilt.

## B.Keeping Track of Tilt with a Gyroscope:

A gyroscope senses its tilting motion by measuring rotation speed around its side axis. This raw data is converted to degrees per second, then multiplied by a tiny time window to estimate the tilt amount. By adding this tilt to the robot's memory of its previous tilt,

## Calibration and reading of MPU 6050 :

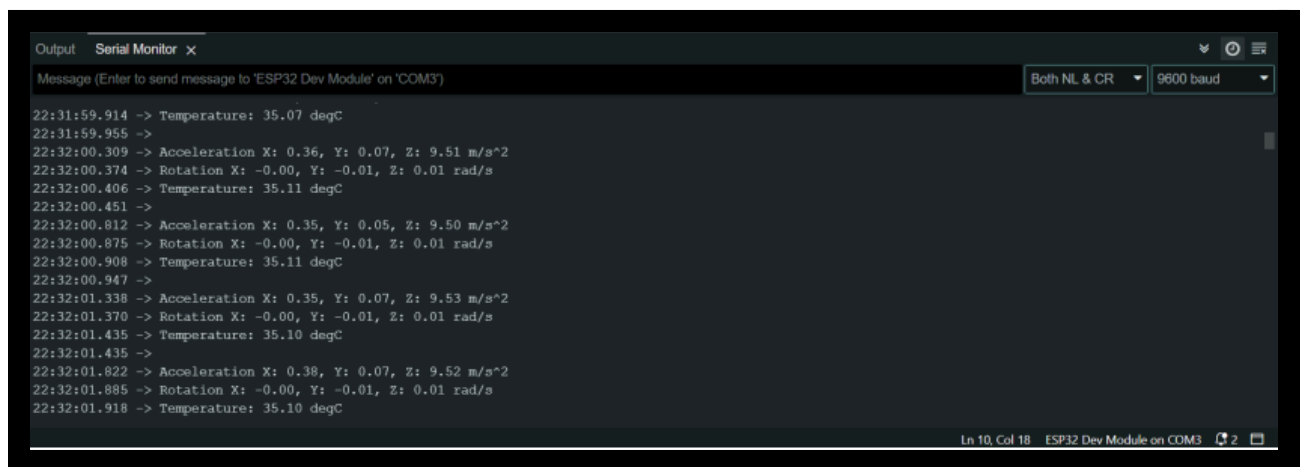


Fig18: MPU 6050 reading Along 6 axis

### 8.2.1.iii.Kalman Filter for Noise Reduction in MPU 6050 Readings

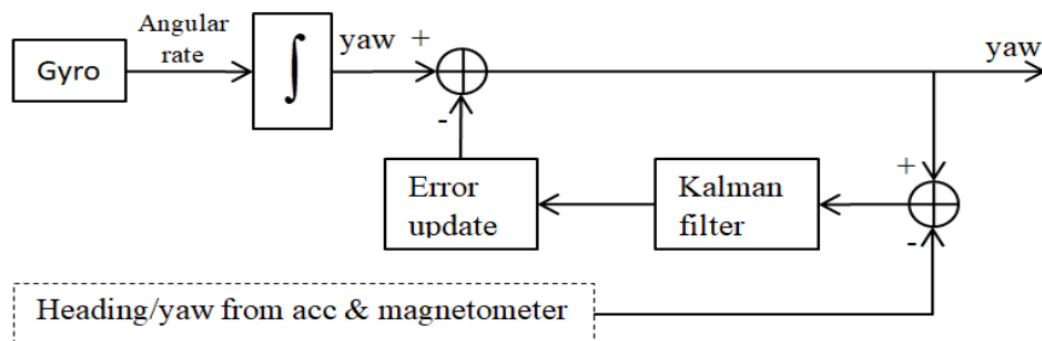


Fig19: Kalman Filter Implementation



An ideal recursive technique for estimating a dynamic system's state from a sequence of imprecise and noisy measurements is the Kalman Filter. There are two stages to its operation:

1. *Prediction*: Estimates the present state using the system model and the prior state as a basis for prediction.
2. *Update*: Applying the current measurement, corrects the anticipated condition.

### Why should Kalman Filter be added:

- **Noise reduction**: Gyroscopes can have drift and noise built in. This is especially true of MEMS gyroscopes, which are frequently used in robotics. By merging gyro data with information from other sensors (such as accelerometers or encoders), a Kalman filter can efficiently reduce noise and increase the precision of angular velocity measurements.
- **State Estimation**: By fusing noisy sensor measurements with a dynamic system model, the Kalman filter offers a method for estimating the real state of the robot—in this example, its angular velocity. This aids in precisely identifying the robot's tilt or orientation, which is necessary for equilibrium.
- **Adaptability to Dynamics**: Self-balancing robots are dynamic systems, meaning that external disturbances or control actions can cause a quick change in angular velocity. The Kalman filter is appropriate for preserving responsiveness and stability because of its capacity to adjust to shifting dynamics.
- **Integration with Multiple Sensors**: Kalman filters are made to efficiently combine data from several sensors. A more reliable estimate of the robot's condition can be achieved in the context of a self-balancing robot by merging gyro data with accelerometer data (for tilt estimation) and possibly encoder data (for wheel position).

In this application, the Kalman filter is preferred over alternative filters such as low-pass or complementary filters because:

- **Optimal Estimation**: Gaussian assumptions concerning noise and measurement errors lead to the best estimation of Kalman filters. They offer the best linear unbiased estimate (BLUE), which is helpful in preserving state estimation accuracy over time.
- **Dynamic System Handling**: Kalman filters take into account a model of the dynamics of the system, in contrast to static filters like low-pass filters, which simply take into account historical observations. This predictive capacity aids in more effectively managing the robot's dynamic behavioral changes.
- **Adaptation to Sensor Characteristics**: Kalman filters provide resilient performance in a variety of operating settings by adjusting their parameters adaptively in response to the environment and the characteristics of the sensors.

In conclusion, a two-wheeled self-balancing robot that uses a Kalman filter for gyroscope readings improves stability, precision, and responsiveness by managing noise well, combining various sensor inputs, and reacting to sudden changes in the system.

## 8.2.1.iv. Understanding Cascaded PID Control

### Inner Fast Feedback PID Control with Wheel Encoder (HC-020K)

#### Components:

1. **Wheel Encoder (HC-020K):** This sensor is mounted on the motor shaft and generates pulses for each revolution of the wheel. The number of pulses per revolution (PPR) is a specific characteristic of the encoder (check HC-020K datasheet for its PPR value).
2. **PID Controller:** This control algorithm calculates the control signal sent to the motor driver based on the error between the desired and actual wheel speed.
3. **Motor Driver:** This electronic circuit amplifies the control signal from the PID controller and provides the necessary power to drive the DC motor.
4. **DC Motor:** This motor drives the robot's wheels.

#### Working Process:

1. **Desired Wheel Speed:** The outer loop (tilt control) determines the desired wheel speed based on the robot's tilt angle and balancing needs. This desired speed is provided as a reference setpoint for the inner loop PID controller.
2. **Actual Wheel Speed:** The wheel encoder generates pulses as the wheel rotates. These pulses are converted into a measurable value representing the actual wheel speed (RPM). This conversion involves:
  - a. Counting the encoder pulses within a specific time interval (e.g., 0.01 seconds).
  - b. Multiplying the pulse count by the encoder's PPR and dividing by the time interval. This provides the revolutions per second (RPS).
  - c. Converting RPS to RPM by multiplying by 60 (seconds per minute).
3. **Error Calculation:** The PID controller calculates the error signal by subtracting the actual wheel speed (RPM) from the desired speed (RPM). This error represents the difference between what the robot should be doing (desired speed) and what it's actually doing (actual speed).
4. **PID Control Action:** The PID controller uses the error signal to compute a control output. This output is a numerical value that determines the direction and intensity of the voltage sent to the motor driver.
  - a. **Proportional (P) Term:** This term is proportional to the current error. A positive error (actual speed lower than desired) results in a positive control output, commanding the motor to spin faster. A negative error leads to a negative control output, slowing down the motor.
  - b. **Integral (I) Term:** This term accumulates the past errors over time. A persistent positive error (motor not reaching desired speed) gradually increases the control output, providing additional corrective action. The integral term helps eliminate steady-state errors.
  - c. **Derivative (D) Term:** This term considers the rate of change of the error. A rapidly increasing error (motor speed quickly deviating from desired) triggers a larger control output to counteract the change more aggressively. The derivative term improves the system's response time.

5. **Motor Control:** The PID controller's output is sent to the motor driver. The driver amplifies this signal and provides the necessary voltage and current to the DC motor.
6. **Feedback Loop:** The actual wheel speed, measured through the encoder, is continuously fed back to the PID controller. This completes the control loop, allowing the PID controller to adjust the motor's behavior based on the latest error information.

#### **Benefits of Inner Loop PID Control:**

- **Accurate Wheel Speed Regulation:** Maintains the desired wheel speed despite disturbances (uneven terrain, friction) by continuously adjusting the motor control signal.
- **Fast Response:** The PID controller reacts quickly to changes in the error signal, ensuring the wheel speed closely follows the reference setpoint.
- **Improved Balance Control:** By precisely controlling wheel speeds, the inner loop contributes significantly to the overall stability and balance of the robot.

#### **Tuning the Inner PID:**

The effectiveness of the inner loop depends on the proper tuning of the PID gains ( $K_p$ ,  $K_i$ ,  $K_d$ ). Here are some general guidelines:

- **Start with low gains:** Begin with conservative values for  $K_p$ ,  $K_i$ , and  $K_d$  to avoid instability.
- **Gradually increase  $K_p$ :** This will make the controller more responsive to errors but can lead to oscillations if set too high.
- **Fine-tune  $K_i$ :** A small  $K_i$  value can help eliminate steady-state errors but a large  $K_i$  can cause sluggish response.
- **Adjust  $K_d$  cautiously:** A small  $K_d$  can improve response time but a large  $K_d$  can make the system overly sensitive to noise.

## **Outer Slow Feedback PID Control with Kalman Filter and MPU-6050**

The outer loop in the cascaded PID control system focuses on maintaining the robot's balance by controlling its tilt angle. Here's a detailed explanation of its working:

#### **Components:**

1. **MPU-6050 Sensor:** This Inertial Measurement Unit (IMU) provides data on the robot's orientation, including its tilt angle. It typically measures acceleration and gyroscope data.
2. **Kalman Filter:** This algorithm is used to improve the accuracy and stability of the tilt angle data obtained from the MPU-6050. It combines sensor measurements with a system model to provide a more reliable estimate of the true state (tilt angle in this case).
3. **PID Controller:** This control algorithm calculates the adjustments needed for the desired wheel speed (reference setpoint for the inner loop) based on the robot's tilt angle.
4. **Inner Loop (Fast Feedback PID):** This loop controls the actual wheel speed based on the reference setpoint provided by the outer loop.

#### **Working Process:**

1. **Tilt Angle Measurement:** The MPU-6050 sensor captures raw acceleration and gyroscope data.
2. **Kalman Filter Processing:** The raw sensor data is fed into the Kalman filter. The filter:
  - a. Uses a system model that describes the relationship between the robot's tilt angle, its accelerations, and the gyroscope readings.

- b. Incorporates the noisy sensor measurements to produce a filtered and more reliable estimate of the robot's tilt angle.
3. **Error Calculation:** The PID controller calculates the error signal by subtracting the desired tilt angle (typically zero for a balanced robot) from the filtered tilt angle provided by the Kalman filter. This error represents the robot's current tilt deviation from its ideal balanced position.
4. **PID Control Action:** Similar to the inner loop, the PID controller uses the error signal to compute a control output. This output translates into an adjusted desired wheel speed (reference setpoint) for the inner loop.
  - a. **Proportional (P) Term:** A positive error (robot tilted forward) results in a decrease in the desired wheel speed (slowing down) to counteract the forward tilt. A negative error (robot tilted backward) increases the desired speed (speeding up) to bring it back upright.
  - b. **Integral (I) Term:** This term accumulates past errors. A persistent tilt (e.g., the robot keeps leaning forward) gradually adjusts the desired speed further, providing a stronger corrective action over time.
  - c. **Derivative (D) Term:** This term considers the rate of change of the tilt angle. A rapidly increasing tilt (robot quickly falling forward) triggers a larger adjustment to the desired speed, aiming to arrest the tilt more aggressively.
5. **Inner Loop Control:** The adjusted desired wheel speed from the outer loop PID becomes the reference setpoint for the inner loop PID control. The inner loop then regulates the actual wheel speed to achieve this new setpoint, ultimately influencing the robot's tilt through its movement.
6. **Feedback Loop:** The filtered tilt angle from the Kalman filter is continuously fed back to the outer loop PID controller. This completes the control loop, allowing the PID to adjust the desired wheel speed based on the latest tilt information.

### Benefits of Outer Loop PID Control with Kalman Filter:

- **Improved Tilt Regulation:** The PID controller actively maintains the robot's desired tilt angle (usually zero for balance).
- **Robustness to Sensor Noise:** The Kalman filter significantly reduces the impact of sensor noise in the MPU-6050 data, leading to more accurate tilt angle estimation and more precise control.
- **Slower Response (Intended):** The outer loop focuses on overall balance and can have a slower response time compared to the inner loop. This allows for smoother and less jittery robot movements.

### Tuning the Outer PID:

Similar to the inner loop, the outer loop PID also requires tuning its gains ( $K_p$ ,  $K_i$ ,  $K_d$ ) for optimal performance. Here are some considerations:

- **Start with conservative gains:** Begin with low values to avoid over-corrections or oscillations.
- **Focus on  $K_p$ :** A higher  $K_p$  will make the controller more responsive to tilt errors but can lead to jerky movements if set too high.
- **Adjust  $K_i$  cautiously:** A small  $K_i$  can help eliminate persistent tilt errors but a large  $K_i$  can make the robot sluggish.
- **Use  $K_d$  sparingly:** The outer loop typically relies less on the derivative term. A small  $K_d$  value might be helpful for faster response but prioritize  $K_p$  and  $K_i$  for overall balance control.

## Understanding Cascaded PID Control

### 1. Outer Loop Sets the Direction:

- The outer loop continuously monitors the robot's tilt and determines the necessary correction based on the error.
- It calculates an adjusted desired wheel speed (reference setpoint) and sends it to the inner loop.
- A positive error (robot tilted forward) typically results in a decrease in the desired wheel speed (slowing down) to counteract the forward tilt.
- A negative error (robot tilted backward) increases the desired speed to bring it back upright.

### 2. Inner Loop Takes Action:

- The inner loop receives the adjusted desired wheel speed from the outer loop.
- It compares this setpoint to the actual wheel speed measured by the encoder.
- Based on this error, the inner loop PID controller regulates the motor power through the motor driver to achieve the desired wheel speed.

### 3. Continuous Feedback:

- The filtered tilt angle from the Kalman filter is continuously fed back to the outer loop PID for ongoing monitoring and adjustment.
- The actual wheel speed, measured by the encoder, is continuously fed back to the inner loop PID for precise speed regulation.

## 8.3.Motivation for Selection of Design Considerations

Our robot blends inspiration from Ascento's jumping agility and GR00T's precise hand control. Ascento's leg-wheel combo inspired us to add legs for obstacle handling. GR00T's hand movements inspired a self-righting system that uses the robot's "hands" (as proposed in our model) to recover from falls, mimicking GR00T's precision. This focus on stability, adaptability, and versatility prepares our robot for diverse competition scenarios.

## 8.4.Material Selection for TWSBR Chassis

For our TWSBR (Two-Wheeled Self-Balancing Robot) chassis, we've opted to use ABS (Acrylonitrile Butadiene Styrene) filament for 3D printing. Here's why:

**Robust and Impact-Resistant:** ABS possesses excellent toughness and can withstand mechanical stress. This property makes it ideal for functional parts in our robot.

**Lightweight Design:** To strike a balance between weight and strength, we'll maintain a 30% infill percentage for the links and shins. This ensures structural integrity while keeping the overall structure lightweight.

**Box Infill:** Since the box won't directly experience impacts, a 15% infill should suffice.

We've already created a small prototype using cardboard to validate our design's balancing and effectiveness.

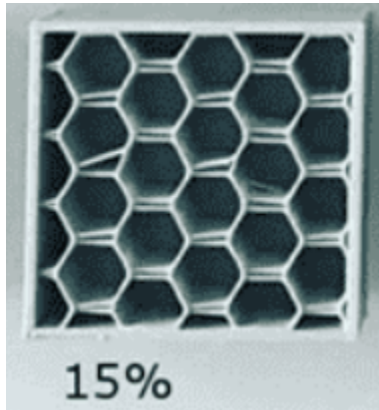


Fig 21: Infill Style for the box

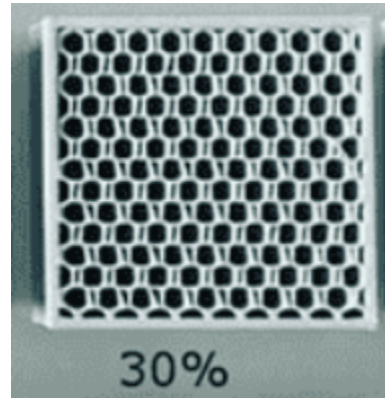


Fig 22: Infill Style for the links and shin

## 8.5.Implementation:

### a) Sensor Integration:

A sensor, such as an accelerometer or gyroscope, measures the robot's tilt angle in real-time. This data is converted into an electrical signal and fed into the microcontroller.

### b) PID Calculation:

The desired tilt angle (setpoint) is compared to the measured tilt angle (actual state) to calculate the error. This error is then used to calculate the proportional, integral, and derivative terms based on predefined PID constants ( $K_p$ ,  $K_i$ ,  $K_d$ ).

Center of gravity is a constant in degree unit that is the balance point of the robot mechanical body. Output of this control is applied to the motor speed.

### c) Motor Control:

The sum of the PID terms is used to generate a control signal that determines the speed and direction of the robot's motors. By adjusting the motor speeds, the robot can tilt forward or backward to influence its center of gravity and regain balance.

### d) Simulink PID Tuner:

- Simulink offers a built-in PID Tuner block specifically designed to simplify the tuning process.
- This block automatically analyzes the robot model and proposes initial PID gains.
- Choosing to use the following tune methods:
  - **Step Response Tuning:** Focuses on the robot's response to a step change in tilt, allowing to fine-tune the gains for settling time and overshoot.

## 8.6. Testing And Planning

### Impact on Step Function Response:

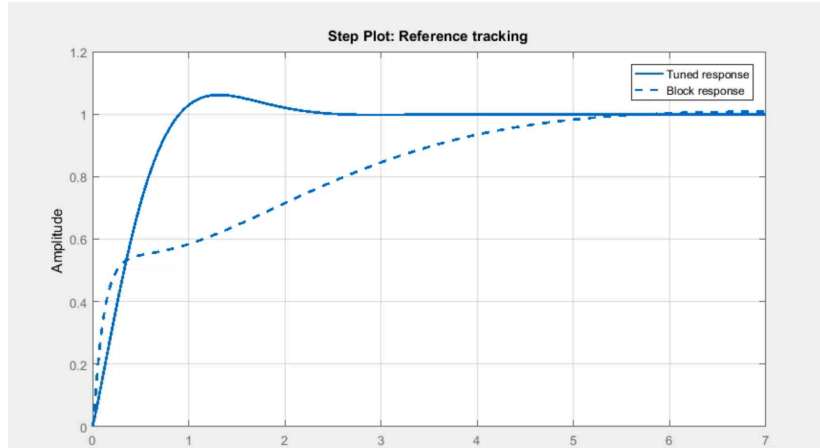


Fig 23.A:Tuned PID in simulink

### 1.External Force:

The presence of an external force adds complexity to the system's response. The impact of the external force depends on its direction, magnitude, and timing relative to the step function input.

**2.Constant Force:** A constant external force acts like a constant offset to the desired setpoint. The PID controller will adjust the output to compensate for

the external force and maintain the desired setpoint.

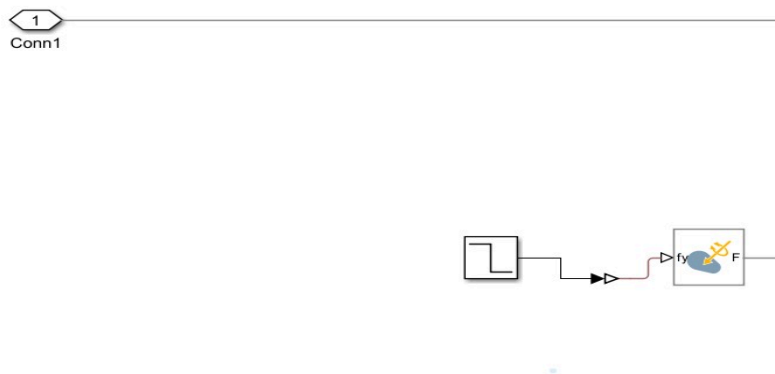
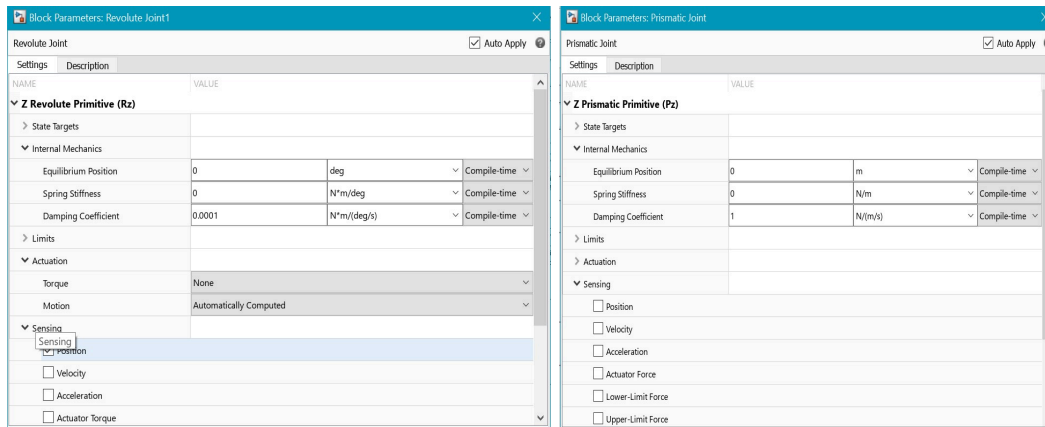


Fig 23.B: External Force applied on top of the body in simulink

## Damping:

- **Rise Time:** Damping slows down the system's response to a step function. However, it also reduces the likelihood of overshoot and helps the system settle at the desired setpoint faster.
- **Overshoot:** Damping significantly reduces overshoot by counteracting the tendency of the system to "overshoot" the setpoint after a change. A well-tuned PID controller with damping should exhibit minimal to no overshoot.
- **Settling Time:** Damping shortens the time it takes for the system's output to reach and remain within a specified range around the desired setpoint.



**Fig24: A: Damping Coefficient in revolute joint in between chassis and wheel.  
B: Figure: Damping Coefficient in revolute joint in between wheels.**

*In our initial attempt to achieve self-balancing, we experimented with a 2 control system using two independent PID loops for each wheel. While this approach seemed intuitive, it ultimately fell short in for robotic stability.*

*There was inherent complexity in managing four independent controllers. Tuning each PID loop's gains proved to be a formidable task, and there was a constant risk of the loops sending conflicting signals to the motors. This lack of coordination between the wheels made achieving a balanced posture nearly impossible.*



## Simulation of PID against external damping force

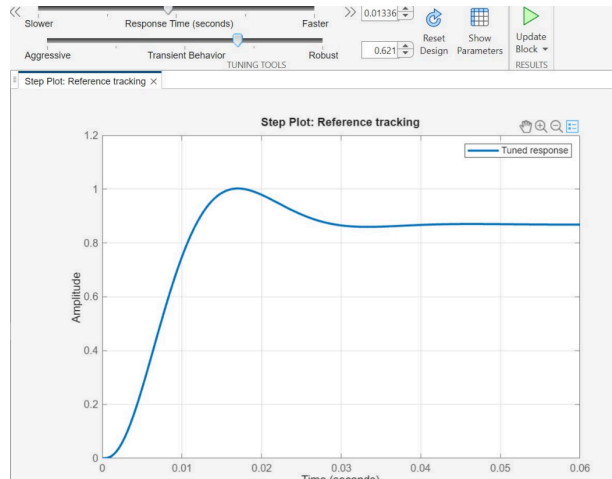


Fig 25.A: PID 1 for left wheel

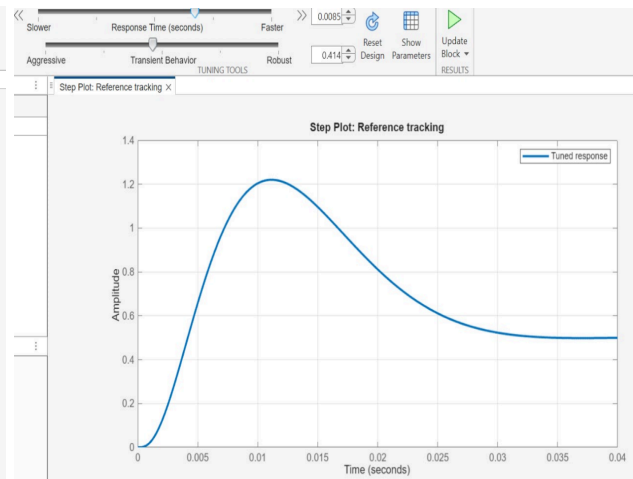


Fig 25.B: PID 2 for right wheel

## 8.7.Observations

**The simulation revealed the following key observations:**

1. **Non-Linear Input Effect:** The input commands did not have a uniform effect on both wheels. The tuned PID controller for the left wheel achieved a short rise time and an optimal overshoot value. However, the PID controller for the right wheel exhibited a faster rise time but with a significantly higher overshoot value.
2. **Separate PID Tuning:** Tuning a PID controller involves adjusting its proportional (P), integral (I), and derivative (D) gains. Tuning each wheel's PID controller independently likely resulted in these differing performance characteristics.

## 8.7.Inference

### 8.7.1.Design Inference

#### **Capabilities and Advantages of Two-Wheeled Self-Balancing Robot**

1. *Simplified Control System:*
  - Efficient implementation using well-studied PID controllers.
2. *Maneuverability:*
  - Excellent agility with the ability to pivot on the spot.

3. *Compact Design:*

- Minimizes footprint, ideal for smaller applications and constrained spaces.

4. *Efficiency:*

- Fewer mechanical components reduce maintenance and energy consumption.

## 8.7.2. Control System Inference

### Introduction to Neural Network in Actual Version of Self Balancing Robot Used For PID Tuning

#### 2. Neural Network Design:

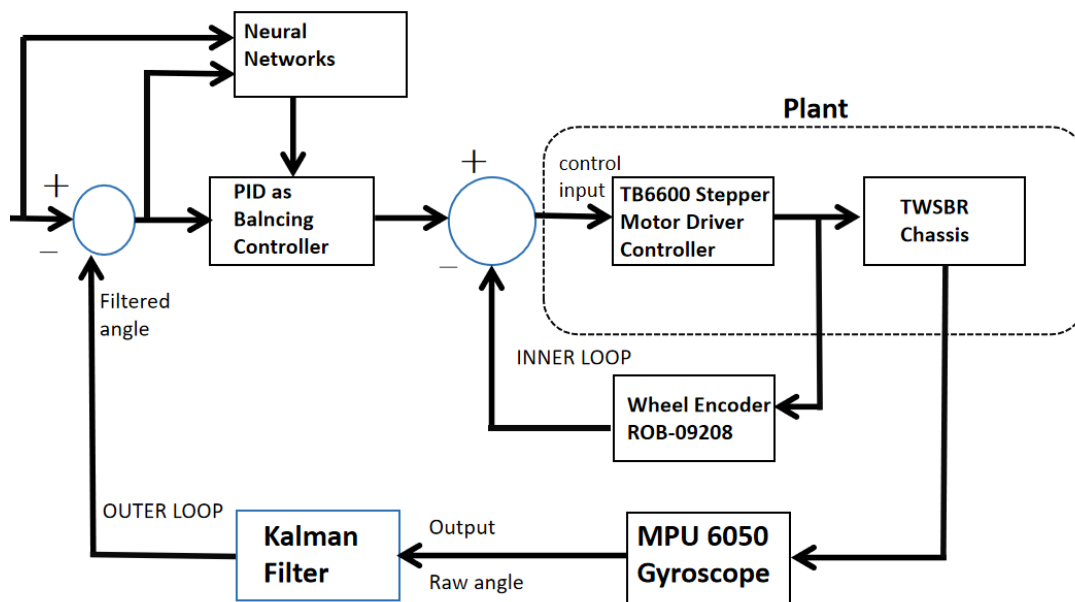
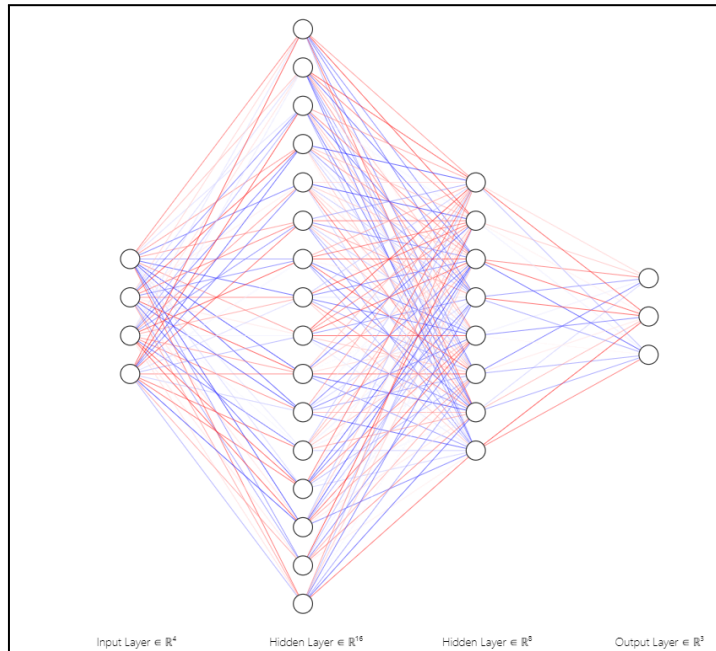


Fig 26: Multilayer Perceptron (MLP) with One Hidden Layer



**Fig 27: Impression of Architecture of Neural Network to be Applied**

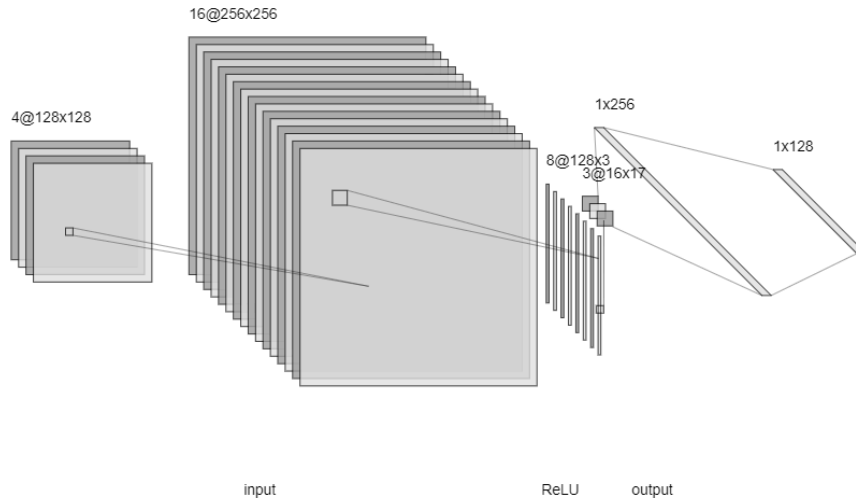
- Input Layer:** The number of neurons in the input layer will depend on the specific sensor data to be tuned for the PID. Number of input data includes:
  - Current tilt angle
  - Angular velocity (rate of change of tilt angle)
  - Wheel speeds (or motor torques)
  - Additional sensor data relevant to balance control (e.g., gyroscope readings) depending on the robot's design.
- Hidden Layer:** One hidden layer with a moderate number of neurons ( 10-20) is a good starting point. This layer performs the main information processing and feature extraction from the input data.
- Output Layer:** The number of neurons in the output layer will correspond to the number of PID gain adjustments we want the network to predict. This typically includes:
  - $\Delta K_p$  for the outer PID loop
  - $\Delta K_i$  for the outer PID loop
  - $\Delta K_d$  for the outer PID loop

## Neural Network Architecture for PID Tuning in a Balancing Robot

**Depth: 4**

- Input Layer:** The layer has an input size of 4.
- Hidden Layer 1:** This layer performs the initial processing of the input data. Complexity level of the robot will be decided during tuning of PID to set the number of neurons. Two situations may arise:

- Lower Complexity: 16 neurons
  - Higher Complexity: 32 neurons
- **Hidden Layer 2:** An additional hidden layer can further enhance the network's ability to learn complex relationships. The number of neurons can be half the size of the first hidden layer (e.g., 8 neurons for higher complexity).
- **Output Layer:** This layer has 3 neurons, each corresponding to one of the PID gains ( $K_p$ ,  $K_i$ ,  $K_d$ ). It uses a linear activation function as the PID gains are continuous values.



**Fig 28:** Depth of the Neural Network using ReLU

#### Activation Functions:

- Using ReLU, a non-linear activation function in the hidden layer, such as the rectified linear unit sigmoid function. This allows the network to learn complex relationships between the inputs and outputs.

#### Training:

Train the network using supervised learning with backpropagation. The training data will consist of paired input-output examples.

- Inputs will be the recorded sensor data from various robot operating conditions.
- Outputs will be the corresponding optimal PID gain adjustments achieved through manual tuning or other methods for those specific operating conditions.

### 3. Training the Neural Network:

- Collect data by operating the robot in various conditions (different speeds, uneven terrain, etc.). During this data collection phase:
  - Record sensor data (inputs) and the corresponding optimal PID gains (outputs) achieved through manual tuning or other methods.

- Train the neural network using the collected data. The network learns the relationship between robot behavior (inputs) and the best PID gains (outputs) for maintaining balance.

#### 4. Gain Adjustment:

- During robot operation, the neural network continuously receives real-time sensor data (tilt angle, angular velocity, etc.).
- Based on this input, the network predicts the optimal adjustments for the PID gains ( $\Delta K_p$ ,  $\Delta K_i$ ,  $\Delta K_d$ ) for both inner and outer loops.
- These adjustments are added to the pre-defined base values of the PID gains, resulting in the final control signals sent to the motors.

#### Advantages of this Architecture:

- It is relatively simple to implement and train.
- Can handle complex relationships between robot behavior and optimal PID gains.
- Computationally efficient compared to more complex network architectures.

## 8.8. Energy Optimisation

- **Design:** The robot has a lightweight 3d printed frame with Abs materials. Its arm structure can be used to lock onto the corners and maintain balance while sleep mode is used.
- **Actuators:** The wheels which are controlled by pid. and the arms and legs that are controlled by servo motors. arms and legs can move to adjust the bot to the most stable position and then we can put the bot in sleep mode.
- **Software Optimization:** During idle periods, the robot enters a low-power standby mode. Dynamic voltage scaling adjusts component voltage based on workload.