

REPORT ON MINOR PROJECT

ELD800

SELF BALANCING BI-WHEEL ROBOT

Submitted by:

Anvisha Chaturvedi
2024EEA2011

Under the guidance of

Prof. SUBASHISH DATTA

In partial fulfillment of the requirements for the award of the degree of
Master of Technology



Department of Electrical Engineering
INDIAN INSTITUTE OF TECHNOLOGY DELHI

Contents

INTRODUCTION	2
OBJECTIVE	3
LITERATURE REVIEW	4
HARDWARE DESIGN	5
MATLAB SIMULATION	21
PID TUNING	21
ARDUINO CODE	22
CHALLENGES	25
CONCLUSION	26

INTRODUCTION

A self-balancing robot is a bi-wheeled mobile robot, modeled as an inverted pendulum on wheels. It represents a dynamic, unstable system that attempts to maintain an upright position. Such systems find practical applications in Segways, autonomous delivery bots, and assistive robotics. The mechanical structure of the self-balancing robot typically comprises two wheels driven by DC motors, allowing the robot to move forward or backward. This type of robot is inherently unstable because its center of mass lies above the pivot point (the wheels). Without control, even a small disturbance can cause it to topple. This makes it an ideal platform for studying and implementing real-time control algorithms and sensor fusion techniques.

It's stability is maintained using feedback from an Inertial Measurement Unit (IMU) sensor consisting of a gyroscope and accelerometer which provides real-time measurements of the robot's tilt angle, angular velocity, which are essential for estimating the robot's state. The data so acquired is processed using techniques such as complementary filters or Kalman filters to reduce noise and improve the accuracy of the estimated orientation. The filtered information is then fed into a control algorithm like a PID controller, Linear Quadratic Regulator (LQR), or Model Predictive Controller (MPC) which determines the appropriate motor commands required to restore balance.

This project leverages the self-balancing robot as a case study to implement and test advanced control strategies.

OBJECTIVE

The primary objective of the project, self-balancing robot, is to design, develop, and implement a bi-wheeled robot capable of maintaining dynamic stability around its vertical upright position, particularly for small angular deviations which is similar to stabilizing an inverted pendulum, a classical control problem.

The robot aims to achieve balance using a Proportional-Integral-Derivative (PID) control which is suitable for systems operating near a linearized equilibrium point. By tuning the PID gains appropriately, the controller generates motor commands that counteract deviations from the upright position, hence restoring balance. The linearization of the system's nonlinear dynamics around the upright position simplifies the control design while still delivering effective performance for small tilt angles.

To achieve this, the robot must continuously estimate its tilt angle and angular velocity using data from an Inertial Measurement Unit (IMU), which includes accelerometers and gyroscopes. Accurate and real-time orientation estimation is critical, necessitating the implementation of Kalman filtering or complementary filtering, to reduce sensor noise and drift.

The controller operates in real-time executing rapid and precise corrections to motor speed and direction to counteract external disturbances which involves integration of feedback control algorithms, sensor processing, and motor actuation.

In addition to achieving balance, the project seeks to develop an extensible platform that can serve as a basis for further research in autonomous navigation, adaptive control, and machine learning in robotics. The robot also serves as a practical implementation of theoretical concepts in control systems, dynamics, signal processing, and embedded systems engineering.

LITERATURE REVIEW

Stabilization Methods for Biwheel Self-Balancing Robots

The stabilization of a self-balancing robot has evolved from simple PID-based control to advanced optimal control techniques. Each method presents trade-offs between implementation complexity, robustness, and computational demands. The primary methodologies adopted and their comparative advantages are:

1. **Classical PID and Cascade Control:** PID controllers are used due to their simplicity and ease of implementation. These systems often incorporate techniques like complementary filtering to combine accelerometer and gyroscope data. Though effective for small disturbances, they may struggle with model uncertainties and external disturbances [1].
2. **Optimal Control Approaches:** Linear Quadratic Regulator (LQR) control utilizes state-space modeling which provides optimal state feedback by minimizing a quadratic cost function. It requires careful tuning of the weighting matrices Q and R [2].
3. **Fuzzy Logic and Hybrid Fuzzy Controllers:** Fuzzy logic controllers are employed to handle nonlinearities inherent in the system. Variants include pure fuzzy logic control, fuzzy-PID hybrids etc. that adapt control parameters in real time and enhance adaptability but require expertise to design effective fuzzy rule sets [3].
4. **Nonlinear Control Techniques:** To deal with nonlinearities and ensure robustness, methods like Sliding Mode Control (SMC) and backstepping are used. These approaches improve performance under large disturbances but are computationally intensive [4].
5. **Reinforcement Learning:** RL techniques, such as Q-learning, deep Q-networks, or actor-critic methods, learn optimal control policies through interaction with the environment. [5] applied RL for balance control of two-wheeled robots. More recently, hybrid control structures combining RL with PID or LQR aim to offer adaptive and robust control, though these approaches require significant computational resources and careful training.

The choice of PID control for this system is motivated by its low computational complexity, which aligns well with the embedded hardware constraints of the system. Moreover, it allows for intuitive tuning and straightforward implementation. The control gains are tuned in order to achieve fast rise time, minimal overshoot, and acceptable steady-state error.

HARDWARE DESIGN

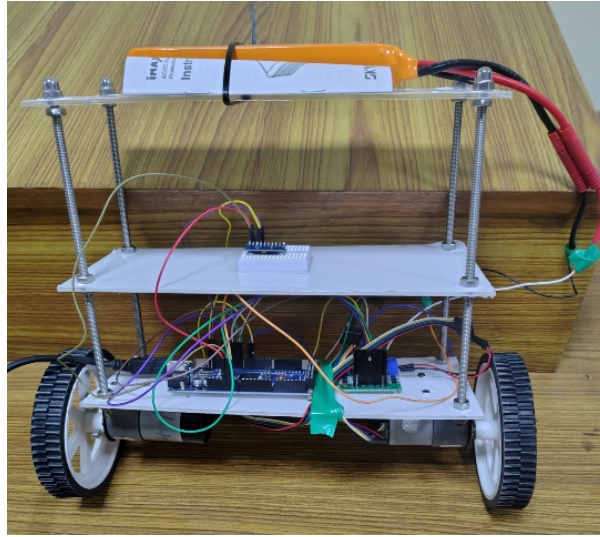


Figure 1: Hardware Setup

1. Chassis:

- (a) The chassis is constructed using three layers of acrylic plates acting as the mounting surfaces for all components.
 - Bottom Plate: This plate is directly above the motors and wheels. It mounts the motor driver and the Arduino board.
 - Middle Plate: This plate holds the IMU sensor.
 - Top Plate: This plate holds the battery.
- (b) Four long, metallic threaded rods are used to connect and provide structural support between the different plates.
- (c) Two 10cm x 2 cm wheels are used.

- 2. A Lithium Polymer (LiPo) battery with 3 cells 11.1V is used to power the system.
- 3. Two 24V DC motors are located at the bottom, directly driving the wheels. The motors are attached to the bottom plate, ensuring stable power transmission to the wheels.

4. Controllers and Sensors:

- (a) **Main Microcontroller Board:** The Arduino Mega 2560 board is the central processing unit. Its ATmega2560 microcontroller will run the main program loop. It reads data from the MPU-6500, executes the PID control algorithm based on the IMU data and generates PWM signals and direction signals for the L298N motor driver.
- (b) **Motor Driver Module:** The L298N is a dual H-bridge motor driver, meaning it can control two DC motors independently. The Arduino Mega's digital output pins connect to the L298N's IN1-IN4 pins (two for each motor) to

control direction and enable pins for PWM speed control. The L298N's OUT1-OUT4 terminals connect directly to the positive and negative terminals of the two 24V DC motors.

(c) **Inertial Measurement Unit (IMU):** MPU 6500 sensor is fundamental for self-balancing. It combines:

- 3-axis Accelerometer: Measures linear acceleration, used to determine the robot's static tilt angle relative to gravity.
- 3-axis Gyroscope: Measures angular velocity (rate of rotation), used to determine how fast the robot is tilting.

The MPU-6500 communicates with the Arduino Mega. The raw data from the accelerometer and gyroscope is processed (using a Kalman Filter) on the Arduino which is input to the PID controller. The MPU-6500 is mounted rigidly on a small white breadboard on the middle plate, connected to the microcontroller with various jumper wires.

HARDWARE CONNECTIONS

The hardware components are then connected using jumper wires as per the following connections:

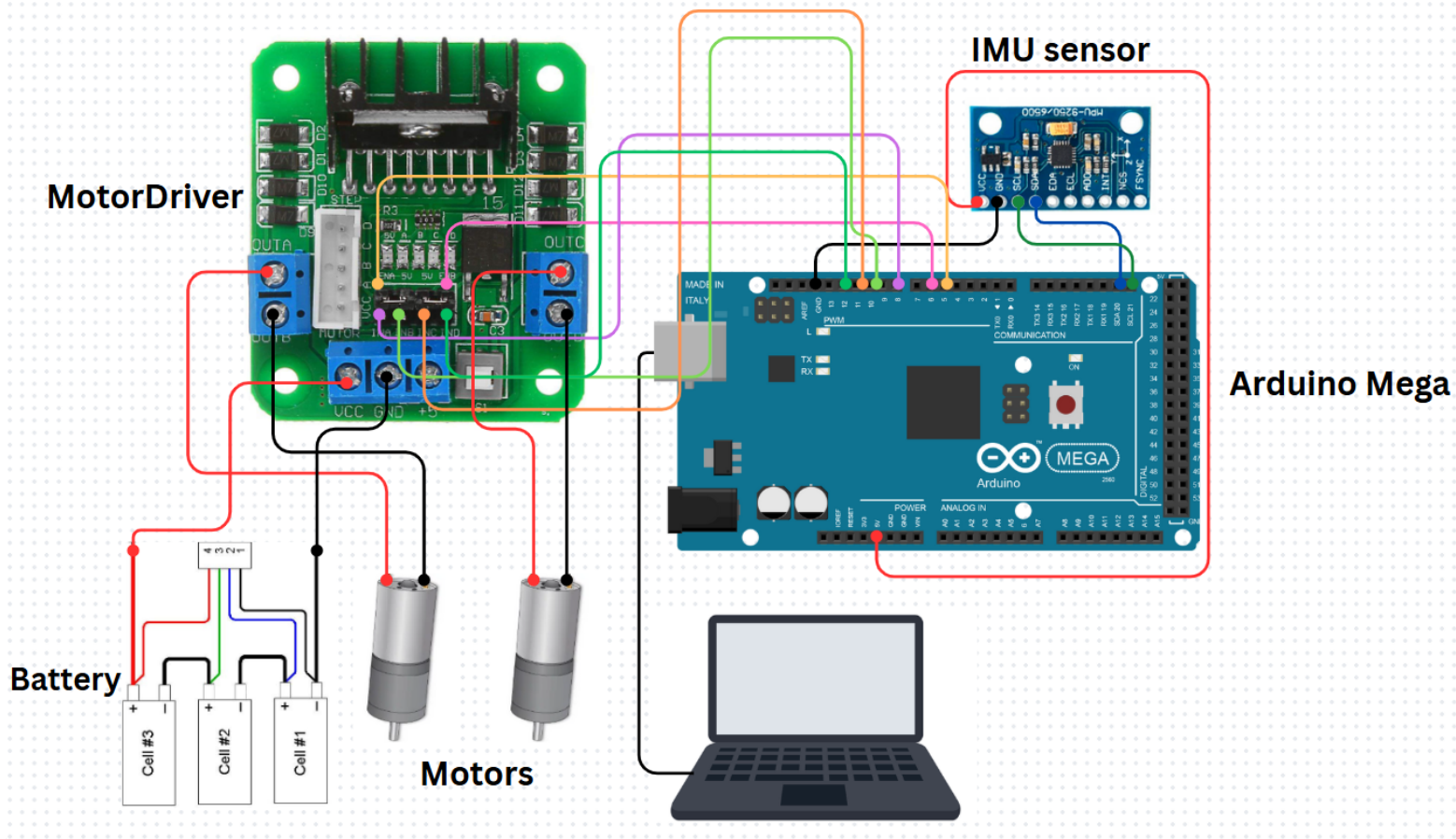


Figure 2: Hardware Connections

1. Power Distribution:

Component	Power Source
Arduino Mega	Laptop
L298N Motor VCC	from LiPo
MPU6500	Arduino Mega (via VCC pin)

2. MPU6500 to Arduino Mega:

MPU6500 Pin	Arduino Mega Pin
VCC	5V
GND	GND
SDA	SDA
SCL	SCL

3. L298N to Arduino Mega (Motor Control):

L298N Pin	Arduino Mega Pin	Description
IN1	Pin 8	Motor A direction 1
IN2	Pin 9	Motor A direction 2
ENA	Pin 5 (PWM)	Motor A speed
IN3	Pin 10	Motor B direction 1
IN4	Pin 11	Motor B direction 2
ENB	Pin 6 (PWM)	Motor B speed
GND	Common ground with battery and Arduino	
VCC	from LiPo Battery	

4. Motors to L298N:

Motor	L298N Output
Motor A	OUT1, OUT2
Motor B	OUT3, OUT4

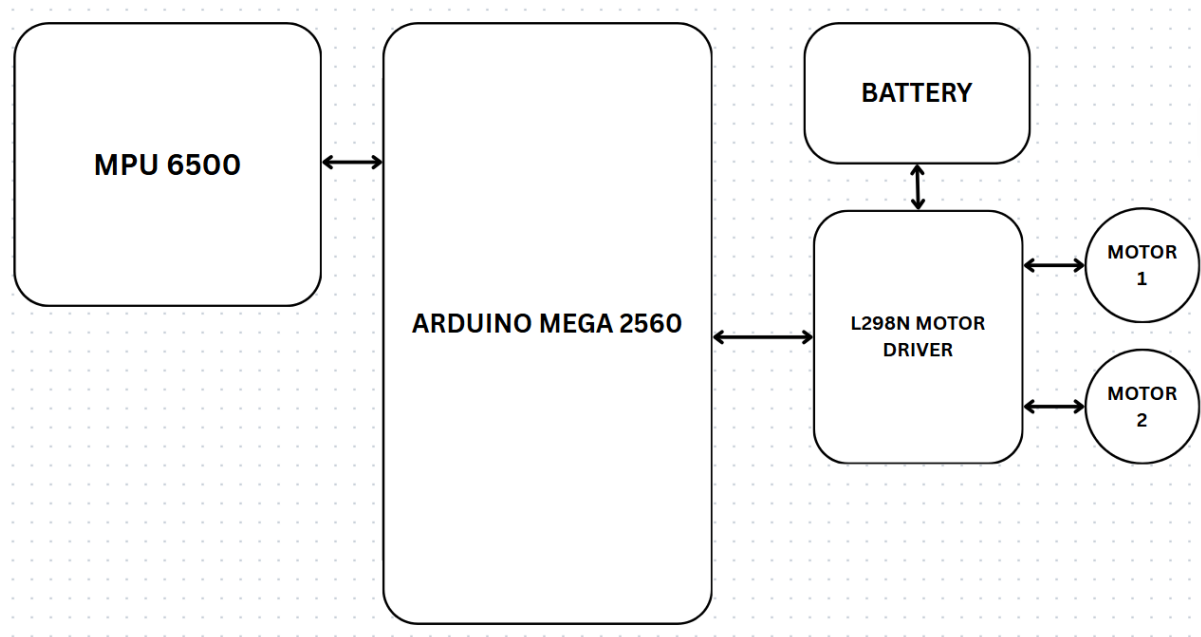


Figure 3: Hardware Connections Block Diagram

SYSTEM MODELING

To mathematically model the system, the principles of classical mechanics are used. The system is treated as an inverted pendulum on wheels [9, 18, 19].

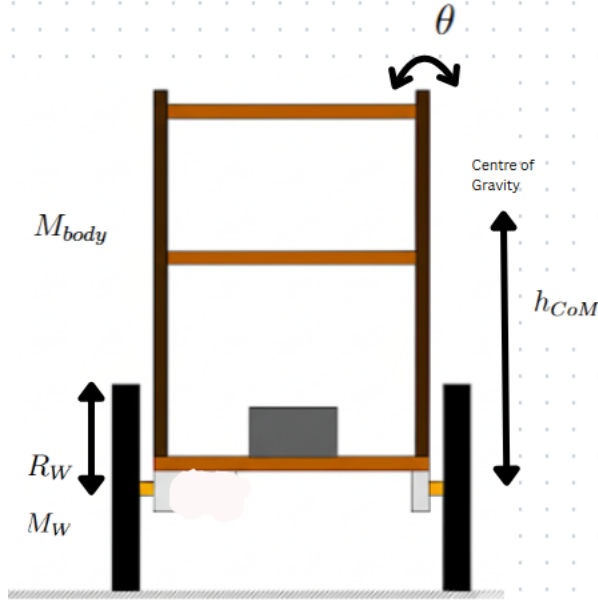


Figure 4: Robot System

Assumptions and Simplifications:

- All components (chassis plates, motors, wheels, battery) are considered rigid bodies.
- Where appropriate, components are modeled as point masses.
- The wheels are assumed to roll without slipping on the ground.
- Rolling Friction between the wheels and the ground and air resistance are assumed to be negligible.
- Ideal DC motors with back EMF and torque constant relationships.

Component Parameters:

Let's define the relevant parameters for each component and part.

1. **Chassis:** Consists of three main horizontal plates and four vertical standoffs.
 - Mass of Top Plate: M_{TP}
 - Mass of Middle Plate: M_{MP}
 - Mass of Bottom Plate: M_{BP}
 - Mass of each Standoff: M_{ST}
 - Total Mass of Standoffs: $4 \times M_{ST}$
 - Moment of Inertia of Top Plate (about its center): I_{TP} (e.g., if rectangular, $I_{TP} = \frac{1}{12}M_{TP}(L_{TP}^2 + W_{TP}^2)$ where L_{TP} and W_{TP} are length and width).

-
- Moment of Inertia of Bottom Plate (about its center): I_{BP} (similarly defined based on geometry).
 - Moment of Inertia of Middle Plate (about its center): I_{MP} (similarly defined based on geometry).
 - Moment of Inertia of each Standoff (about its longitudinal axis): I_{ST} (negligible for balancing dynamics).
 - Length of each Standoff: H_{ST} (vertical distance between plates).
2. **Battery:** The power source, typically mounted on the top plate.
- Mass of Battery: M_B
 - Moment of Inertia of Battery (about its center): I_B (e.g., if cuboid, $I_B = \frac{1}{12}M_B(L_B^2 + W_B^2)$).
 - Center of Mass (CoM) relative to Top Plate: Assume its CoM is at a height $h_{B,TP}$ from the top plate.
3. **Main Control Board (Arduino) and IMU:** Arduino is mounted on the bottom plate and IMU on the middle plate.
- Mass of Arduino and IMU: M_A
 - Center of Mass (CoM) relative to Bottom Plate: Assume its CoM is at a height $h_{A,BP}$ from the bottom plate.
4. **Motors:** Two DC motors, one for each wheel, mounted below the bottom plate.
- Mass of Motor: M_M
 - Moment of Inertia of Motor Rotor (about its rotation axis): J_M (this is critical for motor dynamics).
 - Moment of Inertia of Motor Casing (about its center): $I_{M,casing}$ (usually negligible for balancing).
 - Torque Constant: k_t (motor torque $\tau_M = k_t I_A$, where I_A is armature current).
 - Back EMF Constant: k_e (back EMF $V_b = k_e \omega_M$, where ω_M is motor angular velocity). Assume $k_t = k_e$ in SI units.
 - Motor Terminal Resistance: R_M
 - Motor Inductance: L_M (often neglected for faster dynamics than electrical time constant).
 - Motor Viscous Friction Coefficient: b_M (motor internal friction $\tau_{friction} = b_M \omega_M$).
5. **Wheels:** Two identical wheels, one attached to each motor shaft.
- Mass of Wheel: M_W
 - Radius of Wheel: R_W
-

-
- Moment of Inertia of Wheel (about its axle): J_W (e.g., for a solid cylinder, $J_W = \frac{1}{2}M_W R_W^2$).

6. Overall System Parameters:

- Distance between wheel axles: W_{track} (affects yaw dynamics, but not primary pitch balancing).
- Center of Mass (CoM) of the entire robot: h_{CoM} (vertical height from the wheel axle line when the robot is perfectly upright). This is a crucial parameter.
- Total Mass of the Robot (excluding wheels and motors): $M_{body} = M_{TP} + M_{BP} + M_{MP} + 4M_{ST} + M_B + M_A$.
- Total Mass of the Robot: $M_{total} = M_{body} + 2M_M + 2M_W$.
- Moment of Inertia of the robot body (excluding wheels and motor rotors) about its CoM: I_{body} (this needs to be calculated using the parallel axis theorem for all components relative to the overall CoM).

Coordinate System and Generalized Coordinates:

Let's define a coordinate system:

- **Origin:** On the ground, directly below the initial CoM of the robot.
- **x-axis:** Horizontal, in the direction of forward motion.
- **y-axis:** Horizontal, perpendicular to the direction of motion.
- **z-axis:** Vertical, pointing upwards.

Generalized Coordinates:

1. **Horizontal Position of the Wheel Axle:** x (measured from a fixed origin on the ground).
2. **Angle of the Robot Body with respect to the Vertical:** θ (pitch angle, measured from the z-axis. $\theta = 0$ when upright, $\theta > 0$ when leaning forward).
3. **Angle of Wheel Rotation:** ϕ (angle of wheel rotation relative to its fixed axle, measured from a reference point. $\phi > 0$ for forward rotation). Since there are two wheels, we can assume ϕ_L and ϕ_R for left and right, or an average ϕ for straight motion. For 2D balancing, we can assume $\phi_L = \phi_R = \phi$.

Deriving the Equations of Motion:

We can use either Newton-Euler formulation or the Lagrangian approach. The mathematical modeling of the self-balancing robot, predominantly employs the Lagrangian approach due to its systematic nature for complex systems. This method defines a Lagrangian as the difference between kinetic and potential energies, utilizing Euler-Lagrange equations to derive the motion, especially beneficial for handling constraints like the no-slip condition. While the Newtonian approach (based on Newton's laws of motion and free-body diagrams) is an alternative, but often not the primary choice for modeling complex multi-body systems with constraints like this system's because it necessitates

detailed consideration of all individual forces and torques, often leading to a more complex and numerous set of simultaneous equations.

Lagrangian Approach: The Lagrangian L is defined as $L = T - V$, where T is the total kinetic energy and V is the total potential energy of the system. It calculates the system's total kinetic and potential energies and applies the Euler-Lagrange equations to derive the nonlinear dynamic equations of motion [8] [9]. The Euler-Lagrange equations are given by:

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q}_i} \right) - \frac{\partial L}{\partial q_i} = Q_i$$

Where q_i are the generalized coordinates, $\dot{q}_i$ are their time derivatives, and Q_i are the generalized non-conservative forces (e.g., motor torque, friction).

1. Kinetic Energy (T):

(a) Kinetic Energy of the Robot Body (T_{body}): The robot body (chassis, battery, arduino, motor casings) rotates and translates.

- Translation: $T_{body,trans} = \frac{1}{2} M_{body} v_{CoM,x}^2$
- $v_{CoM,x}$ is the horizontal velocity of the body's CoM.
- The horizontal position of the body's CoM is $x_{CoM} = x - h_{CoM} \sin(\theta)$.
- So, $v_{CoM,x} = \dot{x} - h_{CoM} \cos(\theta) \dot{\theta}$.
- $T_{body,trans} = \frac{1}{2} M_{body} (\dot{x} - h_{CoM} \cos(\theta) \dot{\theta})^2$
- Rotation: $T_{body,rot} = \frac{1}{2} I_{body} \dot{\theta}^2$
- $T_{body} = \frac{1}{2} M_{body} (\dot{x} - h_{CoM} \cos(\theta) \dot{\theta})^2 + \frac{1}{2} I_{body} \dot{\theta}^2$

(b) Kinetic Energy of the Wheels (T_W): Each wheel translates with the body and rotates. We have two wheels.

- Translation: $T_{W,trans} = 2 \times \frac{1}{2} M_W \dot{x}^2 = M_W \dot{x}^2$ (assuming the wheel axle moves with velocity $\dot{x}$).
- Rotation: $T_{W,rot} = 2 \times \frac{1}{2} J_W \dot{\phi}^2 = J_W \dot{\phi}^2$ (angular velocity of wheel ϕ).
- $T_W = M_W \dot{x}^2 + J_W \dot{\phi}^2$

(c) **Kinetic Energy of Motor Rotors (T_M):** The motor rotors also rotate with the wheels.

- $T_M = 2 \times \frac{1}{2} J_M \dot{\phi}^2 = J_M \dot{\phi}^2$

(d) **Total Kinetic Energy:** $T = T_{body} + T_W + T_M$

$$T = \frac{1}{2} M_{body} (\dot{x} - h_{CoM} \cos(\theta) \dot{\theta})^2 + \frac{1}{2} I_{body} \dot{\theta}^2 + M_W \dot{x}^2 + J_W \dot{\phi}^2 + J_M \dot{\phi}^2$$

$$T = \frac{1}{2} M_{body} (\dot{x} - h_{CoM} \cos(\theta) \dot{\theta})^2 + \frac{1}{2} I_{body} \dot{\theta}^2 + M_W \dot{x}^2 + (J_W + J_M) \dot{\phi}^2$$

2. Potential Energy (V):

- The potential energy is primarily due to the height of the body's CoM.
- Reference: Ground level is $V = 0$.
- Height of the body's CoM: $z_{CoM} = R_W + h_{CoM} \cos(\theta)$.
- $V = M_{body}g(R_W + h_{CoM} \cos(\theta))$

3. Lagrangian (L):

$$L = T - V$$

$$L = \frac{1}{2}M_{body}(\dot{x} - h_{CoM} \cos(\theta)\dot{\theta})^2 + \frac{1}{2}I_{body}\dot{\theta}^2 + M_W\dot{x}^2 + (J_W + J_M)\dot{\phi}^2 - M_{body}g(R_W + h_{CoM} \cos(\theta))$$

4. Non-Conservative Forces / Generalized Forces (Q_i):

- **Motor Torque:** Let τ_M be the torque produced by each motor. The total torque applied to the wheels is $2\tau_M$. This torque acts on the wheels, and by Newton's third law, an equal and opposite torque acts on the body.
- **Friction:** We'll consider motor viscous friction b_M .
- **External Force:** F_{ext} (e.g., from disturbance).

The generalized forces are:

- Q_x : Force acting in the x-direction. This is the sum of the horizontal forces at the contact point between the wheels and the ground due to motor torque. Let $F_{contact}$ be the horizontal force at the contact point of each wheel.
- $Q_x = 2F_{contact}$
- Q_θ : Net torque acting on the body to change θ .
 - This is the reaction torque from the motor on the body. If the motor applies τ_M to the wheel, it applies $-\tau_M$ to the body. So, $Q_\theta = -2\tau_M$.
- Q_ϕ : Net torque acting on the wheels to change ϕ .
 - This is the motor torque applied to the wheels, minus friction.
 - $Q_\phi = 2\tau_M - 2b_M\dot{\phi}$

5. Constraint Equation (No-Slip):

The horizontal velocity of the wheel's contact point with the ground is zero.

$$\dot{x} - R_W\dot{\phi} = 0 \implies \dot{x} = R_W\dot{\phi}$$

This means $\dot{\phi} = \dot{x}/R_W$ and $\phi = x/R_W$. We can use this to reduce the number of generalized coordinates, or use Lagrange multipliers. Let's use it to simplify the Lagrangian.

Substitute $\dot{\phi} = \dot{x}/R_W$ into T :

$$T = \frac{1}{2}M_{body}(\dot{x} - h_{CoM} \cos(\theta)\dot{\theta})^2 + \frac{1}{2}I_{body}\dot{\theta}^2 + M_W\dot{x}^2 + (J_W + J_M) \left(\frac{\dot{x}}{R_W} \right)^2$$

$$T = \frac{1}{2}M_{body}(\dot{x} - h_{CoM} \cos(\theta)\dot{\theta})^2 + \frac{1}{2}I_{body}\dot{\theta}^2 + M_W\dot{x}^2 + \frac{(J_W + J_M)}{R_W^2}\dot{x}^2$$

Now we have two generalized coordinates: x and θ .

Euler-Lagrange Equations for $q_x = x$ and $q_\theta = \theta$:

1. **Equation for x (Horizontal Motion):**

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{x}} \right) - \frac{\partial L}{\partial x} = Q_x$$

$$\frac{\partial L}{\partial \dot{x}} = M_{body}(\dot{x} - h_{CoM} \cos(\theta)\dot{\theta}) + 2M_W\dot{x} + 2\frac{(J_W + J_M)}{R_W^2}\dot{x}$$

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{x}} \right) = M_{body}(\ddot{x} - h_{CoM}(-\sin(\theta)\dot{\theta}^2 + \cos(\theta)\ddot{\theta})) + 2M_W\ddot{x} + 2\frac{(J_W + J_M)}{R_W^2}\ddot{x}$$

$$\frac{\partial L}{\partial x} = 0 \text{ (assuming no external x-force like wind)}$$

So,

$$M_{body}(\ddot{x} + h_{CoM} \sin(\theta)\dot{\theta}^2 - h_{CoM} \cos(\theta)\ddot{\theta}) + 2(M_W + \frac{J_W + J_M}{R_W^2})\ddot{x} = 2F_{contact}$$

Let $M_{eq} = M_{body} + 2M_W + 2\frac{J_W + J_M}{R_W^2}$. This is the effective mass for horizontal translation.

$$M_{eq}\ddot{x} - M_{body}h_{CoM} \cos(\theta)\ddot{\theta} + M_{body}h_{CoM} \sin(\theta)\dot{\theta}^2 = 2F_{contact} \quad (\text{Equation 1: Horizontal Motion})$$

2. **Equation for θ (Pitch Motion):**

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{\theta}} \right) - \frac{\partial L}{\partial \theta} = Q_\theta$$

$$\frac{\partial L}{\partial \dot{\theta}} = -M_{body}h_{CoM} \cos(\theta)(\dot{x} - h_{CoM} \cos(\theta)\dot{\theta}) + I_{body}\dot{\theta}$$

Let $A = -M_{body}h_{CoM} \cos(\theta)$ and $B = M_{body}h_{CoM}^2 \cos^2(\theta) + I_{body}$. Then $\frac{\partial L}{\partial \dot{\theta}} = A\dot{x} + B\dot{\theta}$.

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{\theta}} \right) = \dot{A}\dot{x} + A\ddot{x} + \dot{B}\dot{\theta} + B\ddot{\theta}$$

$$\dot{A} = M_{body}h_{CoM} \sin(\theta)\dot{\theta}$$

$$\dot{B} = 2M_{body}h_{CoM}^2 \cos(\theta)(-\sin(\theta)\dot{\theta}) = -2M_{body}h_{CoM}^2 \cos(\theta) \sin(\theta)\dot{\theta}$$

So,

$$\begin{aligned} \frac{d}{dt} \left(\frac{\partial L}{\partial \dot{\theta}} \right) &= M_{body}h_{CoM} \sin(\theta)\dot{\theta}\dot{x} - M_{body}h_{CoM} \cos(\theta)\ddot{x} \\ &\quad - 2M_{body}h_{CoM}^2 \cos(\theta) \sin(\theta)\dot{\theta}^2 + (M_{body}h_{CoM}^2 \cos^2(\theta) + I_{body})\ddot{\theta} \end{aligned} \quad (1)$$

Now,

$$\frac{\partial L}{\partial \theta} = \frac{1}{2} M_{body} \dot{\theta} (\dot{x} - h_{CoM} \cos(\theta) \dot{\theta}) (-h_{CoM} (-\sin(\theta))) - M_{body} g (-h_{CoM} \sin(\theta))$$

$$\frac{\partial L}{\partial \theta} = M_{body} h_{CoM} \sin(\theta) \dot{\theta} (\dot{x} - h_{CoM} \cos(\theta) \dot{\theta}) + M_{body} g h_{CoM} \sin(\theta)$$

Now, equate these to Q_θ :

$$\begin{aligned} -M_{body} h_{CoM} \cos(\theta) \ddot{x} + (I_{body} + M_{body} h_{CoM}^2 \cos^2(\theta)) \ddot{\theta} - \\ M_{body} h_{CoM}^2 \sin(\theta) \cos(\theta) \dot{\theta}^2 - M_{body} g h_{CoM} \sin(\theta) = -2\tau_M \end{aligned}$$

(Equation 2: Pitch Motion) (2)

Electrical Dynamics of the Motor:

For each motor, the voltage equation is:

$$V_{in} - V_b = I_A R_M + L_M \frac{dI_A}{dt}$$

$$V_{in} - k_e \dot{\phi} = I_A R_M + L_M \frac{dI_A}{dt}$$

And the motor torque:

$$\tau_M = k_t I_A$$

For simplification, often L_M is neglected, giving:

$$I_A = \frac{V_{in} - k_e \dot{\phi}}{R_M}$$

$$\tau_M = k_t \frac{V_{in} - k_e \dot{\phi}}{R_M}$$

Substitute $\dot{\phi} = \dot{x}/R_W$:

$$\tau_M = \frac{k_t}{R_M} V_{in} - \frac{k_t k_e}{R_M R_W} \dot{x}$$

Let $K_M = \frac{k_t}{R_M}$ (motor constant) and $K_B = \frac{k_t k_e}{R_M R_W}$ (back EMF constant reflected to wheel velocity).

$$\tau_M = K_M V_{in} - K_B \dot{x}$$

Full Non-Linear Coupled Model:

Combine Equation 1, Equation 2, and the motor torque expression. We have three coupled non-linear differential equations (if we include the motor electrical dynamics without simplifying L_M , otherwise two mechanical equations and one algebraic motor torque equation).

We can express $2F_{contact}$:

$$2F_{contact} = M_{eq} \ddot{x} - M_{body} h_{CoM} \cos(\theta) \ddot{\theta} + M_{body} h_{CoM} \sin(\theta) \dot{\theta}^2$$

The torque generated by the motors, $2\tau_M$, is the non-conservative force Q_θ .

$$2\tau_M = 2(K_M V_{in} - K_B \dot{x})$$

Substitute into Equation 2:

$$\begin{aligned} -M_{body}h_{CoM}\cos(\theta)\ddot{x} + (I_{body} + M_{body}h_{CoM}^2\cos^2(\theta))\ddot{\theta} \\ - M_{body}h_{CoM}^2\sin(\theta)\cos(\theta)\dot{\theta}^2 - M_{body}gh_{CoM}\sin(\theta) \end{aligned} \quad (3)$$

$$= -2(K_M V_{in} - K_B \dot{x})$$

These are the non-linear equations of motion.

Linearized Model for Control Design:

For control system design (e.g., using LQR), it's common to linearize the model around the upright equilibrium position ($\theta = 0, \dot{\theta} = 0, \dot{x} = 0, \ddot{x} = 0$).

Assumptions for linearization:

- $\sin(\theta) \approx \theta$
- $\cos(\theta) \approx 1$
- $\dot{\theta}^2 \approx 0$ (higher order term)

Linearized Equation 1 (Horizontal Motion):

$$M_{eq}\ddot{x} - M_{body}h_{CoM}\ddot{\theta} = 2F_{contact} \quad (\text{This is the force at the contact point})$$

Linearized Equation 2 (Pitch Motion):

$$-M_{body}h_{CoM}\ddot{x} + (I_{body} + M_{body}h_{CoM}^2)\ddot{\theta} - M_{body}gh_{CoM}\theta = -2\tau_M$$

Let $I_{total_pitch} = I_{body} + M_{body}h_{CoM}^2$. This is the moment of inertia of the body about the wheel axle line if the body was a point mass at its CoM. So,

$$-M_{body}h_{CoM}\ddot{x} + I_{total_pitch}\ddot{\theta} - M_{body}gh_{CoM}\theta = -2\tau_M$$

Linearized Motor Torque:

$$\tau_M = K_M V_{in} - K_B \dot{x}$$

Substitute linearized τ_M into the linearized pitch equation:

$$\begin{aligned} -M_{body}h_{CoM}\ddot{x} + I_{total_pitch}\ddot{\theta} - M_{body}gh_{CoM}\theta &= -2(K_M V_{in} - K_B \dot{x}) \\ -M_{body}h_{CoM}\ddot{x} + I_{total_pitch}\ddot{\theta} - M_{body}gh_{CoM}\theta &= -2K_M V_{in} + 2K_B \dot{x} \end{aligned}$$

State-Space Representation:

Define state variables:

$$x = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}$$

We need to solve the two coupled linear equations for $\ddot{x}$ and $\ddot{\theta}$. Let's rewrite them:

1. $M_{eq}\ddot{x} - (M_{body}h_{CoM})\ddot{\theta} = 2F_{contact}$
2. $-(M_{body}h_{CoM})\ddot{x} + I_{total_pitch}\ddot{\theta} = M_{body}gh_{CoM}\theta + 2K_B\dot{x} - 2K_MV_{in}$

This forms a matrix equation:

$$\begin{bmatrix} M_{eq} & -M_{body}h_{CoM} \\ -M_{body}h_{CoM} & I_{total_pitch} \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 2F_{contact} \\ M_{body}gh_{CoM}\theta + 2K_B\dot{x} - 2K_MV_{in} \end{bmatrix}$$

The external force $2F_{contact}$ is related to the internal motor torque. In the ideal case, $2F_{contact} = 2\tau_M/R_W = 2(K_MV_{in} - K_B\dot{x})/R_W$.

Let's assume the input is V_{in} and the output is θ . The state equations will be in the form $\dot{x} = Ax + Bu$.

This gives a comprehensive mathematical model from non-linear to linearized forms, accounting for all significant physical parameters. For real-world implementation, friction parameters, motor efficiencies, and sensor noise would also need to be considered.

THE FINAL LINEARIZED MODEL

1. **Horizontal Motion:** $M_{eq}\ddot{x} - (M_{body}h_{CoM})\ddot{\theta} = \frac{2}{R_W}\tau_M$ where $M_{eq} = M_{body} + 2M_W + 2\frac{J_W + J_M}{R_W^2}$
2. **Pitch Motion:** $-(M_{body}h_{CoM})\ddot{x} + I_{total_pitch}\ddot{\theta} = M_{body}gh_{CoM}\theta - 2\tau_M$ where $I_{total_pitch} = I_{body} + M_{body}h_{CoM}^2$

And the linearized motor torque relationship:

$$\tau_M = K_MV_{in} - K_B\dot{x}$$

where $K_M = \frac{k_t}{R_M}$ and $K_B = \frac{k_t k_e}{R_M R_W}$

Let's substitute the expression for τ_M into the two mechanical equations.

Substitute τ_M into Equation 1:

$$\begin{aligned} M_{eq}\ddot{x} - (M_{body}h_{CoM})\ddot{\theta} &= \frac{2}{R_W}(K_MV_{in} - K_B\dot{x}) \\ M_{eq}\ddot{x} - (M_{body}h_{CoM})\ddot{\theta} + \frac{2K_B}{R_W}\dot{x} &= \frac{2K_M}{R_W}V_{in} \quad (\text{Equation 1'}) \end{aligned}$$

Substitute τ_M into Equation 2:

$$-(M_{body}h_{CoM})\ddot{x} + I_{total_pitch}\ddot{\theta} = M_{body}gh_{CoM}\theta - 2(K_M V_{in} - K_B \dot{x})$$

$$-(M_{body}h_{CoM})\ddot{x} + I_{total_pitch}\ddot{\theta} - 2K_B \dot{x} - M_{body}gh_{CoM}\theta = -2K_M V_{in} \quad (\text{Equation 2'})$$

Now we have a system of two coupled linear differential equations with $\ddot{x}$ and $\ddot{\theta}$ as the highest derivatives. We need to express $\ddot{x}$ and $\ddot{\theta}$ in terms of the state variables and the input.

Let's write this in matrix form:

$$\begin{bmatrix} M_{eq} & -M_{body}h_{CoM} \\ -M_{body}h_{CoM} & I_{total_pitch} \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{2K_B}{R_W} \dot{x} + \frac{2K_M}{R_W} V_{in} \\ M_{body}gh_{CoM}\theta + 2K_B \dot{x} - 2K_M V_{in} \end{bmatrix}$$

Let $M = \begin{bmatrix} M_{eq} & -M_{body}h_{CoM} \\ -M_{body}h_{CoM} & I_{total_pitch} \end{bmatrix}$. We need to invert M to solve for $\begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix}$:

$$\begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} = M^{-1} \begin{bmatrix} \frac{2K_B}{R_W} \dot{x} + \frac{2K_M}{R_W} V_{in} \\ M_{body}gh_{CoM}\theta + 2K_B \dot{x} - 2K_M V_{in} \end{bmatrix}$$

The determinant of M is $\Delta = M_{eq}I_{total_pitch} - (M_{body}h_{CoM})^2$.

$$M^{-1} = \frac{1}{\Delta} \begin{bmatrix} I_{total_pitch} & M_{body}h_{CoM} \\ M_{body}h_{CoM} & M_{eq} \end{bmatrix}$$

Now, let's define the state variables: $x_1 = x$ (horizontal position of the wheel axle) $x_2 = \dot{x}$ (horizontal velocity) $x_3 = \theta$ (pitch angle of the robot body) $x_4 = \dot{\theta}$ (pitch angular velocity)

The state vector is $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$. The input is $u = V_{in}$.

From the definitions: $\dot{x}_1 = x_2$ $\dot{x}_3 = x_4$

Now we need to find expressions for $\dot{x}_2 = \ddot{x}$ and $\dot{x}_4 = \ddot{\theta}$ using the inverted matrix equation.

$$\ddot{x} = \frac{1}{\Delta} \left[I_{total_pitch} \left(\frac{2K_B}{R_W} \dot{x} + \frac{2K_M}{R_W} V_{in} \right) + M_{body}h_{CoM} (M_{body}gh_{CoM}\theta + 2K_B \dot{x} - 2K_M V_{in}) \right] \quad (4)$$

$$\ddot{\theta} = \frac{1}{\Delta} \left[M_{body}h_{CoM} \left(\frac{2K_B}{R_W} \dot{x} + \frac{2K_M}{R_W} V_{in} \right) + M_{eq} (M_{body}gh_{CoM}\theta + 2K_B \dot{x} - 2K_M V_{in}) \right]$$

$$\ddot{\theta} = \frac{1}{\Delta} \left[\left(\frac{2K_B M_{body}h_{CoM}}{R_W} + 2K_B M_{eq} \right) \dot{x} + M_{eq} M_{body}gh_{CoM}\theta + \left(\frac{2K_M M_{body}h_{CoM}}{R_W} - 2K_M M_{eq} \right) V_{in} \right]$$

Let's collect the terms in the state-space form: $\dot{x} = Ax + Bu$

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 = \ddot{x} &= \left[\frac{1}{\Delta} \left(2K_B M_{body} h_{CoM} - \frac{2K_B I_{total_pitch}}{R_W} \right) \right] x_2 \\
&\quad + \left[\frac{g(M_{body} h_{CoM})^2}{\Delta} \right] x_3 \\
&\quad + \left[\frac{1}{\Delta} \left(\frac{2K_M I_{total_pitch}}{R_W} - 2K_M M_{body} h_{CoM} \right) \right] u
\end{aligned} \tag{5}$$

$$\begin{aligned}
\dot{x}_3 &= x_4 \\
\dot{x}_4 = \ddot{\theta} &= \left[\frac{1}{\Delta} \left(2K_B M_{eq} - \frac{2K_B M_{body} h_{CoM}}{R_W} \right) \right] x_2 \\
&\quad + \left[\frac{M_{eq} M_{body} g h_{CoM}}{\Delta} \right] x_3 \\
&\quad + \left[\frac{1}{\Delta} \left(\frac{2K_M M_{body} h_{CoM}}{R_W} - 2K_M M_{eq} \right) \right] u
\end{aligned} \tag{6}$$

Now, write the state matrix A and input matrix B :

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{1}{\Delta} \left(2K_B M_{body} h_{CoM} - \frac{2K_B I_{total_pitch}}{R_W} \right) & \frac{g(M_{body} h_{CoM})^2}{\Delta} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{1}{\Delta} \left(2K_B M_{eq} - \frac{2K_B M_{body} h_{CoM}}{R_W} \right) & \frac{M_{eq} M_{body} g h_{CoM}}{\Delta} & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ \frac{1}{\Delta} \left(\frac{2K_M I_{total_pitch}}{R_W} - 2K_M M_{body} h_{CoM} \right) \\ 0 \\ \frac{1}{\Delta} \left(\frac{2K_M M_{body} h_{CoM}}{R_W} - 2K_M M_{eq} \right) \end{bmatrix}$$

Output Equation: If we are interested in measuring the horizontal position (x) and the pitch angle (θ), the output equation is $y = Cx + Du$.

$$y = [\theta]$$

$$C = [0 \quad 0 \quad 1 \quad 0]$$

$$D = [0]$$

This completes the LTI state-space model of the robot.

Robot Physical Parameters and Final State Space Representation:

The values of the physical parameters are computed and substituted as:

$M_{body} = 1.084$	(mass of robot excluding wheel and motor)
$K_e = 0.118$	(back emf constant)
$R_M = 8$	(terminal resistance)
$R_W = 0.05$	(radius of wheel)
$K_b = K_e^2 / (R_M R_W)$	
$h_{CoM} = 0.1314$	(cog to centre of wheel)
$I_{body} = 0.02647$	(body inertia)
$I_{total} = I_{body} + M_{body}(h_{CoM})^2$	(total pitch)
$M_W = 0.096$	(mass of wheel)
$J_W = 0.5M_W(R_W)^2$	(wheel inertia)
$J_M = 0$	(rotor inertia)
$M_{eq} = M_{body} + (2M_W) + (2(J_W + J_M)/(R_W^2))$	(total mass)
$K_m = K_e / R_M$	(motor constant)
$\delta = M_{eq}I_{total} - (M_{body}h_{CoM})^2$	
$g = 9.81$	(acc due to gravity)

Based on these parameters, the state space is thus computed as:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -1.27078736 & 4.77208256 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -2.46508122 & 45.96607409 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0.53846922 \\ 0 \\ 1.04452594 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 \end{bmatrix}$$

MATLAB SIMULATION

State Space and Transfer Function:

The state-space model computed above is simulated using MATLAB to obtain the transfer function:

$$G(s) = \frac{\theta(s)}{V(s)} = \frac{1.045s - 1.053 \times 10^{-15}}{s^3 + 1.271s^2 - 45.97s - 46.65}$$

Using MATLAB, the poles and zeros of the system without controller are found. Poles of the transfer function are 6.6697, -6.9314 and -1.0091 and zero of the transfer function is 1.0084×10^{-15}

A pole located in the right-half of the complex s-plane signifies that the system is inherently unstable, causing the robot to fall over if left uncontrolled. The other two poles are located in the left-half of the s-plane, indicating stable modes. However, the presence of even a single right-half plane pole makes the entire system unstable.

PID TUNING

The SISO Design Tool (sisotool) in MATLAB is used for designing and tuning the PID controller for the given system with the aim to shape the system's time-domain response for faster rise time, reduced overshoot, and improved settling time, while ensuring system stability. The tuning method was selected as "Robust response time" and the design mode was set to "Time Domain". The Response Time was set as 0.227 and the Transient Behavior was set as 0.2442. After tuning, the resulting PID controller was of the form:

$$C(s) = 332.42 \times \frac{(1 + 0.17s)(1 + 0.17s)}{s}$$

The closed-loop step response of the system after PID tuning was analyzed to evaluate the performance of the designed controller.

The following characteristics were observed from the step response plot:

- The system exhibits a rapid rise time of approximately **0.5 seconds**, indicating that the robot quickly begins to respond to any change in the reference input or external disturbance.
- The step response shows a moderate overshoot with a peak amplitude of around **1.8**.
- The system settles within approximately **2 seconds**, indicating that the oscillations damp out quickly and the output stabilizes close to the desired steady-state value.
- The response remain stable throughout, with minimal oscillations and **no sustained steady-state error**.

These time-domain characteristics indicate that the robot is now capable of rapidly correcting its tilt angle and returning to its upright balanced position after receiving a command input or experiencing a disturbance. Thus, the designed PID controller successfully improves the robot's dynamic response and stability.

The final PID controller gains were obtained as follows:

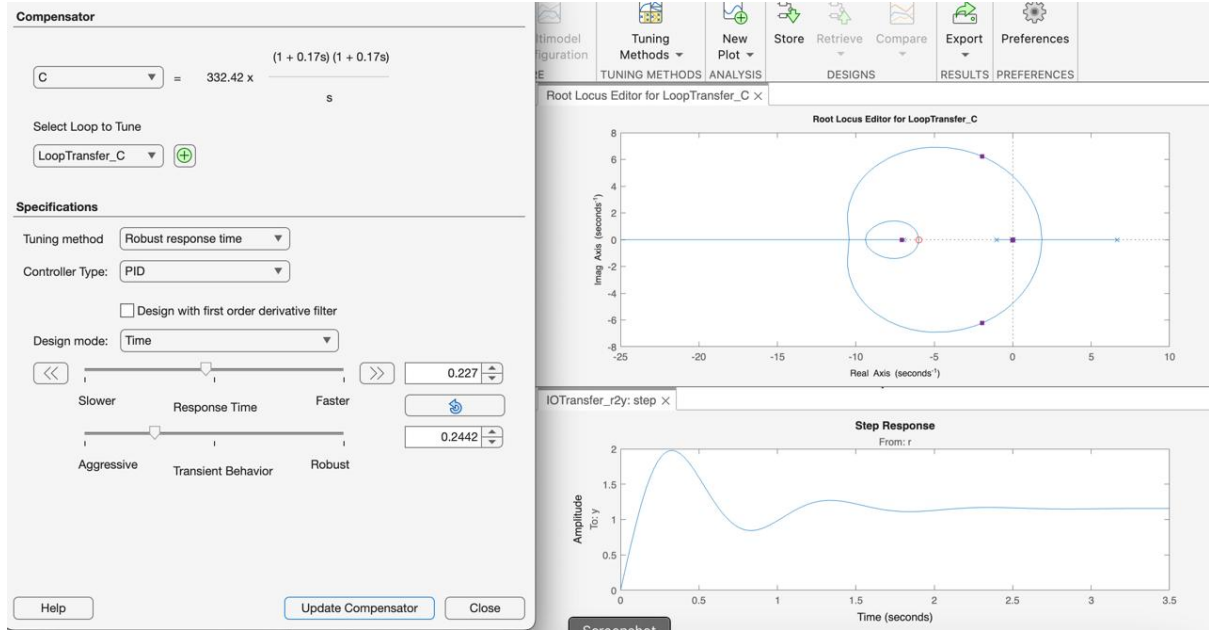


Figure 5: PID Tuning

$$K_p = 113.02$$

$$K_i = 332.42$$

$$K_d = 9.61$$

These gains are then used in the robot's control algorithm to ensure the desired balancing performance.

ARDUINO CODE

The self-balancing robot operates through a continuous, integrated process designed to maintain its upright position. This process begins with the initial activation and calibration of the MPU6500 IMU. During calibration, the robot learns its neutral position by precisely accounting for Earth's gravity, ensuring that all subsequent measurements of its tilt are accurate.

Once calibrated, the system continuously gathers information by constantly reading data from its IMU. The initial tilt angles derived solely from acceleration data, which can be somewhat noisy, are combined with highly precise rotational speed data from the gyroscope which is then processed by a Kalman filter providing a much more accurate, smooth, and reliable estimate of the robot's actual tilt (pitch and roll angles). This precise angle information is fundamental for the robot to stay balanced.

If the robot's tilt angle moves beyond a smaller, acceptable range (such as 5 degrees from perfectly upright), the PID control loop activates. The system then calculates the difference between its desired upright position and its current tilted angle. Based on this error, the PID algorithm determines how much corrective action is needed by considering the current tilt, how long it has been tilted, and how quickly it's tilting.

The calculated PID output translates directly into motor commands. This output, representing the required balancing effort, is then adjusted to fit the appropriate speed range for the robot's motors. These calculated motor speeds and directions are then sent to the motor driver, which instructs the motors to spin in a way that generates the necessary force to push the robot back towards its upright target position, actively counteracting the tilt. If the robot's tilt angle exceeds a specific critical limit (for example, tilting more than 30 degrees), the system automatically cuts power to the motors. This rapid response is designed to prevent the robot from completely falling over and potentially damaging itself. The system continually works to maintain balance.

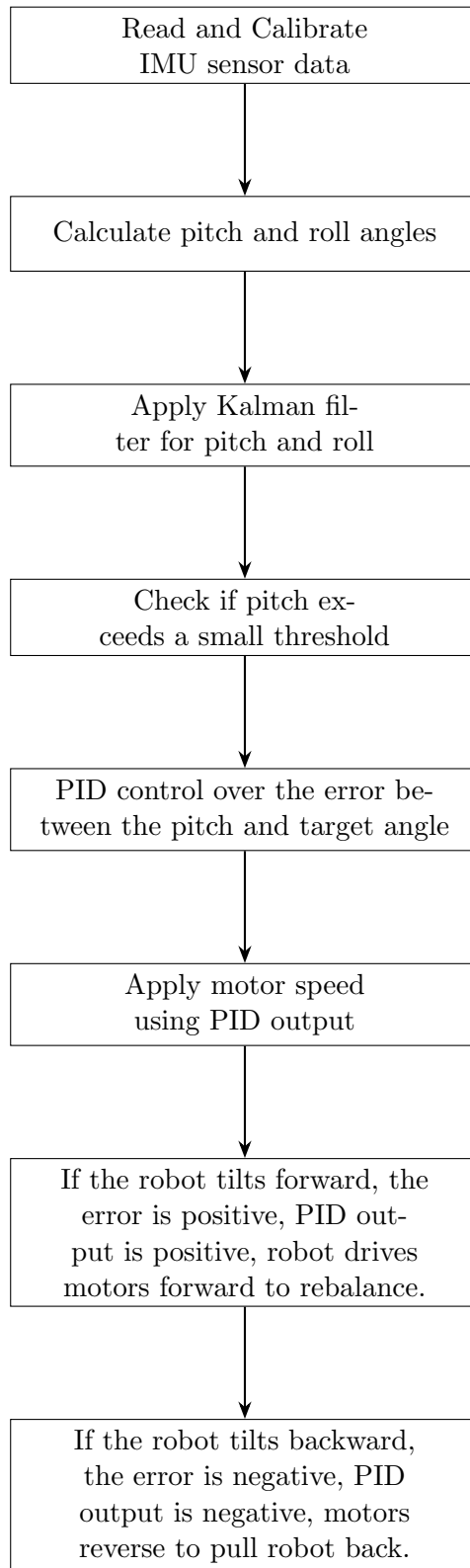


Figure 6: Flowchart of Self-Balancing Robot Control Logic

CHALLENGES

The development of the self-balancing robot involved several practical challenges due to both hardware limitations and system modeling assumptions. Below are the key challenges encountered in the project:

- **Nonlinear System Dynamics:** The actual dynamics of a self-balancing robot are highly nonlinear, especially for large tilt angles. However, for controller design, the system is linearized around small angles (typically within $\pm 10\text{--}15^\circ$). This linearization assumption limits the stable operating range of the controller and can lead to instability if the robot tilts beyond this range. Possible solutions include designing more advanced nonlinear or adaptive controllers.
- **Modeling Errors:** Accurate modeling of the system parameters (such as mass, inertia, and center of gravity location) is essential for effective control. Small modeling errors or parameter mismatches can degrade controller performance.
- **Prototype Inconsistencies:** Variations in component placement, such as uneven battery or wiring positions, can shift the robot's center of gravity. This shift affects balance and introduces unexpected behavior.
- **IMU Noise:** The MPU6500 IMU introduces noise, bias drift over time, and measurement delays. These issues reduce the accuracy of tilt angle estimation and affect control decisions.
- **Structural Flexibility and Vibrations:** Mechanical vibrations and flexing of structural components affect IMU readings and delay motors' responses.
- **Wheels:** Heavier wheels increase angular momentum, making it difficult for the robot to quickly correct its position. Additionally, any mechanical play or loose coupling between the motor shaft and wheels introduces backlash, causing delays in response. Choosing appropriate wheel mass and ensuring tight coupling between motor and wheels can address these issues.
- **Center of Gravity Shifts:** Asymmetric frame design or uneven weight distribution caused lateral CoG shifts. This can be minimized by ensuring symmetric placement of all components and keeping the CoG as close to the vertical axis as possible.
- **Surface Irregularities:** Variations in surface friction and uneven floors can affect robot balance.
- **Sensor-Motor Delays:** Small delays in IMU data acquisition and motor response introduce phase lag affecting real-time control.

These factors emphasize the importance of careful mechanical design, accurate system modeling, reliable sensor data processing, and robust controller tuning. Addressing these challenges requires a combination of hardware improvements, sensor filtering techniques, and iterative control parameter optimization to ensure that the robot maintains balance effectively within its operational limits.

CONCLUSION

The design and implementation of the self-balancing robot presented in this project demonstrates the integration of classical mechanics, control theory, and embedded systems. By modelling the robot as an inverted pendulum and linearizing its dynamics around the unstable upright equilibrium point, a state-space representation was derived to characterize the system's behaviour. Through rigorous system modelling, including the derivation of the Lagrangian and the incorporation of motor electrical dynamics, the foundation was laid for implementing real-time control. The PID controller was designed and tuned to stabilize the system, mitigating the effects of external disturbances and inherent instability. Simulation results validated the theoretical model, indicating the system's capacity to achieve and maintain balance within small angular deviations. From a hardware perspective, the project involved careful selection and integration of mechanical and electronic components to meet the dynamic and structural demands of a self-balancing system. The robot was built on a robust, tiered chassis with optimal placement of the center of mass to enhance control responsiveness. The MPU6500 IMU provided real-time tilt measurements, which were processed using Kalman filtering for noise reduction and improved accuracy. The Arduino Mega 2560 served as the control platform, executing the PID algorithm and managing motor actuation via the L298N motor driver. This setup facilitated closed-loop control, with precise feedback-driven adjustments to motor speed and direction, ensuring dynamic equilibrium in real time. This project not only exemplifies the practical application of theoretical control principles but also underscores the challenges involved in building an autonomous balancing system. The successful implementation of a PID-controlled self-balancing robot lays the groundwork for future advancements, such as adaptive control techniques, nonlinear observers, or machine learning-based stabilization. Moreover, the platform offers a valuable educational and research tool for further exploration of feedback control, robotics, and real-time embedded systems in unstable dynamic environments.

References

- [1] K. V. Kiran, N. Sharma, A. Gaur, and M. Patidar, “Self Balancing Rover Using PID Control,” *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, vol. 10, no. 5, pp. 1784–1790, May 2022.
- [2] M. T. Islam, S. M. R. Joarder, and M. S. Islam, “Application of LQR Control for Two-Wheel Self-Balancing Robot,” *International Journal of Research in Engineering and Science (IJRES)*, vol. 9, no. 1, pp. 01–08, 2021.
- [3] J. Wu and W. Zhang, “Design of fuzzy logic controller for two-wheeled self-balancing robot,” in *Proceedings of 2011 6th International Forum on Strategic Technology*, vol. 2, pp. 1266–1270, IEEE, Aug. 2011.
- [4] N. G. M. Thao, D. H. Nghia, and N. H. Phuc, “A PID backstepping controller for two-wheeled self-balancing robot,” in *2010 International Forum on Strategic Technology*, pp. 76–81, IEEE, Oct. 2010.
- [5] J. Kim, I. Lee, and J. Back, “Balance control of two-wheeled robot based on reinforcement learning,” in *2010 IEEE International Conference on Robotics and Biomimetics*, pp. 2978–2983, IEEE, Dec. 2010.
- [6] V. Mudeng, B. Hassanah, Y.T.K. Priyanto, and O. Saputra, “Design and Simulation of Two-Wheeled Balancing Mobile Robot with PID Controller,” *Proceedings of the 2021 International Conference on Computer Engineering, Network and Intelligent Multimedia (CENIM)*, Institut Teknologi Kalimantan, Indonesia, 2021.
- [7] H. Chin, “2.004 Lab 8 Intro: Self-Balancing Robot Control, Part I: Stabilization Using PID Control,” Massachusetts Institute of Technology, Fall 2019.
- [8] J. Gafford, J. Chen, and G. Eng, “Modeling and Control of a Two-Wheeled Self-Balancing Robot,” Final Project Report, School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA, 2012.
- [9] A. Goswami, M. Kundu, and D. Chakraborty, “Modeling and Simulation of Two-Wheeled Self Balancing Robot,” in *2016 IEEE First International Conference on Control, Measurement and Instrumentation (CMI)*, pp. 112–116, Jan. 2016.
- [10] D.P.J. van den Brandt, “Two-wheeled Robot Stabilization and Motion Control,” Master’s Thesis, Eindhoven University of Technology, The Netherlands, 2010. Supervisors: Prof. Dr. K.G.W. Goossen.
- [11] J. Weibel and L. Gräser, “NXT Ballbot: Model-Based Design – Control of Self-Balancing Robot on a Ball, Built with LEGO Mindstorms NXT,” *Institute of Automatic Control Engineering, Technical University of Munich*, 2009.
- [12] O. Halawa and A. Maged, “Mechatronics Project: Two-Wheeled Self-Balancing Robot,” Helwan University, Egypt, May 2023.
- [13] Y. Ding, J. Gafford, and M. Kunio, “Modeling, Simulation and Fabrication of a Balancing Robot,” Harvard University and Massachusetts Institute of Technology.
- [14] V.Y. Philippart, K.O. Snel, A.M. de Waal, J.S.Y. Jeedella, and E. Najafi, “Model-Based Design for a Self-Balancing Robot Using the Arduino Micro-Controller Board,”

Electrical and Mechatronics Engineering Departments, Fontys University of Applied Sciences, Eindhoven, Netherlands, 2017.

- [15] A.H. Ahmed and O.E. Aburas, “A Guide to Implement a Two-Wheeled Robot Using Pole-Placement on Arduino,” *Elmergib Journal of Engineering Research*, vol. 1, no. 1, pp. 1–10, Jun. 2020.
- [16] H.-S. Juang and K.-Y. Lum, “Design and Control of a Two-Wheel Self-Balancing Robot Using the Arduino Microcontroller Board,” in *Proceedings of the 2013 10th IEEE International Conference on Control and Automation (ICCA)*, Hangzhou, China, June 2013, pp. 634–639.
- [17] R. Bakiya Lakshmi, S.M. Bashir, and G.S.J. Daniel, “Design, Simulation and Optimal Control of Two-Wheel Self-Balancing Robot Using LQR and PID,” *International Journal of Engineering and Advanced Technology (IJEAT)*, vol. 11, no. 3, pp. 20–25, Feb. 2022.
- [18] P. Frankovský, L. Dominik, A. Gmitterko, I. Virgala, P. Kurylo, and O. Perminova, “Modeling of Two-Wheeled Self-Balancing Robot Driven by DC Gearmotors,” *Acta Mechanica Slovaca*, vol. 19, no. 4, pp. 6–11, 2015.
- [19] H. Huang, J. Yuan, and Q. Pan, “A Modified Dynamical Formulation for Two-Wheeled Self-Balancing Robots,” *International Conference on Information and Automation*, pp. 364–369, 2011.
- [20] K. Lauszus, “Balanduino – Balancing Robot Kit,” TKJ Electronics, 2025.
- [21] L.G.B. Putra, F. Wahab, and T.A. Tamba, “Design and Implementation of Linear Quadratic Regulator Control for Two-Wheeled Self-Balancing Robot,” *Bull. Electr. Eng. Inf.*
- [22] R.Varthan, S.Ganesh, and A.M.Anbarasi, “Real-Time Implementation of a Self-Balancing Robot using PID Controller,” *Int. J. Eng. Res. Tech. (IJERT)*
- [23] O.A. Choudhry, M. Wasim, A. Ali, M.A. Choudhry, and J.Iqbal, “Modelling and Robust Controller Design for an Underactuated Self-Balancing Robot with Uncertain Parameter Estimation,” *PLOS ONE*, vol.18, no.8, e0285495, Aug 2023.
- [24] A. Abdelgawad, T. Shohdy, and A. Nada, “Model- and Data-Based Control of Self-Balancing Robots: Practical Educational Approach with LabVIEW and Arduino,” arXiv, May 2024.
- [25] S. Nougues, S. Melo Guayacan, D. Garzón Cuadros, and H. León-Rodríguez, “Modelling and Design a Self-balancing Dual-wheeled Robot with PID Control,” in *ICCAS 2023*.
- [26] S. Lekshmy, A. George, and C.V. Athira, “Self-Balancing Robot,” Department of Electronics and Communication Engineering, Vimal Jyothi Engineering College, Chemperi, Kannur, India, 2019.
- [27] S. Sánchez Prieto, T. Arribas Navarro, M. Gómez Plaza, and Ó. Rodríguez Polo, “A Monoball Robot Based on LEGO Mindstorms,” *Journal of Intelligent and Robotic Systems*, vol. 77, no. 3–4, pp. 449–464, 2015.

-
- [28] R.A. García-García and M. Arias-Montiel, “Linear Controllers for the NXT Ballbot with Parameter Variations Using Linear Matrix Inequalities,” *Journal of Applied Research and Technology*, vol. 15, no. 4, pp. 354–362, 2017.