



ORIGINAL CONTRIBUTION

SELF BALANCING BOT USING ARDUINO UNO AND MPU6050

¹Mayank Vatsa, ²Nimisha Nisha, ³Pushpanjay Gaurav, ⁴Priyanshu Kumar Gupta, ⁵Kamanashis Goswami

1,2,3,4UG student, Department of Electronics & Communication Engineering, Haldia Institute of Technology, Haldia, Purba Medinipur, West Bengal.

5Assistant Professor, Department of Electronics & Communication Engineering, Haldia Institute of Technology, Haldia, Purba Medinipur, West Bengal.

ABSTRACT

Two-wheeled self-balancing robots are compact mobile robotic systems based on the principle of an inverted pendulum. Unlike conventional four-wheeled robots, a two-wheeled balancing robot is inherently unstable and must continuously sense its orientation and generate corrective motor commands to remain upright. This makes it an effective experimental platform for studying feedback control, embedded systems, inertial sensing, and robotics. This paper presents the design and implementation of a low-cost two-wheeled self-balancing robot using an Arduino UNO, MPU6050 IMU, L298N motor driver, DC geared motors, and a 7.4-V battery. The MPU6050 measures acceleration and angular velocity to estimate the robot's pitch angle, which is compared with a setpoint of 176 and controlled using a PID controller with $K_p=100$, $K_i=140$, and $K_d=0.8$. The controller generates PWM motor commands to maintain balance, using DMP-based orientation processing and a 10-ms sampling interval. The study demonstrates the feasibility of dynamic balancing with low-cost hardware and discusses sensor calibration, PID tuning, limitations, applications, and future improvements such as encoder feedback, Kalman filtering, and LQR control.

KEYWORDS: Self-Balancing Robot, Two-Wheeled Robot, Feedback Control, Embedded Robotics.

1. INTRODUCTION

Two-wheeled self-balancing robots are an important and interesting area of robotics and control engineering. Unlike conventional robots with four or more wheels, a two-wheeled robot must continuously maintain its own balance while operating. Its structure can be considered similar to an inverted pendulum, where even a small change in the tilt angle can cause the robot to become unstable and fall. Therefore, the system needs to continuously sense its orientation and make quick corrective movements to maintain stability [2], [3]. Self-balancing robots have potential applications in personal transportation, surveillance, warehouse automation, service robotics, and educational robotics. They are also widely used as experimental platforms for studying and comparing control techniques such as PID, LQR, fuzzy control, and other advanced methods [2], [3], [5]. In this project, an Arduino

UNO is used as the main controller because of its low cost, simple architecture, and ease of programming. The Arduino UNO is based on the ATmega328P microcontroller and provides sufficient digital, analog, and PWM interfaces for controlling the prototype [1]. An MPU6050 IMU is used to measure the robot's motion and tilt by combining data from a three-axis accelerometer and a three-axis gyroscope. The sensor data is used to estimate the robot's tilt angle, which is then provided to a PID controller. The controller calculates the required corrective action and adjusts the motor speed and direction to bring the robot back toward its upright position [4], [5].

2. LITERATURE REVIEW

Villacrés et al. [2] compared PID, LQR, and Sliding Mode Control (SMC) for stabilizing a two wheeled inverted-pendulum robot highlighting

the importance of suitable feedback-control techniques for maintaining stability.

Bakiya Lakshmi et al. [3] investigated PID and LQR-base control for a two-wheel self-balancing robot, demonstrating the application of classical and optimal control techniques for balancing.

Abdelgawad et al. [4] presented a model- and data-based control approach using Arduino for self-balancing robots demonstrating, the suitability of low-cost embedded platforms for practical control experiments.

Putra et al. [5] and Nam [6] investigated LQR and robust PI-based control approaches for two-wheeled self-balancing and inverted-pendulum robots, focusing on stability and control performance.

Krishna et al. [7] explored PID combined with deep reinforcement learning, while Hassan et al. [8] investigated MPU6050-based sensing with PID control for self-balancing robots. These studies demonstrate the continued development of both conventional and advanced control approaches for self-balancing systems.

The present work focuses on a low-cost self-balancing robot using Arduino UNO, MPU6050, L298N motor driver, DC geared motors, and PID-based closed-loop control.

3. PROPOSED SYSTEM

3.1. System Architecture

The proposed robot consists of four major subsystems: sensing, processing and control, motor driving, and mechanical and power systems. The MPU6050 continuously senses the robot's tilt and provides the required motion data to the Arduino UNO for orientation estimation and error calculation. The calculated error is processed by the PID controller to generate an appropriate control signal, which is converted into PWM output and given to the L298N motor driver. The driver controls the DC motors to move the wheels in the required direction and correct the robot's tilt. If the robot tilts forward, the wheels move forward, while a backward tilt

causes the motors to rotate in the opposite direction. This continuous sensing, error calculation, control, and correction process maintains the robot in an upright position and forms a closed-loop feedback control system.

3.2. Hardware Implementation

The Arduino UNO serves as the central controller of the self-balancing robot and is responsible for MPU6050 initialization, sensor-data acquisition, pitch-angle processing, PID computation, PWM generation, motor direction control, and serial monitoring. The MPU6050 IMU is used as the primary feedback sensor, combining a three-axis accelerometer and gyroscope to measure acceleration and angular velocity. The pitch angle is particularly important for detecting the forward and backward tilt of the robot. The MPU6050 communicates with the Arduino UNO through the I²C interface, with SDA connected to A4, SCL to A5, and INT to D2. An L298N dual H-bridge motor driver is used to control the two geared DC motors. Its control inputs are connected to D6, D9, D10, and D11 of the Arduino UNO for motor direction and PWM speed control. The motors provide corrective motion to maintain the robot in an upright position and can also produce differential motion for steering. The complete prototype is powered by a 7.4-V battery, while proper power distribution and regulation are used to minimize voltage fluctuations and ensure stable operation of the Arduino, MPU6050, and motors.

Table 1. Hardware components and function

Components	Function
Arduino UNO	Main processing and control unit
MPU6050 IMU	Measures acceleration and angular velocity for orientation estimation
L298N Motor Driver	Controls direction and speed of DC motors
DC Geared Motors	Drive the left and right wheels and provide corrective motion
7.4-V Battery	Provides power to the robot
Two-Wheel Chassis	Provides mechanical support for the robot

3.3. Block Diagram

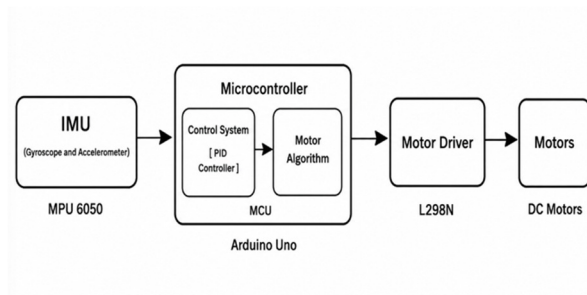


Figure 1: Block diagram of the Self balancing Robot

3.4. Circuit Diagram

The connections for this Arduino-based self-balancing robot are pretty simple. This is a self-balancing robot using Arduino and MPU6050, so we need to interface the MPU6050 with Arduino and connect the motors through the motor driver module. The whole setup is powered by a 7.4V Li-ion battery. The circuit diagram for the same is shown below.

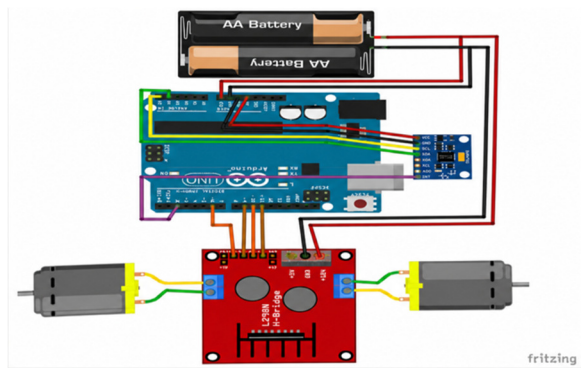


Figure 2: Circuit Diagram of the proposed Self balancing robot

3.5. Pin Configuration

Table2. Pin configuration of the proposed system

Component	Pin	Arduino UNO
MPU6050	VCC	+5V*
MPU6050	GND	GND

MPU6050	SCL	A5
MPU6050	SDA	A4
MPU6050	INT	D2
L298N	IN1	D6
L298N	IN2	D9
L298N	IN3	D10
L298N	IN3	D11

4. MATHEMATICAL MODEL OF THE SYSTEM

The two-wheeled inverted pendulum (TWIP) representation is used to model the dynamics of the self-balancing robot. Unlike a simple cart-pole system, the dynamics of a two-wheeled balancing robot are more complex because wheel rotation, body motion, motor torque, wheel inertia, friction, and non-slip constraints are strongly coupled. Therefore, the nonlinear dynamics are commonly linearized around the upright equilibrium before designing controllers such as PID or LQR. The robot can be approximated as an inverted pendulum mounted on two driven wheels, where M represents the mass of the robot body, m represents the equivalent wheel and motor mass, l is the distance between the wheel axle and the centre of mass, I is the moment of inertia of the robot body, r is the wheel radius, g is the gravitational acceleration, x is the horizontal displacement, θ is the body tilt angle, and u is the motor control input. The upright equilibrium position is represented by $\theta = 0$, and for small disturbances, where $|\theta| \ll 1$, the approximations $\sin \theta \approx \theta$ and $\cos \theta \approx 1$ can be applied to simplify the mathematical model. The desired reference angle is generally taken as $\theta(\text{ref}) = 0$, meaning that the robot should remain perfectly upright. The controller continuously calculates the error between the desired and measured angles as $e(t) = \theta(\text{ref}) - \theta(t)$ and generates a corrective motor command whenever the robot deviates from its upright position, causing the wheels to move in the appropriate direction to restore and maintain balance[2].

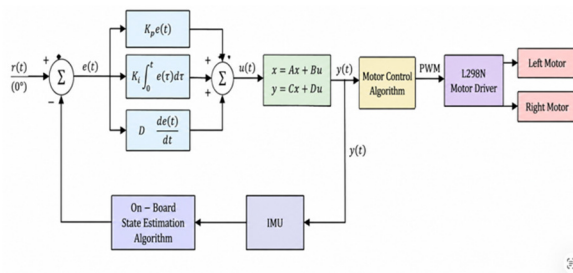


Figure 3: Mathematical model and PID control block diagram

5. SYSTEM IMPLEMENTATION

5.1. MPU6050 Calibration

Calibration is a critical stage in the operation of a self-balancing robot because the MPU6050 may not produce exactly zero readings even when the robot is stationary due to sensor bias, manufacturing tolerances, mechanical mounting errors, temperature variations, and electrical noise. To obtain accurate orientation measurements, the robot is first placed in its intended upright equilibrium position and kept completely stationary. The MPU6050 is then initialized, and the accelerometer and gyroscope readings are measured to determine their respective offsets. These offset values are stored and used as reference values during normal operation. The corresponding pitch reference is also determined so that the controller can accurately identify deviations from the upright position. In addition, the implementation uses the MPU6050's Digital Motion Processor (DMP) to process motion data and obtain orientation information from quaternion and gravity-vector data. Proper calibration ensures that the measured tilt angle is reliable, allowing the PID controller to generate accurate corrective commands and significantly improving the stability and balancing performance of the robot.

5.2. PID Control

The PID (Proportional–Integral–Derivative) controller is used to generate the corrective motor command required to maintain the robot in an upright position. It continuously compares the desired reference angle with the measured pitch angle from the MPU6050 and calculates the error

between them. The continuous-time PID equation is $u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$, where K_p is the proportional gain, K_i is the integral gain, and K_d is the derivative gain. The proportional term, given by $u_P = K_p e(t)$, produces a corrective response proportional to the instantaneous error; a larger tilt results in a stronger motor correction, while a very low K_p can make the robot respond slowly and a very high K_p can cause excessive oscillations. The integral term, given by $u_I = K_i \int e(t) dt$, accumulates the error over time and helps eliminate persistent or steady-state errors. However, excessive integral gain can lead to integral windup, resulting in instability and large oscillations, particularly because a self-balancing robot is a rapidly changing system. The derivative term, given by $u_D = K_d \frac{de(t)}{dt}$, responds to the rate of change of the error and helps predict the robot's movement. A properly tuned derivative gain provides damping, reduces oscillations, and improves the overall stability and response of the balancing system. By combining these three terms, the PID controller continuously adjusts the motor speed and direction to bring the robot back toward its upright reference position.



Figure 4: Robot prototype used for PID tuning

5.3. PID Parameters used in the prototype

The supplied implementation uses the PID gains $K_p = 100$, $K_i = 140$, and $K_d = 0.8$, with a setpoint of 176° . These values are the specific parameters documented in the actual project code and are used by the controller to determine the corrective motor response required to maintain the robot in

its balanced position. The proportional gain determines the immediate response to tilt error, the integral gain compensates for accumulated error, and the derivative gain provides damping by responding to the rate of change of the error. However, these PID gains are implementation-specific and should not be considered universal values for all self-balancing robots. The appropriate values depend on several factors, including the robot's mechanical geometry, total mass, motor characteristics, battery voltage, wheel diameter, center-of-mass position, sensor mounting, and overall mechanical construction. Therefore, these parameters generally require practical tuning and optimization when applied to a different robot design.

5.4. PID Tuning Procedure

The project follows a manual tuning method for determining suitable PID gains. Initially, all three gains are set to zero, i.e., $K_p = K_i = K_d = 0$, so that the controller produces no corrective response. The proportional gain (K_p) is then increased gradually until the robot begins to produce a meaningful corrective response to changes in its tilt angle. Once an appropriate proportional response is achieved, the integral gain (K_i) is introduced and gradually adjusted to reduce persistent or steady-state errors and improve the robot's ability to maintain its desired equilibrium position. Finally, the derivative gain (K_d) is increased carefully to reduce rapid oscillations and provide additional damping to the system. This step-by-step tuning approach allows the controller to achieve a balance between fast corrective response, low steady-state error, and stable operation without excessive oscillation. Because balancing occurs continuously, tuning should be performed carefully with the robot physically supported during initial tests. Because balancing occurs continuously, tuning should be performed carefully with the robot physically supported during initial tests.

Table 3. PID tuning procedure and objectives

Stage	Parameters	Objectives
1	$K_i = K_d = 0$	Establish proportional response
2	Increase K_p	Obtain fast correction
3	Add K_i	Reduce steady-state error
4	Add K_d	Reduce oscillation
5	Fine tune all	Improve stability

6. CONTROL ALGORITHM AND SYSTEM IMPLEMENTATION

6.1. Control Algorithm

The control algorithm of the self-balancing robot begins with the initialization of the Arduino UNO, followed by the initialization of the MPU6050 sensor through the I²C interface. The system first verifies proper sensor communication and then initializes the Digital Motion Processor (DMP) for orientation data processing. After this, the calibrated sensor offset values are applied, and the equilibrium setpoint is established according to the desired upright position of the robot. The PID parameters K_p , K_i , and K_d are then configured. During normal operation, the system continuously acquires the robot's orientation data from the MPU6050 and calculates the control error using $e(t) = \text{Setpoint} - \text{Input}$. The PID controller processes this error to generate the corrective control output according to $u = K_p e + K_i \int e \, dt + K_d (de/dt)$. Based on the magnitude and sign of the PID output, the Arduino determines the appropriate motor direction and generates corresponding PWM signals to control the speed of the left and right motors through the L298N driver. This entire process is repeated continuously at a high frequency, allowing the robot to detect changes in its tilt and immediately move the wheels in the required direction to restore and maintain its upright balance.

6.2. Software Implementation

The program uses the I2Cdev, PID_v1, and MPU6050 MotionApps20 libraries to interface with the MPU6050, implement the PID control algorithm, and process orientation data using the sensor's Digital Motion Processor (DMP). The implementation defines the MPU6050 sensor object and PID controller and configures the controller in automatic mode with a 10-ms sample time, allowing the system to update the control response at regular intervals. The PID output is limited to a range of -255 to +255, which corresponds to the typical 8-bit PWM range used in Arduino-based motor-control applications. The positive and negative values of the PID output determine the required motor direction, while the magnitude determines the PWM duty cycle and consequently the motor speed. This software configuration enables the Arduino to continuously process the robot's orientation, calculate the required corrective action, and control the motors to maintain the desired upright position.

6.3. Motor Control

The PID controller decides the direction and speed of the motors based on the robot's tilt. If the robot starts falling forward, the motors move forward to bring the wheels under the centre of mass and restore balance. Similarly, if it tilts backward, the motors rotate in the opposite direction to correct the tilt. The motor speed is continuously adjusted according to the PID output, so larger tilts produce stronger corrections. When the robot is close to the upright position, the motors stop or run at very low speed. This continuous adjustment helps the robot maintain its balance during operation.

6.4. Closed-Loop Working

The self-balancing robot works through a continuous closed-loop feedback system. The MPU6050 measures the robot's motion and provides its pitch angle, which is compared with the desired upright position to calculate the error. The PID controller processes this error and determines the required motor direction and speed. The Arduino then sends PWM and direction signals to the L298N motor driver, which controls the DC motors

to correct the robot's tilt. The updated position is measured again by the MPU6050, and the process repeats continuously. This allows the robot to quickly respond to disturbances and maintain its balance.

7. EXPERIMENTAL IMPLEMENTATION

The physical prototype consists of a two-wheel chassis with two geared DC motors, an Arduino UNO, MPU6050 IMU sensor, L298N motor driver, and battery supply mounted on a vertical frame. The motors provide both balancing and movement, while the Arduino processes the orientation data from the MPU6050 and generates corrective motor-control signals. The robot's balancing performance depends on the interaction between its mechanical and control systems. Factors such as centre-of-mass height, wheel radius, chassis weight, motor torque, friction, sensor mounting, and frame rigidity significantly affect stability and response. Hence, effective self-balancing requires proper mechanical construction along with accurate sensor processing and PID tuning.

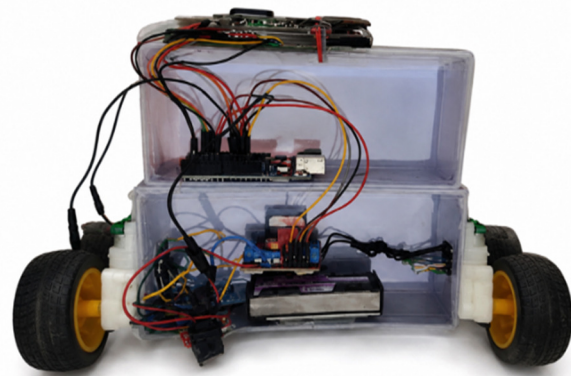


Figure 4: .Fabricated prototype of the self-balancing robot

8. RESULTS AND DISCUSSION

The prototype successfully demonstrates closed-loop balancing using the Arduino UNO,

MPU6050 IMU, and PID controller. The robot detects pitch-angle deviations through continuous sensor feedback and generates corrective motor commands to maintain its upright position. As sufficient numerical experimental data are unavailable, parameters such as settling time, overshoot, and steady-state error are not reported. The balancing performance depends mainly on proper PID tuning, sensor accuracy, motor response, battery voltage, and mechanical factors



Figure 5: Experimental prototype of the self-balancing robot

such as backlash and wheel-floor friction. Overall, stable operation requires proper coordination between the control system, sensors, motors, and mechanical structure.

9. CONCLUSION

The paper successfully demonstrates a low-cost two-wheeled self-balancing robot using an Arduino UNO, MPU6050, L298N motor driver, DC motors, and PID control. The system uses real-time feedback to detect tilt and generate corrective motor actions for maintaining balance. The prototype demonstrates the practical application of PID-based control in an unstable robotic system and provides a foundation for future improvements such as better sensor fusion, wheel encoders, and improved motor control.

ACKNOWLEDEMENT

The authors gratefully acknowledge the guidance of Dr. Ritwik Haldar, Department of Electronics & Communication Engineering, Haldia Institute of Technology, throughout this work, and the Department for providing the laboratory facilities used to build and test the prototype

REFERENCES

- [1] Arduino, "Arduino UNO R3 – Technical Specifications," Arduino Documentation.
- [2] J. Villacrés, M. Viscaíno, M. Herrera, and O. Camacho, "Controllers Comparison to Stabilize a Two-Wheeled Inverted Pendulum: PID, LQR and Sliding Mode Control," 2016.
- [3] R. Bakiya Lakshmi, S. M. Bashir, and G. S. Daniel, "Design, Simulation and Optimal Control of Two-Wheel Self-Balancing Robot Using LQR and PID," *Journal of Propulsion Technology*, vol. 45, no. 3, 2024.
- [4] A. Abdelgawad, T. Shohdy, and A. Nada, "Model- and Data-Based Control of Self-Balancing Robots: Practical Educational Approach with LabVIEW and Arduino," 2024.

- [5] G. B. Putra, F. Wahab, and T. A. Tamba, "Design and Implementation of Linear Quadratic Regulator Control for Two-Wheeled Self-Balancing Robot," *Bulletin of Electrical Engineering and Informatics*, vol. 14, no. 2.
- [6] N. H. Nam, "Control of Two-Wheeled Inverted Pendulum Robot Using Robust PI and LQR Controllers," *Journal of Military Science and Technology*, 2020.
- [7] G. S. Krishna, D. Sumith, and G. Akshay, "Epersist: A Self Balancing Robot Using PID Controller and Deep Reinforcement Learning," *arXiv*, 2022.