Robust Control of a Self-Balancing Robot: PID Implementation in MATLAB Simulink and Hardware under Disturbances

Pretom Das, Sabyasachi Monem, Minhaj Abedin Siam
Department of Electrical, Electronics & Communication Engineering
Military Institute of Science & Technology, Dhaka, Bangladesh
Email: pretomdas44@gmail.com, sabya4844@gmail.com, minhajabedinsiam144@gmail.com

***Abstract*—This project presents the design and development of a two-wheeled self-balancing robot that maintains its upright position by continuously correcting its tilt using a closed-loop control system. Compared to conventional four-wheeled robots, two-wheeled platforms are more compact and cost-effective, requiring fewer motors and components. However, they introduce a challenging control problem: maintaining balance on only two wheels. The proposed system architecture integrates an Arduino Nano microcontroller, DC stepper motors, and an MPU-based gyroscope and accelerometer sensor. A Proportional–Integral–Derivative (PID) controller, implemented through the Arduino IDE, governs the balancing mechanism. The paper provides a detailed description of the materials and components used, along with system flowcharts and closed-loop control diagrams to illustrate the control strategy.A step-by-step integration process, including wiring and final assembly, is also provided to demonstrate the practicality of the robot as a low-cost platform for study-level experimentation.**

***Index Terms*—Self-balancing robot, Arduino Uno, PID controller, DC stepper motor, gyroscope, accelerometer, two-wheeled robot, closed-loop control.**

I. Introduction

A self-balancing robot is essentially a practical realization of the inverted pendulum problem in control engineering. [1] In the case of a conventional pendulum, the center of gravity (COG) lies below the pivot point, allowing it to swing naturally around its equilibrium. However, in a self-balancing robot, the COG lies above the pivot point, making the system inherently unstable. Stability is achieved by continuously sensing the tilt angle and driving the wheels to minimize oscillations, thereby maintaining balance. By reducing the amplitude of this inverted pendulum, a highly stable robot can be developed that can stand upright and resist external disturbances. [2] [3]

Inverted pendulum–based systems have a wide range of applications in modern engineering. [4] For instance, Segways are two-wheeled personal transportation devices capable of carrying a rider for indoor or outdoor mobility. Similarly, rockets and missiles employ inverted pendulum stabilization principles to maintain orientation and trajectory during flight. More recently, hoverboards have gained popularity as compact self-balancing transporters without handlebars, while semi-autonomous toys using the same principle are now available in consumer markets, often responding to voice commands, gestures, or smartphone applications. [5]

In this work, a two-wheeled self-balancing robot is designed and implemented using Arduino Uno as the central microcontroller [6]. The Arduino is integrated with an MPU-6050 inertial measurement unit (IMU) that combines a gyroscope and accelerometer to measure tilt and angular velocity. A Proportional-Integral-Derivative (PID) controller processes this sensor data and generates corrective signals for the motors, ensuring real-time balance. Unlike four-wheeled robots, which are statically stable, a two-wheeled configuration reduces the number of actuators and components but introduces a more challenging dynamic control problem. This makes it an excellent study platform for understanding closed-loop control systems and embedded programming.

Several researchers have addressed the self-balancing robot [7]problem using different approaches, such as mathematical modeling, advanced control algorithms, and simulation tools like MATLAB. However, the focus of this project is on a low-cost, hardware-based design using easily available components. By providing a detailed step-by-step description of hardware connections, control implementation, and tuning of PID parameters, this paper aims to serve as both a practical guide and a foundation for future developments. [2] [8] Potential extensions include adaptive control, autonomous navigation, and integration with wireless communication modules for remote monitoring and control. [9] [10]

In this project, a disturbance force of 25 N was deliberately applied to the robot by pulling it from its balanced position. The system responded by dynamically adjusting its balance through PID tuning and successfully restored stability. This hardware implementation demonstrates the robustness of the control design against external disturbances and validates the effectiveness of the PID-based stabilization strategy.

II. Methodology

A. Mathematical Model and Control Equations

1) Inverted Pendulum Dynamics: The self-balancing robot can be modeled as an inverted pendulum on wheels. The

nonlinear dynamic equation of motion is:

$$I\ddot{\theta} + mgl\sin(\theta) = Fl\cos(\theta) \tag{1}$$

where: I = moment of inertia of the body about the wheel axis (kg·m^2), θ = tilt angle of the robot from vertical (rad), m = mass of the robot body (kg), l = distance from wheel axle to the center of mass (m), g = gravitational acceleration (9.81 m/s^2), F = control force applied by the wheels (N).

For small-angle approximation $(\sin\theta \approx \theta, \cos\theta \approx 1)$, the linearized equation becomes:

$$I\ddot{\theta} + mgl\theta = Fl \tag{2}$$

Here, $\ddot{\theta}$ is the angular acceleration of the pendulum.

2) Error Calculation: The error signal is defined as the difference between the desired reference angle and the actual measured tilt angle:

$$e(t) = \theta_{ref}(t) - \theta(t) \tag{3}$$

where $e(t)$ = error at time t, $\theta_{ref}(t)$ = reference tilt angle (typically 0° for upright balance), and $\theta(t)$ = measured tilt angle.

3) PID Control Law: A PID Controller (Proportional–Integral–Derivative Controller) is the most widely used feedback controller in control systems and automation. Its main job is to continuously calculate the difference (called error) between a desired setpoint (SP) and the actual process variable (PV), and then adjust the control input to minimize that error. The closed loop control system of the self-balancing robot is shown. The output shows the actual position of the robot. The MPU-6050 sensor senses the actual position of the robot. Sensor feedbacks the data to compare with the reference input. Reference input or the aim of the robot is to stay in the 90 degree vertical position with reference to ground, as shown in Fig.2. The comparison of output and reference input provides the tilt angle. This tilt is the error which needs to be reduced to zero. It requires a fast controller to handle this task of error reduction. PID controller has been used for this purpose. PID controller sends the command to the motor driver, and in result, it controls the motion of the stepper motors precisely.The balancing robot is modeled as an inverted pendulum. The PID controller regulates motor torque to minimize the error between the desired vertical orientation and the measured tilt angle: The PID controller generates the control input $u(t)$ as:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de(t)}{dt} \tag{4}$$

where $u(t)$ is the control signal applied to the motors, K_p = proportional gain, K_i = integral gain, K_d = derivative gain, and $e(t)$ = error signal.

- **Proportional term (K_p):** Provides a control action proportional to the present error. A higher K_p increases system responsiveness but may cause oscillations.
- **Integral term (K_i):** Eliminates steady-state error by accumulating past errors. However, too large a K_i can make the system unstable.
- **Derivative term (K_d):** Predicts future error by considering its rate of change. It improves stability and reduces overshoot but is sensitive to noise.

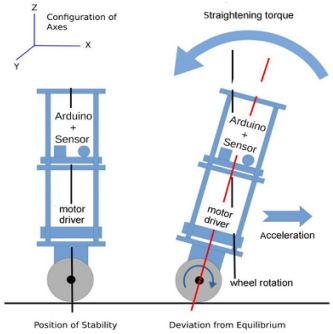


Fig. 1: Self-balancing positions of robot.

B. Software Implementation

1) Simulink Design: A block diagram of the control system was developed in MATLAB Simulink. The robot's dynamics were simulated using a simplified inverted pendulum model. MATLAB's PID Tuner was used to optimize controller gains for stability and minimal overshoot. Simscape Multibody visualization confirmed system behavior before hardware implementation.

TABLE I: Simulink Parameters

Parameter	Value
Total Weight	810 gm
Gravity	[0 -9.80665 0]
Disturbance Force	25 N
Time of Disturbances	3.5 s
Desired Angle	0°
Desired Distance	0 m

According to the parameters the circuit was designed in matlab simulink. while designing the circuit, at first no disturbance were given. in that case output of scope is smooth. the output tuned by PID and gradually output is equal to the set point. Then we added a disturbance, this can be any external force or obstacle that the balancing bot can face. In this case, the scope shows a spike , but it was settled by PID tuner.

C. Hardware Implementation

1) Component Description:

a) Arduino Uno: Arduino Uno serves as the central controller. It reads sensor data via I2C, executes PID logic, and outputs PWM signals to the motor driver.

b) MPU6050 (Accelerometer + Gyroscope): This IMU provides angular velocity and acceleration data. A complementary filter is applied to fuse accelerometer and gyroscope readings for stable tilt estimation.

c) L298N Motor Driver: The L298N dual H-bridge driver delivers bidirectional control of two DC motors using PWM inputs from the Arduino.

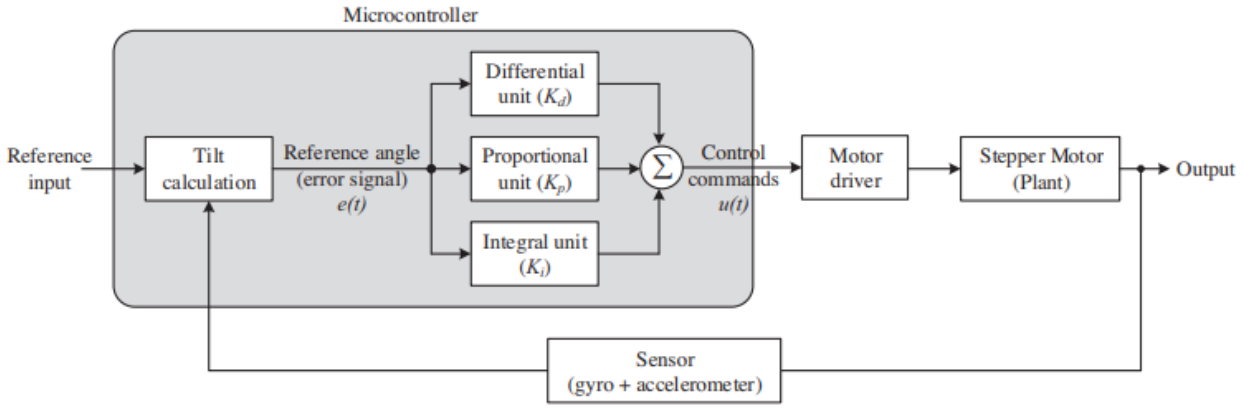


Fig. 2: The closed-loop control system of the self-balancing robot.

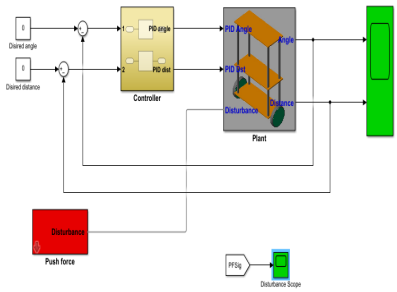


Fig. 3: Circuit Diagram implemented in simulink.

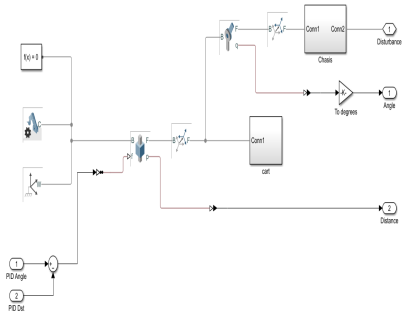


Fig. 4: Robot Modeling in Simulink.

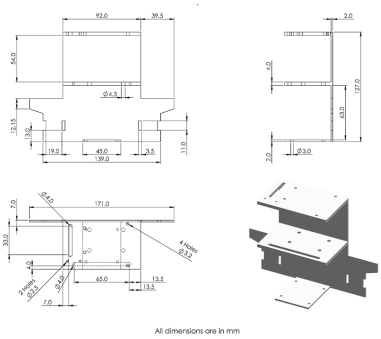


Fig. 5: Chassis Design in Solid Works.

D. Algorithm of microcontroller programming: Self-Balancing Robot Control

The Arduino IDE was used to program the Uno. The MPU6050 outputs were read via the Wire library, filtered to compute tilt angle, and fed into the PID algorithm. The resulting PWM duty cycles were applied to the L298N motor driver, adjusting wheel rotation to maintain balance.

1) Stepwise Procedure:

1. Initialization

- Configure I^{2}C communication at 400 kHz for MPU6050 and SH1106 OLED.
- Initialize OLED display and show startup message.
- Initialize MPU6050 sensor and load DMP firmware.
- Set calibrated gyro/accelerometer offsets.
- Enable DMP interrupt.
- Configure PID controller with K_p, K_i, K_d, sample time, and output limits.

2. Main Control Loop

- Wait for new sensor data from MPU6050 interrupt or FIFO buffer.
- If no new data:
  - Run PID with last known angle.
  - Drive motors with computed PID output.
  - Refresh OLED HUD.

3. Data Acquisition

- On interrupt, read DMP status and FIFO count.
- Handle FIFO overflow by resetting buffer.
- If valid, read one packet of motion data from FIFO.

4. Angle Estimation

- Extract quaternion data.
- Compute gravity vector.
- Derive yaw, pitch, roll (YPR) angles.
- Convert pitch angle (rad) to degrees and normalize.

5. PID Control

- Compare pitch angle (`input`) with reference setpoint ($\approx 181°$).
- Compute control signal using equation (4)

6. Motor Actuation

- Send PID output to L298N motor driver.
- Positive $\rightarrow$ forward, negative $\rightarrow$ backward.
- Enforce minimum absolute speed to overcome motor deadband.
- Apply correction factors for left/right motors.

7. Visual Feedback

- Update OLED HUD with:
  - Current pitch angle.
  - PID output and setpoint.
  - PID gains (K_p, K_i, K_d).
  - Balance bar visualization.

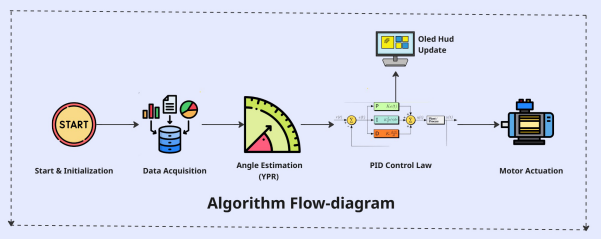


Fig. 6: Algorithm Flow Diagram

2) Circuit Implementation: The parameter that are used to implement the hardware setup:

TABLE II: Hardware Specifications

Parameter	Value
Weight	750 gm
Microcontroller	Arduino Uno
Rated Torque	0.35 kg·cm
Sensitivity of Gyroscope	(131, 65.5,32.8, or 16.4) [LSB/(°/s)]
Sensitivity of Accelerometer	(16384, 8192, 4096, or 2048) [LSB/sec]

According to the parameter, chassis was printed using a 3D printer, all the connections were done according to the figure shown in fig..

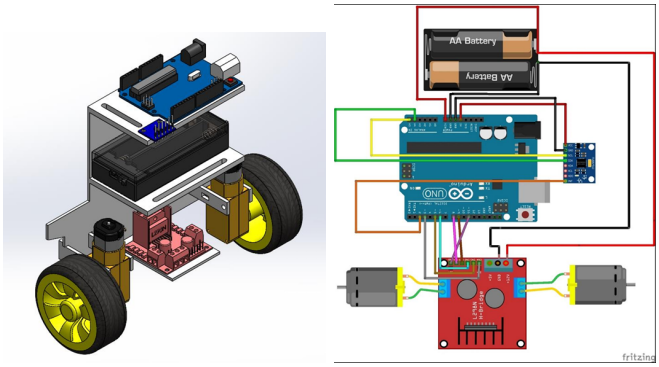
III. Results and Analysis

A. Simulink Output

The plant–controller model was evaluated in MATLAB/Simulink under two scenarios: (i) nominal operation without external disturbance and (ii) with an injected disturbance representing a push or bump to the chassis.

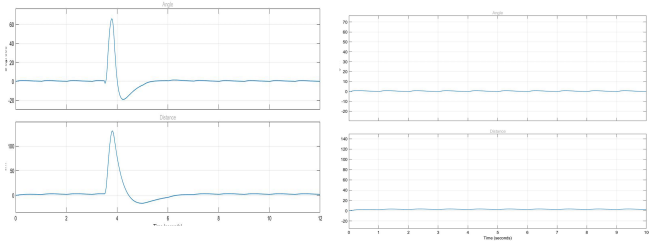
Nominal (no disturbance). With the reference (tilt angle) set to zero and no disturbance applied, the output exhibits a smooth transient and converges to the set-point. After PID tuning, the closed-loop response shows smooth output like the figure 5.

With disturbance. A transient disturbance was then injected to emulate an external force/obstacle acting on the robot body. As expected for an inverted-pendulum system, the output exhibits a spike at the disturbance instant due to the abrupt error increase. The tuned PID loop rejects this perturbation and restores the state to the set-point.



(a) 3D assembly (CAD). (b) Circuit / wiring diagram.

Fig. 7: Self-balancing robot: (a) mechanical layout, (b) electrical connections.



(a) With disturbance: spike then PID recovery. (b) Without disturbance: smooth convergence.

Fig. 8: Simulink scope outputs for the self-balancing robot: (a) injected push-force; (b) nominal operation.

Simulation results demonstrated successful stabilization of the inverted pendulum model. The tuned PID reduced overshoot and settling time, confirming theoretical feasibility. These results confirm that the proportional term provides immediate corrective torque, the derivative term damps the transient induced by the disturbance, and the integral term eliminates residual bias after recovery.

B. Hardware Output

The prototype robot achieved self-balancing behavior. Minor oscillations were observed due to sensor noise and motor backlash, but the system was able to recover from small disturbances. The experimental results aligned closely with simulation outputs in terms of rise time and steady-state accuracy.

IV. Conclusion

This work demonstrated a practical and low-cost approach to stabilizing a two-wheeled self-balancing robot using a PID controller. A MATLAB/Simulink model was used to tune the controller and verify feasibility, then the design was realized on Arduino Uno with an MPU6050 IMU and an L298N motor driver. Under nominal conditions the closed loop converged smoothly to the upright set-point; when a transient push-force disturbance was applied, the response showed a brief spike followed by rapid recovery with negligible steady-state error, validating disturbance rejection and confirming the

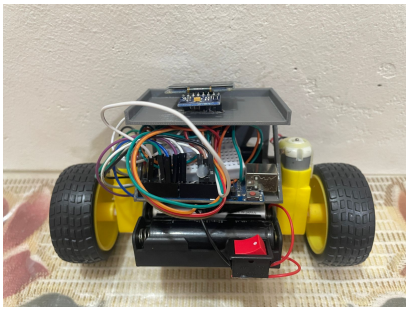


Fig. 9: Hardware visualisation.

roles of K_p (immediate correction), K_d (damping), and K_i (bias removal). The hardware results were consistent with simulation, indicating that the modeling and tuning process effectively transfers to the embedded platform.

Future work will focus on improving robustness and precision through enhanced sensor fusion (e.g., Kalman filtering), gain scheduling or adaptive PID, and exploration of state-space controllers (e.g., LQR) for multi-objective performance. Mechanical refinements (reduced backlash, balanced mass distribution) and power-stage upgrades are also expected to further reduce oscillations and improve recovery time.

REFERENCES

[1] A. S. Shekhawat and Y. Rohilla, "Design and control of two-wheeled self-balancing robot using arduino," in *2020 International Conference on Smart Electronics and Communication (ICOSEC)*. IEEE, 2020, pp. 1025–1030.

[2] M. A. Imtiaz, M. Naveed, N. Bibi, S. Aziz, and S. Z. H. Naqvi, "Control system design, analysis & implementation of two wheeled self balancing robot (twsbr)," in *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2018, pp. 431–437.

[3] Y.-J. Kim, J.-Y. Lee, and J.-J. Lee, "A balance control strategy of a walking biped robot in an externally applied force," in *2012 IEEE International Conference on Information and Automation*. IEEE, 2012, pp. 572–577.

[4] H. Jin, J. Hwang, and J. Lee, "A balancing control strategy for a one-wheel pendulum robot based on dynamic model decomposition: Simulations and experiments," *IEEE/ASME Transactions on Mechatronics*, vol. 16, no. 4, pp. 763–768, 2010.

[5] O. Jamil, M. Jamil, Y. Ayaz, and K. Ahmad, "Modeling, control of a two-wheeled self-balancing robot," in *2014 International Conference on Robotics and Emerging Allied Technologies in Engineering (iCREATE)*. IEEE, 2014, pp. 191–199.

[6] M. A. A. I. Samad, "Modeling, implementation, simulation and comparison of different control theories on a two wheel self balancing robot model in simulink."

[7] H.-S. Juang and K.-Y. Lum, "Design and control of a two-wheel self-balancing robot using the arduino microcontroller board," in *2013 10th IEEE International Conference on Control and Automation (ICCA)*. IEEE, 2013, pp. 634–639.

[8] W. An and Y. Li, "Simulation and control of a two-wheeled self-balancing robot," in *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2013, pp. 456–461.

[9] A. S. Wardoyo, S. Hendi, D. Sebayang, I. Hidayat, and A. Adriansyah, "An investigation on the application of fuzzy and pid algorithm in the two wheeled robot with self balancing system using microcontroller," in *2015 International Conference on Control, Automation and Robotics*. IEEE, 2015, pp. 64–68.

[10] A. Xi, T. W. Mudiyanselage, D. Tao, and C. Chen, "Balance control of a biped robot on a rotating platform based on efficient reinforcement learning," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 4, pp. 938–951, 2019.