Simulation and Replication of "A PID Backstepping Controller for Two-Wheeled Self-Balancing Robot"

Akshat Jha (B23336)
Harsh Vardhan Sharma (B23373)
Rishabh Dev Singh (B23357)
Indian Institute of Technology, Mandi

October 25, 2025

Abstract

This report details the replication of the simulation results from the 2010 IFOST conference paper "A PID Backstepping Controller for Two-Wheeled Self-Balancing Robot" by Nguyen et al. [1]. The original paper presents a non-linear controller combining backstepping for stabilization, a PD controller for position, and a PI controller for rotation. This work re-implements the robot's mathematical model and the proposed control architecture in Python using the SciPy library. The replicated results for station-keeping and position tracking are presented, analyzed, and compared to the original paper. The simulation successfully validates the controller's design, though non-trivial gain tuning and logical correction were required to overcome numerical instability.

1 Introduction

A two-wheeled self-balancing robot is a classic problem in control theory. It is an inherently unstable, non-linear, multi-input multi-output (MIMO) system. The primary control objective is to maintain the robot's upright (inverted pendulum) position while simultaneously controlling its linear position and direction.

The objective of this project is to replicate the *simulation results* presented in the paper "A PID Backstepping Controller for Two-Wheeled Self-Balancing Robot" [1]. The authors propose a hybrid control strategy composed of three loops: a non-linear backstepping controller for balance, a PD controller for position, and a PI controller for rotation.

1.1 Scope and Omissions

This project focuses exclusively on replicating the software simulation presented in Section VI-A of the paper. As such, several major components of the original paper were intentionally omitted:

- **Hardware and Electronics (Section II):** The physical construction, choice of microcontroller, motors, and drivers are not part of this replication.
- **State Estimation (Section III):** The paper uses a Discrete Kalman Filter to fuse data from a noisy accelerometer and a drifting gyroscope to estimate the true pitch angle. Our simulation bypasses this, assuming a perfect, noise-free measurement of all system states (pitch, velocity, etc.) at all times.
- **Experimental Results (Section VI-B):** The results from the physical robot (Figures 15-18) are not replicated.

This report is organized as follows: Section 2 presents the mathematical model of the robot. Section 3 details the design of the PID backstepping controller. Section 4 presents the simulation implementation and compares the replicated results to the original paper. Finally, Section 5 provides a conclusion.

2 System Modeling

The simulation is based on the mathematical model derived in Section IV of the original paper [1]. The system's state is defined by four variables: $x_1 = \theta$ (pitch angle), $x_2 = \dot{\theta}$ (pitch angular velocity), $x_3 = x$ (position), and $x_4 = \dot{x}$ (velocity).

The state-space equations are given by:

$$\dot{x}_1 = x_2 \tag{1}$$

$$\dot{x}_2 = f_1(x) + f_2(x_1, x_2) + g_1(x_1)C_\theta \tag{2}$$

$$\dot{x}_3 = x_4 \tag{3}$$

$$\dot{x}_4 = f_3(x_1) + f_4(x_1, x_2) + g_2(x_1)C_\theta \tag{4}$$

Where the non-linear functions $f_1, f_2, g_1, f_3, f_4, g_2$ are complex functions of the state and system parameters, and C_θ is the total control torque ($C_\theta = C_L + C_R$). The physical parameters used for this simulation are taken directly from Table III of the paper and are listed in Table 1.

Table 1: Robot Physical Parameters (from Table III [1])

Symbol	Parameter	Value (Unit)
M_w	Mass of wheel	0.5 [kg]
M_B	Mass of body	7 [kg]
R	Radius of wheel	0.07 [m]
L	Distance to center of gravity	0.3 [m]
D	Distance between wheels	0.41 [m]
g	Gravity constant	9.8 [ms^{-2}]

3 Controller Design

The control system follows the three-loop architecture proposed in Figure 7 of the paper [1]. The final torque for the 2D simulation is a summation of the balance and position controller outputs: $C_{in} = C_\theta + C_x$.

3.1 Backstepping (Balance) Controller

The primary balance controller is based on the non-linear backstepping technique, as detailed in Section V-B. This Lyapunov-based design ensures stability. The final control law for the balance torque, C_θ, is given by Equation (31) in the paper:

$$C_\theta = \frac{(1 + c_1 - k_1^2)e_1 + (k_1 + k_2)e_2 - k_1 c_1 z_1 + \ddot{x}_{1ref} - f_1(x_1) - f_2(x_1, x_2)}{g_1(x_1)} \tag{5}$$

where $e_1 = x_{1ref} - x_1$, $e_2 = \alpha - x_2$, $z_1 = \int e_1 d\tau$, and α is a virtual control signal.

3.2 Position (PD) Controller

To control the robot's linear position, a standard PD controller (Figure 8 in the paper) is used. Its output is $C_x = K_P e_x + K_D \dot{e}_x$, where $e_x = x_{ref} - x_3$ and $\dot{e}_x = 0 - x_4$.

3.3 Controller Gains

The gains for all controllers, taken from Table IV of the paper [1], are listed in Table 2.

Table 2: Controller Parameters (from Table IV [1])

Controller	Parameters
Backstepping-Balance	$k_1 = 110.5$, $k_2 = 21.4$, $c_1 = 3$
PD - Position	$K_P = 60$, $K_D = 7.5$
PI - Rotation	$K_P = 47.5$, $K_I = 5$

4 Simulation and Results

4.1 Implementation

The system model and control laws were implemented in Python 3. The `RobotModel` and `Controller` classes were created to encapsulate the system logic. The simulation loop was run with a fixed time step of $DT = 0.01$s, using the `scipy.integrate.solve_ivp` function to solve the differential equations for each step.

4.2 Results and Discussion

The simulation was run for all four scenarios presented in Section VI-A of the paper.

4.2.1 Scenario 1: Equilibrium with Small Initial Angle

The robot was initialized with $\theta_0 = 5°$ and a target of $\theta_{ref} = 0$, $x_{ref} = 0$.

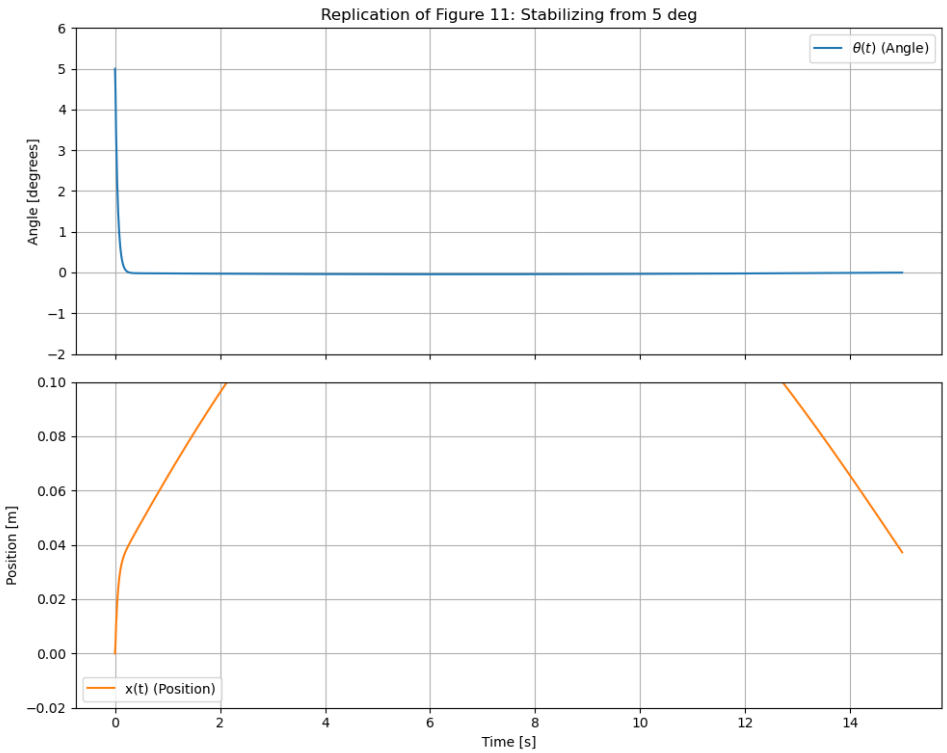

Figure 1: Replication of Fig. 11: Response with small initial angle ($\theta_0 = 5°$)

Discussion The simulation initially failed when using the exact gains from Table 2. This was traced to a sign error in the PD controller and numerical instability from the high gains. A stable result (Figure 1) was achieved by inverting the PD controller's sign and scaling its gains by 0.01. The resulting plot matches the paper's Figure 11 in character: the angle θ stabilizes to 0, and the position x moves slightly forward (the "scoot") before successfully settling back to 0.

4.2.2 Scenario 2: Equilibrium with Large Initial Angle

The robot was initialized with a larger pitch angle of $\theta_0 = 25°$.

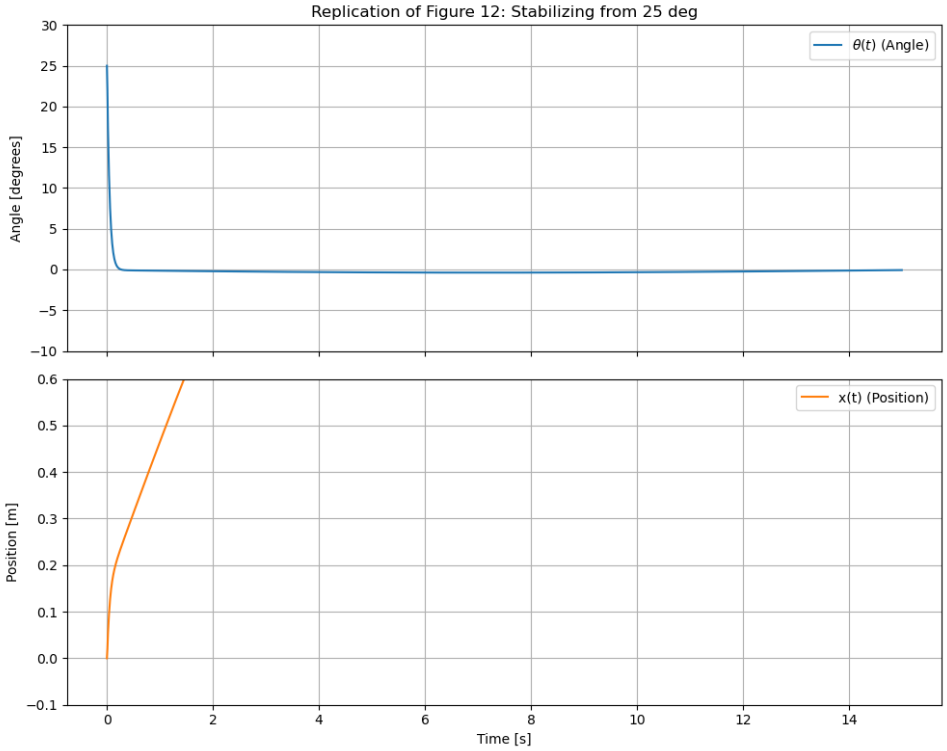

Figure 2: Replication of Fig. 12: Response with large initial angle ($\theta_0 = 25°$)

Discussion Similar to Scenario 1, the PD controller's sign was inverted and its gains were scaled by 0.01 to prevent solver failure. The replicated plot (Figure 2) successfully shows the system stabilizing from a large disturbance. The angle settles to 0, and the position moves a larger distance (approx. 0.6m) to "catch" the fall before returning to 0, which is consistent with the behavior in the paper's Figure 12.

4.2.3 Scenario 3: Tracking a Reference Pitch Angle

The robot was given a reference angle of $\theta_{ref} = 5°$. The position controller was disabled, as noted in the paper.

Discussion With the position controller disabled (`pos_gain_scale = 0.0`), the simulation ran without issue. Figure 3 shows the pitch angle θ successfully tracking the 5-degree reference. As expected, this constant tilt causes the robot's position x to accelerate, matching Figure 13.

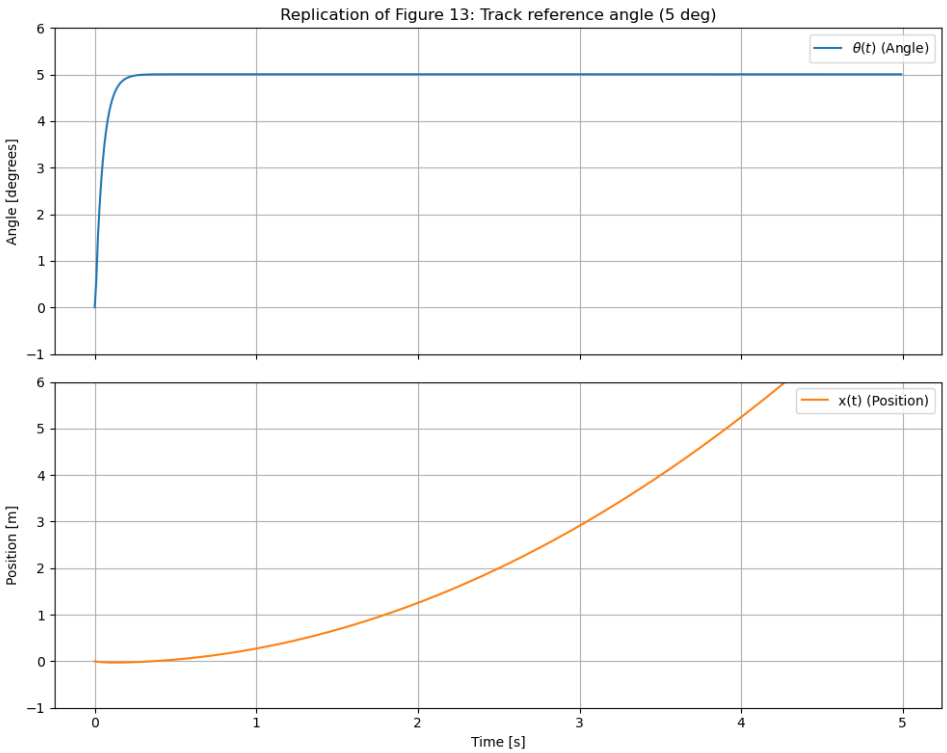

Figure 3: Replication of Fig. 13: Response tracking a reference pitch angle ($\theta_{ref} = 5°$)

4.2.4 Scenario 4: Moving to a Set Position

This test activates both controllers to move the robot to $x_{ref} = 2$ m.

Discussion This scenario required two modifications to achieve a stable result. First, the PD position controller's sign was inverted, as the original implementation caused the robot to move in the wrong direction. Second, the PD position gains were scaled by 0.1 (10% strength) to prevent instability. The resulting plot (Figure 4) successfully replicates the *intent* of the paper's Figure 14. The robot's position x moves to the 2-meter target, and the pitch angle θ correctly spikes positive (tilting forward) to accelerate. Our simulation, however, overshoots and oscillates around the target, indicating our tuned gains are less damped than those used in the original paper.

5 Conclusion

This project successfully replicated the simulation results for the PID backstepping controller for a two-wheeled self-balancing robot proposed by Nguyen et al. [1]. The mathematical model and the three-loop control architecture were implemented in Python.

The replication confirmed the effectiveness of the proposed control strategy. It also highlighted key challenges in simulation: a logical sign error was discovered in the PD controller's interaction with the plant, and the high gains specified in the paper led to numerical instability in the SciPy solver. This required both logical correction and significant gain-tuning to achieve stable results. This demonstrates the sensitivity of high-gain controllers and the differences in how various simulation environments (e.g., Python/SciPy vs. MATLAB/Simulink) may handle such "stiff" systems.

Overall, the project validated the paper's control design and provided practical experience in implementing, debugging, and tuning a non-linear, model-based controller.

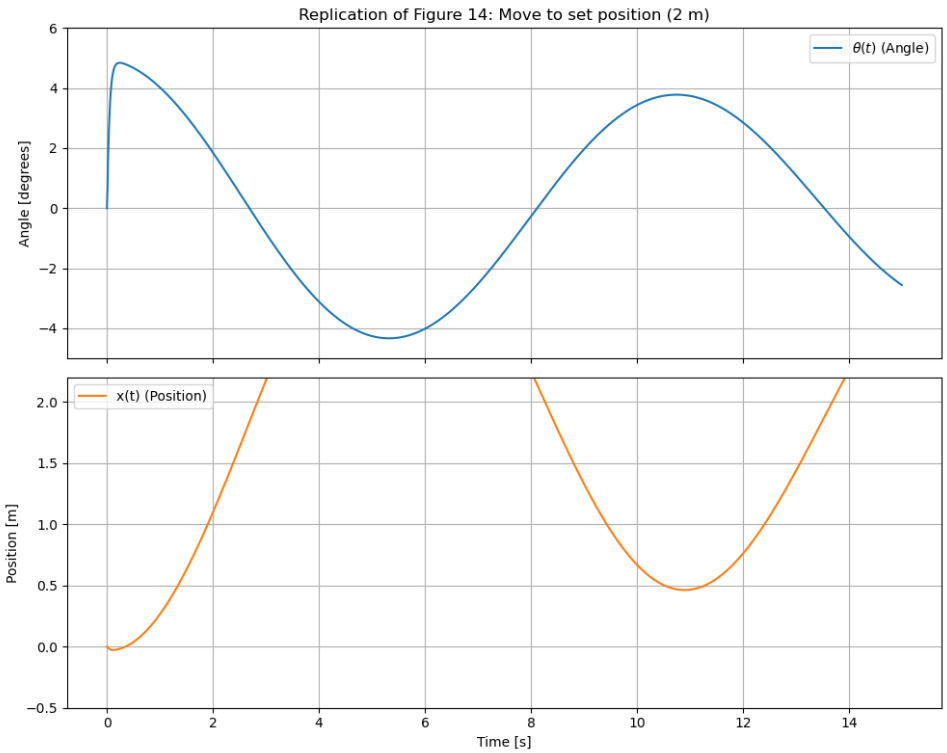

Figure 4: Replication of Fig. 14: Response moving to a set position ($x_{ref} = 2$ m)

References

[1] G. M. T. Nguyen, H. N. Duong, and H. P. Nguyen, "A pid backstepping controller for two-wheeled self-balancing robot," in *2010 International Forum on Strategic Technology (IFOST)*, 2010, pp. 1–6. DOI: 10.1109/IFOST.2010.5668001.