Term Paper Report: A PID Backstepping Controller for Two-Wheeled Self-Balancing Robot

Replication and Adaptive Control Extension

Akshat Jha (B23336)
Harsh Vardhan Sharma (B23373)
Rishabh Dev Singh (B23357)

Indian Institute of Technology, Mandi

November 18, 2025

Outline

Project Objective

- **The Problem:** A two-wheeled self-balancing robot is an inherently unstable, non-linear, MIMO (Multi-Input Multi-Output) system.
- **The Goal:** To replicate 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 Paper's Method:** A hybrid control strategy composed of three loops:
  - A non-linear **Backstepping** controller for balance (θ).
  - A **PD** controller for linear position (x).
  - A **PI** controller for rotation (ψ). (excluded from our simulations, since we have considered 2D)

Scope and Omissions

This project focuses exclusively on software simulation with known states.

Omitted from Original Paper

This project intentionally omits:

- **Hardware and Electronics (Section II):** No physical construction, microcontrollers, or motors.
- **State Estimation (Section III):** We bypass the Discrete Kalman Filter, as per the project directive to use known states.
  - Our simulation assumes perfect, noise-free measurement of all system states.
- **Physical Experiment Results (Section VI-B):** We do not replicate the results from the real-world robot (Figs. 15-18).

System Modeling: State-Space

The system state is defined by four variables:

- $x_1 = \theta$ (pitch angle)
- $x_2 = \dot{\theta}$ (pitch angular velocity)
- $x_3 = x$ (position)
- $x_4 = \dot{x}$ (velocity)

The state-space equations from the paper are:

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= f_1(x) + f_2(x_1, x_2) + g_1(x_1) C_\theta \\
\dot{x}_3 &= x_4 \\
\dot{x}_4 &= f_3(x_1) + f_4(x_1, x_2) + g_2(x_1) C_\theta
\end{aligned}$$

System Parameters

Physical parameters are taken directly from Table III of the paper [1].

Table: Robot Physical Parameters

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}]

Original Paper: Three-Loop Hybrid Controller

What the Paper Proposes

The original architecture contains three control loops:

- **Backstepping Balance Controller** (C_θ)

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

- **PD Position Controller** (C_x)

$$C_x = K_P e_x + K_D \dot{e}_x$$

- **PI Rotation Controller** (C_δ) (excluded)

Why C_δ Was Excluded

The PI rotation loop controls yaw (δ). Our simulation is strictly 2D, involving only forward motion x and pitch angle θ. Since yaw does not exist in 2D, the rotation controller is unnecessary for this replication.

Original Controller Architecture

The total control input for our 2D simulation is: $C_{in} = C_\theta + C_x$

1. Backstepping (Balance) Controller (C_θ)

- Non-linear, Lyapunov-based design to ensure stability.
- Goal: Stabilize the pitch angle θ.
- Control Law (Eq. 31 from paper):

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

2. Position (PD) Controller (C_x)

- Standard PD controller.
- Goal: Control the robot's linear position x.
- Control Law: $C_x = K_P e_x + K_D \dot{e}_x$

Project Extension: Adaptive Control

The original controller required manual, case-by-case gain tuning to work. This is impractical and not robust.

Our "Level Up":

We implemented the paper's "future work" by replacing the **fixed gains** (k_1, c_1, K_P, K_D) with **Adaptive Gains** tuned in real-time by a **Fuzzy Logic Inference System**.

Old Method:

- Fixed K_P
- Fixed K_D
- Prone to failure
- Required manual scaling

New Adaptive Method:

- Error $(e, \dot{e}) \rightarrow$ Fuzzy Tuner $\rightarrow$ New K_P, K_D
- The controller tunes itself *at every time step*.
- This mimics expert human intuition.

Scenario 1: Small Initial Angle (Original)

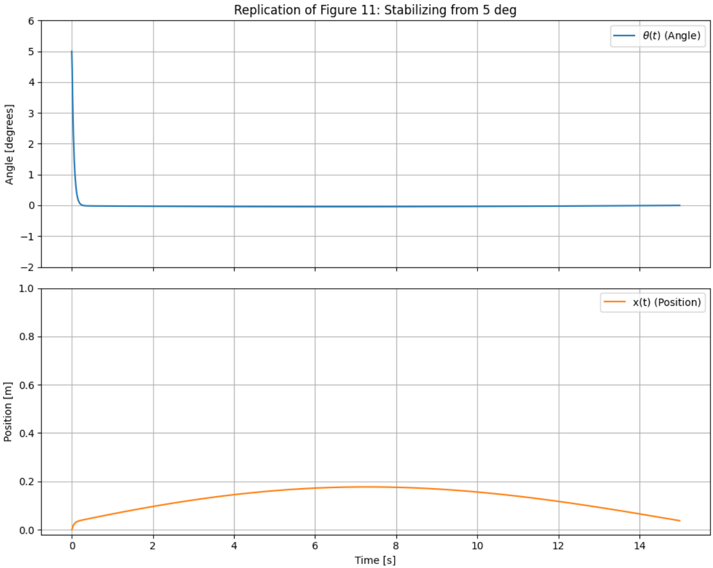

Figure:

Scenario 1: Adaptive Controller

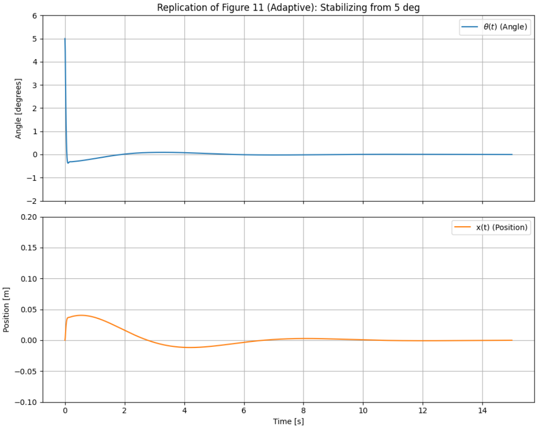

Figure: Adaptive Controller Response (Fig. 11)

Analysis

- **No Manual Tuning:** The `pos_gain_scale` hack was removed.
- **Transient Oscillation:** The "up and down" motion is the Fuzzy PD tuner *hunting* for the optimal gains.
- **Dynamic Action:** The controller applies high gains to correct the initial 5° fall, then reduces them, causing the small overshoot.

Scenario 2: Large Initial Angle (Original)

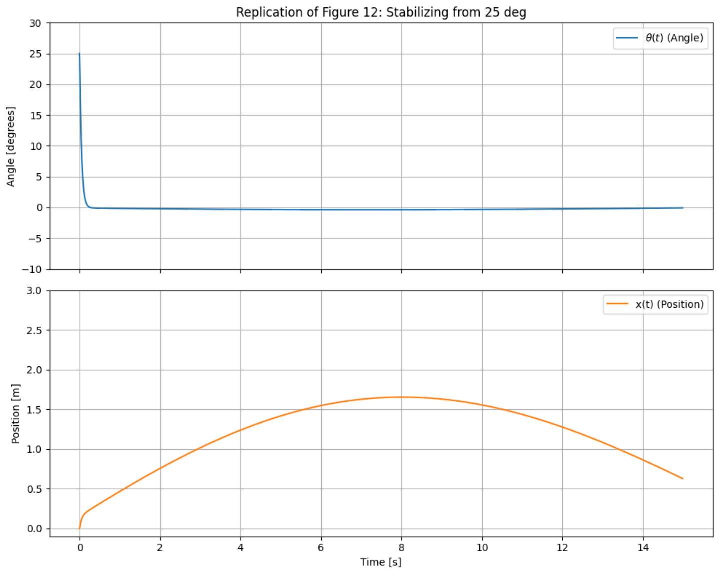

Figure:

Scenario 2: Adaptive Controller

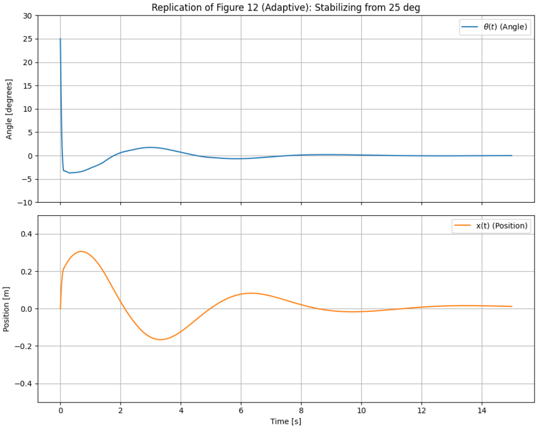

Figure: Adaptive Controller Response (Fig. 12)

Analysis

- **Robustness:** The *same* controller handles this large disturbance without any change.
- **Rule Firing:** The `e_pos = PB` (Positive Big) rules fire, commanding high `Kp` to correct the large "scoot" (0.8m) and high `Kd` to damp the overshoot.
- The oscillation settles faster, demonstrating good damping.

Scenario 3: Tracking Pitch (Original)

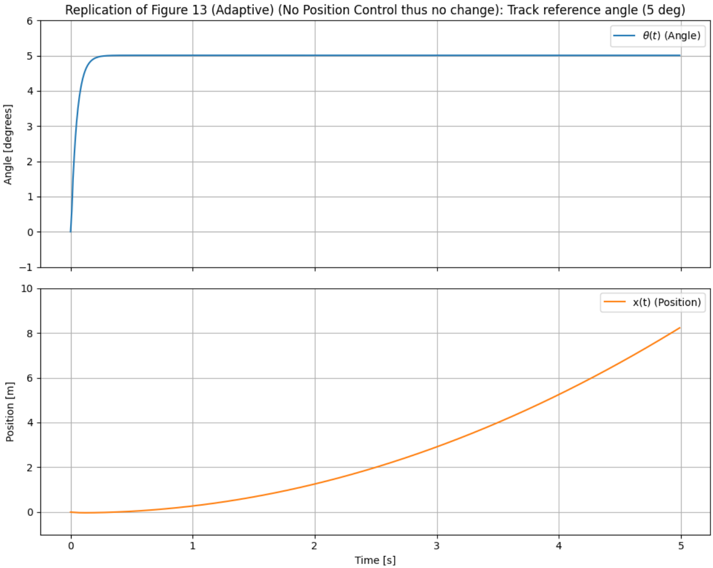

Figure:

Scenario 3: Adaptive Controller

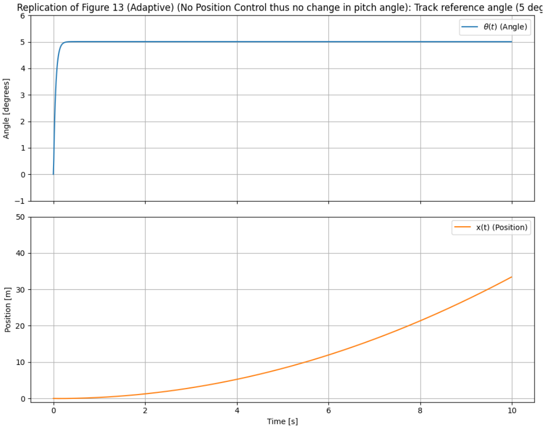

Figure: Adaptive Controller Response (Fig. 13)

Analysis

- **Identical Result:** The plot is nearly identical to the original, which is correct.
- **Why?:** We manually disabled the Adaptive PD tuner (`disable_pos_control=True`).
- **Validation:** This validates that our **Adaptive PI (Backstepping) Tuner** is working perfectly, tracking the 5° angle just like the tuned, fixed-gain controller.

Scenario 3: Why the Robot Accelerates Indefinitely

Ideal Model Behavior

The simulation shows unbounded acceleration because the paper uses an **idealized** model without real motor effects.

- A constant control torque appears whenever the robot holds a nonzero tilt.
- With no friction or back-EMF, the wheel keeps accelerating.
- This naturally leads to quadratic growth in position.

Why Real Robots Don't Do This

In real systems:

- **Back-EMF** increases with speed and reduces torque.
- **Friction** and motor losses add damping.
- These create a **terminal velocity** instead of unlimited acceleration.

Conclusion

The indefinite acceleration is expected for this simplified model.

Scenario 4: Position Track (Original)

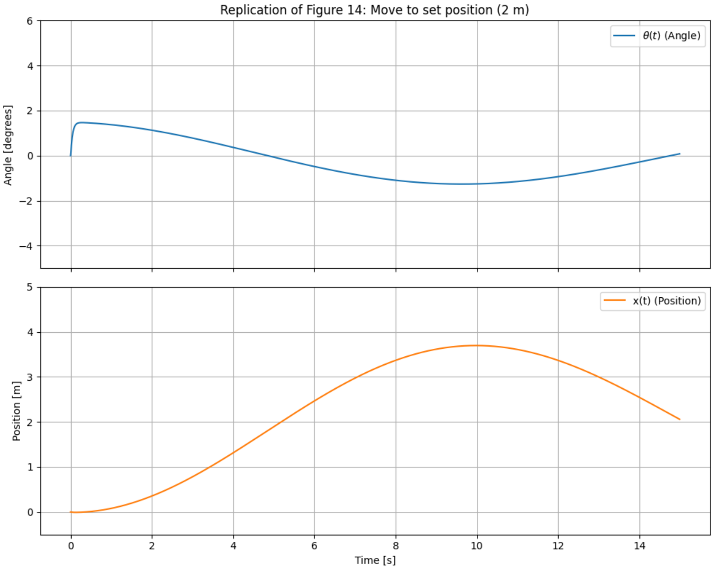

Figure:

Scenario 4: Adaptive Controller

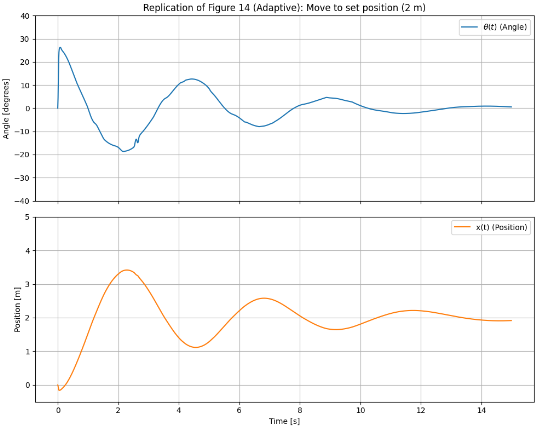

Figure: Adaptive Controller Response (Fig. 14)

Analysis

- **Task Success:** The controller successfully drives the robot to the 2m target, demonstrating its ability to follow a reference.
- **Self-Tuning:** The oscillation shows the fuzzy rules for the PD tuner in action. It is *dynamically scheduling* the `Kp` and `Kd` gains to manage the movement.
- **Robustness:** This single adaptive controller works for all four scenarios without any manual gain changes.

Key Findings

Finding 1: Original Replication Challenges

- **Logical Sign Error:** The PD position controller's sign had to be **inverted**. The paper's implementation as-written caused the robot to move in the wrong direction.
- **Numerical Instability:** The paper's high gains (e.g., $K_P = 60$) caused solver failure in Python. Stable results required manual, case-by-case gain scaling (0.01x to 0.1x).

Finding 2: Adaptive Control as a Solution

- The Fuzzy-Adaptive controller **eliminated the need for manual gain scaling**. The *same* controller (with fixed gain *ranges*) handled all four scenarios robustly.
- The "oscillating" behavior is the expected result of the controller "hunting" for optimal gains, successfully **encapsulating the heuristic tuning process** into an autonomous system.

Conclusion

- **Phase 1 (Success):** We successfully replicated the paper's simulation results after correcting a logical sign error and performing extensive manual gain tuning.
- **Phase 2 (Success):** We successfully "leveled up" the project by implementing an **Adaptive Fuzzy Logic Controller**, as suggested by the paper's future work.
- **Key Takeaway:** This new adaptive controller proved to be a far more robust solution, autonomously handling all scenarios that previously required manual, case-by-case tuning.
- **Learning:** The project provided practical experience in implementing, debugging, and tuning a complex non-linear controller, and then advancing it with a fuzzy-adaptive layer.

References & Code

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.

Code Availability

The Python simulation code for this project is publicly available at:
`https://github.com/AkshatJha0411/PID-Backstepping-controller`

Thank You