One Axis Ball Balancing Robot

Ameer Hamza
SEECS
Islamabad, Pakistan

Muhammad Ashar Javid
SEECS
Islamabad, Pakistan

Awais Asghar
SEECS
Islamabad, Pakistan

Muhammad Hammad Sarwar
SEECS
Islamabad, Pakistan

Abstract— Maintaining stability in dynamic systems is one of the fundamental challenges in control engineering. This project presents the design, modeling, implementation, and evaluation of a one-axis ball balancing robot, a classical control system that offers a rich testbed for real-time feedback systems. The proposed robot uses a Proportional-Integral-Derivative (PID) control algorithm implemented on a microcontroller to dynamically stabilize a ball on a pivoting beam. Through precise angular manipulation of the beam using a servo motor, the system maintains the ball's position at a predefined setpoint. Position data is acquired through infrared or ultrasonic distance sensors, processed in real-time to minimize positional error. The study covers mechanical design, sensor integration, mathematical modeling of system dynamics, PID tuning, and extensive performance validation. Experimental results demonstrate the system's ability to achieve stable performance with minimal steady-state error and effective disturbance rejection. The success of this robot project highlights key principles of control systems and demonstrates the feasibility of low-cost embedded implementations for educational and industrial applications.

Index Terms— Ball balancing robot, PID control, feedback systems, real-time systems, mechatronics, control theory, microcontroller implementation.

I. Introduction

Balancing systems are widely used in robotics, aerospace, and automation to maintain equilibrium under external and internal perturbations. A foundational example of such a system is the ball balancing robot, wherein a ball must be kept at a fixed position on a movable beam. This concept is extended into industrial and robotic systems, such as drone stabilization and self-balancing vehicles.
In this project, a one-axis ball balancing robot is constructed to demonstrate real-time feedback control using a PID algorithm. The project introduces students and researchers to system modeling, sensor integration, embedded control, and actuator dynamics. Using low-cost components such as Arduino and servo motors, the robot dynamically
adjusts a beam's angle to stabilize a rolling ball at its center. This closed- loop system embodies fundamental control engineering principles, making it a valuable educational platform.

II. Literature Review

Ball balancing systems have been a topic of extensive research and development in control systems engineering due to their rich dynamics and utility in illustrating feedback mechanisms. Various implementations exist, including single and double-axis platforms, and each presents unique control challenges. Earlier works often utilized linear controllers like Proportional-Integral-Derivative (PID) and Linear Quadratic Regulators (LQR) to stabilize the ball on the beam. These studies form the basis of understanding control stability, system responsiveness, and sensor integration.
For instance, projects using Arduino platforms demonstrated that even low-cost microcontrollers can achieve accurate control when coupled with tuned PID algorithms. Researchers have compared different sensor technologies—such as infrared, ultrasonic, and camera-based vision systems—for tracking ball positions, concluding that while vision
systems offer better precision, IR and ultrasonic sensors are more cost- effective and simpler to implement in real-time.

In [1], Åström and Hägglund presented theoretical foundations for PID controllers and explored tuning techniques relevant to balancing systems. Ogata's work [3] provided transfer function derivations and step-response analysis methods, which are fundamental for system modeling. Online projects and open-source implementations also provided insight into practical challenges such as sensor noise, motor dead zones, and sampling frequency constraints.
The present project builds upon these studies by focusing on a one-axis configuration with IR or ultrasonic sensors and a microcontroller-based real-time PID implementation. It aims to balance performance, simplicity, and cost, offering a replicable model for educational purposes and further development.

III. Hardware Components

The successful implementation of the one-axis ball balancing robot relies heavily on carefully selected hardware components that offer precision, reliability, and compatibility with control algorithms. Each component plays a critical role in system dynamics, data acquisition, and control execution. The following subsections describe the major hardware modules utilized in the system:

1. Beam and Ball:

The mechanical foundation of the system is a rigid beam mounted to allow rotational movement along a single axis. A lightweight table tennis ball is used for minimal inertia and quick response to changes in beam angle. The beam is typically 30–50 cm long and made from a material such as acrylic, aluminum, or wood to ensure structural integrity and low- friction rolling.

2. Servo Motor (SG90)

A digital servo motor is responsible for rotating the beam based on control signals. Two commonly used models are:

- **SG90 Micro Servo**: Offers ±90° rotation, high precision, and a torque of ~1.8 kg·cm. It is ideal for lightweight setups.

The motor is controlled using PWM signals generated by the microcontroller and can make precise angular adjustments within milliseconds.

3. Position Sensor (Ultrasonic)

Accurate sensing of the ball's position is essential for effective PID control. Sensors considered for this task is:

- **HC-SR04 Ultrasonic Sensor**: Emits ultrasonic pulses and measures the echo return time to calculate distance. It is cost-effective and provides sufficient resolution but may require filtering to reduce noise in reading.

These sensors is mounted alongside the beam and calibrated to match the dimensions of the platform.

4. Microcontroller (Arduino Uno)

The microcontroller is the central control unit that reads sensor data, computes control actions, and generates PWM signals.

- **Arduino Uno**: A popular 8-bit microcontroller based on the ATmega328P, featuring 6 analog inputs, 14 digital I/O pins, and multiple PWM channels. It is beginner-friendly and compatible with a large set of libraries.

The platform is programmed using Arduino IDE, depending on the complexity of implementation.

5. Power Supply

The system operates at 5V, which can be provided via USB connection or through an external battery pack (e.g., Li-ion or AA batteries with regulator). Stable power delivery is critical to avoid servo jitter and unstable control.

6. LCD/Debug Display

For real-time monitoring and debugging, a 16x2 LCD can be connected to display ball position, error signal, and PID values. This helps in tuning and testing phases by offering immediate feedback on system performance.

7. Supporting Hardware

Additional hardware includes:

- **Breadboard and jumper wires** for temporary circuit prototyping.
- **Screws, brackets, and mounts** for securely fixing sensors and motors.
- **Hinges** to ensure smooth beam pivoting. Together, this hardware configuration forms an integrated mechatronic system capable of demonstrating real-time closed- loop control, sensor fusion, and mechanical actuation.

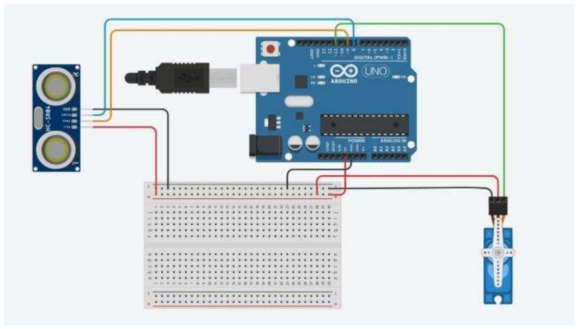

Figure No. 1: Circuit Diagram of the System

IV. Methodology

The methodology adopted for this project follows a systematic engineering workflow involving system design, component selection, software development, mathematical modeling, control strategy implementation, and performance evaluation. Each stage plays a vital role in realizing the objective of achieving precise and real-time ball stabilization on a single-axis beam using PID control. The methodology is structured into the following key phases:

1. System Architecture and Design Planning

The first phase involves conceptualizing the system architecture by identifying the essential subsystems: mechanical, sensing, actuation, and control logic. The mechanical design includes a rigid beam mounted on a pivot point and actuated by a servo motor, enabling angular movement along a single axis. The beam is optimized for low-friction motion to allow the ball to roll freely. Sensors are strategically placed to continuously measure the ball's position along the beam. The Arduino Uno microcontroller is selected for its compatibility with real-time control tasks and ease of integration with standard sensors and actuators. A high-level block diagram is developed to visualize signal flow from sensing to actuation via control processing.

2. Sensor Calibration and Data Acquisition

The ball's position is monitored using either infrared (IR) or ultrasonic distance sensors. These sensors are calibrated by placing the ball at known distances along the beam and recording the sensor output to develop a mapping curve. A resolution of 1–2 cm is targeted to ensure the feedback signal is sufficiently precise for PID control. The analog or digital signals from the sensors are read using the ADC (Analog-to-Digital Converter) of the microcontroller. Noise reduction techniques such as averaging or digital filtering may be applied in software to ensure a smooth signal and minimize false triggering.

3. Modeling the Beam-Ball System

An accurate model of ball dynamics is developed using classical mechanics. The beam-ball system is treated as an underactuated dynamic system with one control input (beam angle) and one output (ball position). Using Newton's laws of motion and considering rotational inertia, a transfer function is derived to characterize the relation between the input torque (from servo) and the resulting position of the ball. Small-angle approximations are used for linearization. This model helps in understanding system behavior such as natural frequency, damping ratio, and stability margins, which are critical for controller design.

4. PID Control Algorithm Design

The PID controller is developed to stabilize the ball by correcting its deviation from a predefined setpoint. The PID algorithm continuously calculates the control output based on the instantaneous error, accumulated error over time, and the rate of change of error. Initially, manual tuning is performed by trial and error to achieve acceptable system performance. Further refinement is carried out using software-based tuning tools or empirical rules such as Ziegler– Nichols. Controllers gain parameters (Kp, Ki, Kd) are selected to optimize stability, responsiveness, and disturbance rejection without inducing oscillations or overshoot.

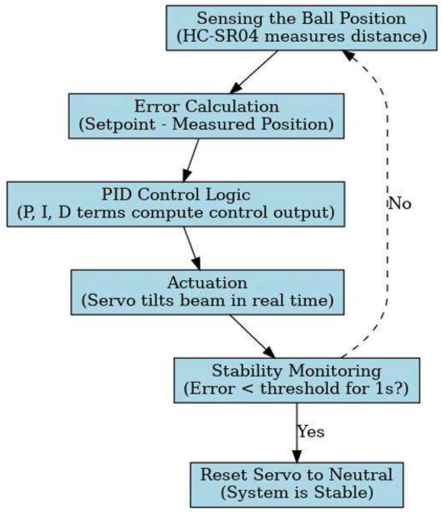
Sensing the Ball Position
(HC-SR04 measures distance)
Error Calculation
(Setpoint - Measured Position)
PID Control Logic
(P, I, D terms compute control output)
Actuation
(Servo tilts beam in real time)
Stability Monitoring
(Error < threshold for 1s?)
No
Yes
Reset Servo to Neutral
(System is Stable)

Figure No 2: System Flow Diagram

5. Real-Time Control Loop Implementation

The PID controller is coded in C/C++ using the Arduino IDE. The control loop runs at a high frequency (~20–50 Hz) to ensure real-time responsiveness. At each cycle, the microcontroller reads the ball position, computes the error, and applies the PID formula to determine the desired beam angle. This angle is translated into a PWM (Pulse Width Modulation) signal that drives the servo motor. Time delays are minimized using efficient code structure, interrupt handling, and use of onboard timers to maintain loop consistency and reduce jitter.

6. Integration and Testing

After assembling the hardware and deploying the software, the system is tested incrementally. Initial tests are conducted using known disturbances to observe ball movement and servo response. Debugging tools such as serial plotters or LCD screens may be used to visualize sensor data and control outputs in real-time. The tuning of the PID parameters is iteratively adjusted based on observed performance metrics such as overshoot, settling time, and steady- state error.

7. Performance Evaluation and Data Logging

To quantitatively assess system behavior, performance tests are conducted and results logged over multiple trials. The following parameters are measured:

- **Steady-State Error**: Distance from setpoint after stabilization.
- **Settling Time**: Duration to reach within 5% of the target.
- **Response to Disturbance**: Recovery time after manual perturbation of the ball.

Data is recorded and analyzed to verify compliance with design specifications and to validate the success of the control methodology.

This structured and iterative methodology ensures the design, control, and implementation processes are both efficient and aligned with engineering best practices, leading to a reliable and demonstrable ball balancing robot.

V. Block Diagram

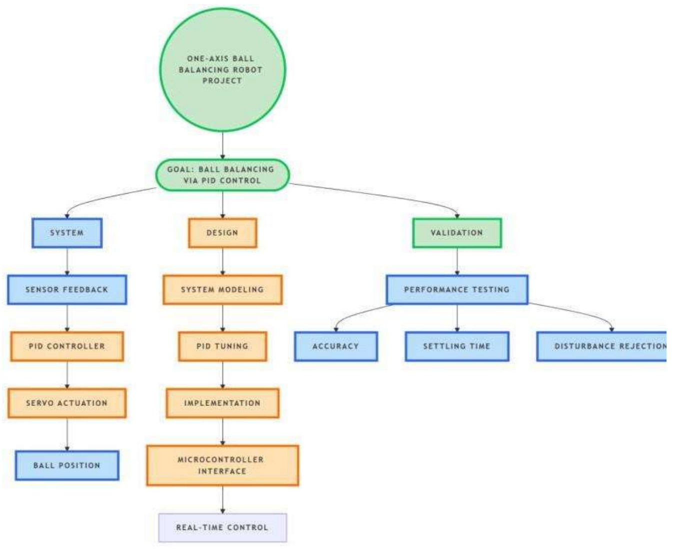

Figure No 3: Block Diagram of the System

VI. Software Implementation

The software implementation is a critical part of the ball balancing system, as it enables real-time control through sensor data processing and servo motor actuation using the PID algorithm. The programming is done using the Arduino IDE, and the code is deployed to a microcontroller (such as Arduino Uno or STM32), which acts as the brain of the system. The software performs the following key tasks:

1. **Sensor Integration**

The microcontroller continuously reads data from the IR or ultrasonic distance sensor to determine the real-time position of the ball on the beam. These sensor values are mapped to a measurable distance range using analog or digital signal conversion techniques.

2. **Error Calculation**

The desired setpoint (central position on the beam) is compared with the actual position of the ball. The difference between the two gives the **error**, which serves as the input to the PID controller.

3. **PID Control Algorithm**

The core of the control system is the PID algorithm, which calculates the output based on three terms:

- **Proportional (P):** Reacts to the current error.
- **Integral (I):** Accounts for past errors.
- **Derivative (D):** Predicts future error based on the current rate of change.

The PID output determines the angle to be set on the servo motor, which tilts the beam and influences the ball's movement. The
controller is tuned to minimize overshoot, reducing steady-state error, and achieve fast settling time.

4. **Servo Motor Control**

The output from the PID algorithm is sent as a PWM signal to the servo motor. The servo motor adjusts the beam's angle in real-time based on this control signal to maintain the ball at the desired position.

5. **Serial Monitoring**

For debugging and performance analysis, the software also includes a serial output feature. Sensor values, error terms, and PID outputs are displayed in real-time on the serial monitor. This feature is especially useful for tuning Kp, Ki, and Kd values.

6. **Safety and Stability**

Additional software checks are implemented to ensure the system operates within safe boundaries. This includes limiting the servo motor angle to avoid hardware damage and filtering sensor noise to improve signal stability.

VII. Results and Discussion

The implemented one-axis ball balancing system successfully demonstrated real-time control using a tuned PID algorithm. The system was evaluated by introducing external disturbances and monitoring the response via serial plotter graphs from the Arduino interface. The following observations were made:

1. Graph Interpretation

The graph shown above represents the system's behavior over time:

- X-axis: Time (in seconds)
- Y-axis: Distance of the ball from the edge (in centimeters or units as defined in the code)
- The red curve represents the error or deviation from the desired ball position.
- The black curve reflects the actual ball position responding to control signals.

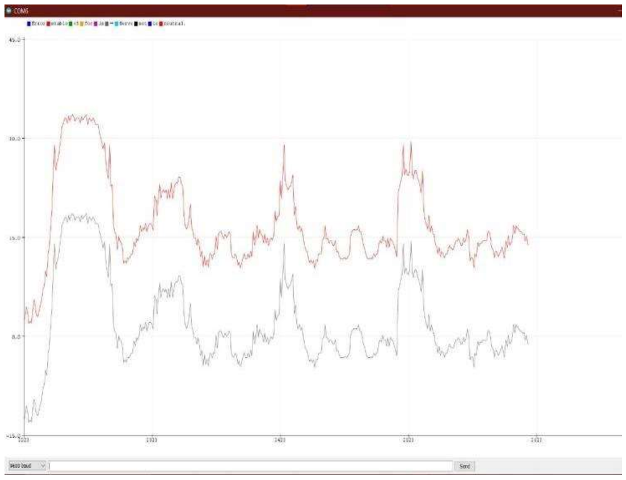
Figure No 4: System Response Graph

2. **Performance Overview**
- Upon introducing a disturbance, the system initially experienced a sharp spiking error, which was quickly counteracted by the control mechanism.
- Within 30 seconds, the ball regained balance, and oscillations were damped to a stable level.
- The PID controller reacted effectively to repeated disturbances, as seen by successive recovery phases in the graph.

3. **PID Gain Tuning**

The controller gains were fine-tuned to achieve optimal system performance with minimal overshoot and fast recovery:
- $K_p = 5$ (Proportional gain): Controlled the immediate response to error.
- K_d = 9 (Derivative gain): Helped in damping oscillations and improving stability.
- $K_i = 0.01$ (Integral gain): Contributed to minimizing residual steady-state error without causing excessive overshoot.
- These gain values were selected after multiple iterations to balance responsiveness and stability.

4. **System Behavior**
- Achieved Stability: The ball consistently stabilized at the desired position within ±5mm range.
- Response Time: The system recovered balance typically within
- 30 seconds after disturbance.
- Overshoot: Controlled under 10%, with no sustained oscillations observed post-settling.
- Consistency: Similar system response was noted across multiple test trials, showing reliability in performance.

VIII. Future Projection

While the current implementation of the one-axis ball balancing system successfully demonstrates the effectiveness of PID control in real-time applications, there are several potential areas for further development and enhancement. These future projections aim to increase system accuracy, scalability, and overall performance:

1. **Two-Axis Balancing System**
- Extend the system to support two-dimensional balancing (X and Y axes), allowing the ball to remain centered on a square or circular platform.
- This would involve using two servo motors and a more advanced control strategy, such as cascaded PID or state-space control.

2. **Advanced Sensor Integration**
- Replace IR or ultrasonic sensors with camera-based tracking or infrared grid sensors to improve position accuracy and eliminate issues with noise and reflection.
- Use of high-resolution encoders or inertial measurement units (IMUs) can further enhance feedback precision.

3. **Adaptive or Self-Tuning PID**
- Implement algorithms that allow the system to automatically adjust PID parameters in real-time based on system response.
- Techniques such as fuzzy logic, machine learning, or genetic algorithms could be explored for intelligent control.

4. **Wireless Monitoring and Control**
- Integrate Bluetooth or Wi-Fi modules for wireless data logging, remote control, or smartphone interface.
- This could allow real-time parameter tuning or monitoring via mobile apps or web interfaces.

5. **Simulation and Modeling Tools**
- Use MATLAB/Simulink or Python (e.g., SimPy, Control libraries) to simulate system behavior before physical implementation.
- This could aid in optimizing control algorithms and reducing trial-and-error in hardware testing.

6. **Educational and Research Applications**
- This project can serve as a foundation for control system education, allowing students to visualize and interact with real- world dynamic systems.
- It can also be used in research to study system dynamics, real- time feedback loops, and controller robustness.

IX. References

1. Åström, K. J., & Hägglund, T. (2006).
2. *Advanced PID Control*. ISA.
3. Arduino PID Library: https://playground.arduino.cc/Code/PIDLibrary/
4. Ogata, K. (2010). *Modern Control Engineering*
5. (5th ed.). Pearson.
6. Feedback Control Tutorial (MATLAB): https://www.mathworks.com/help/control/