(Established under Karnataka Act No. 16 of 2013)

100-ft Ring Road, Bengaluru – 560 085, Karnataka, India

Robotic Systems

UE22EC342BC2

Self-balancing-robot

BY:

K NARESH KRISHNA PES1UG22EC901
MRIDUL DINESH PES1UG22EC903

Introduction

The self-balancing robot is a two-wheeled, inverted pendulum-style system designed to maintain its upright position using feedback control. This project explores the design and implementation of such a robot through **two parallel approaches**: a **hardware prototype** built with Arduino, sensors, and motors, and a **software simulation** developed using ROS 2 Jazzy and the Gazebo simulator. Both implementations aim to demonstrate the principles of dynamic stability, sensor integration, and closed-loop control. The hardware version offers hands-on insight into real-world electronics and sensor noise, while the software version enables safe and flexible experimentation with control algorithms and robot dynamics in a virtual environment. Together, they provide a comprehensive learning experience in robotics systems design.

Implementation of the Project

The self-balancing robot project was implemented through **two distinct approaches**: a **hardware prototype** using physical electronic components and an **independent software simulation** using ROS 2 and Gazebo. Both approaches aim to achieve the same objective — maintaining balance through continuous feedback and control — but each highlights different aspects of robotics system development.

Hardware Implementation (Arduino-Based)

The hardware version involves building a real-world self-balancing robot using affordable and accessible components. The setup includes:

- **Microcontroller**: Arduino UNO
- **Sensor**: MPU6050 (3-axis gyroscope + 3-axis accelerometer)
- **Motor Driver**: L298N H-bridge
- **Actuators**: Two DC motors with wheels
- **Power Supply**: Battery pack

Process:

1. **Sensor Integration**: The MPU6050 sensor is connected to the Arduino using the I2C protocol. It provides raw data on the robot's orientation, including tilt angle and angular velocity.

2. **Data Reading and Filtering**: The Arduino reads the accelerometer and gyroscope values using the Adafruit MPU6050 library. These values can be further processed using complementary or Kalman filters (if desired) to get a stable tilt angle.

3. **Motor Control**: Based on the tilt data, the Arduino controls the motor directions through digital pins and adjusts their speed using PWM signals. The L298N driver module handles the current flow to the motors.

4. **Balancing Logic**: Although the current implementation shows basic forward and backward motor control, it can be extended with a PID control algorithm to dynamically adjust motor speed and direction to maintain balance.

This physical setup provides hands-on experience in electronics, sensor calibration, and real-world control challenges such as sensor noise and latency.

Software Implementation (ROS 2 + Gazebo Simulation)

The software implementation was developed independently using **ROS 2 Jazzy** and the **Gazebo (gz)** simulation environment. It represents a complete digital twin of the balancing robot and allows experimentation with control strategies without hardware constraints.

Components:

- **URDF Model**: The robot is modeled using URDF (Unified Robot Description Format), defining its structure, joints, inertial properties, and wheel placements.

- **IMU Plugin**: A simulated IMU is attached to the robot base to mimic real-world sensor readings.

- **Controllers**:
    - **Joint Velocity Controller** for controlling wheel motors
    - **Custom PID Node** to compute motor speeds based on tilt angle
- **ROS 2 Nodes**:
    - A node subscribes to IMU data and calculates the angular deviation of the robot.
    - A PID controller processes this error and publishes velocity commands to the wheel joints.

Workflow:

Simulation Launch: The robot is spawned in a Gazebo world using ROS 2 launch files.
Sensor Feedback: The simulated IMU continuously publishes orientation data (pitch, roll, yaw).
Balancing Control: A ROS 2 node processes the IMU data and applies a control strategy (e.g., PID) to drive the wheel joints in real time.
Visualization and Debugging: The simulation allows live tuning of control parameters, easy debugging of system behavior, and detailed visualization using tools like Rviz and Gazebo client.

This simulation provides a clean, flexible environment for testing and refining algorithms without risk of damaging physical components.