Task: Implement a Differential Drive Robot Controller with RPM Computation and Waypoint Navigation

Objective

Develop a **ROS 2 C++ node** for a **differential drive robot** that:

1. **Processes velocity commands** to compute **RPM** for each wheel.
2. **Publishes computed RPMs** for motor control.
3. **Implements a separate Python script** to navigate the robot between two waypoints using **odometry and PID control**.
4. **Works with Gazebo Harmonic or later and ROS 2 Humble or later**.

Part 1: Differential Drive Controller (C++)

1. Environment & Dependencies

- **ROS 2 Humble or later**
- **Gazebo Harmonic or later**
- **ament_cmake** build system
- Dependencies: `rclcpp`, `std_msgs`, `geometry_msgs`, `nav_msgs`, `gazebo_ros`

---

2. Node Requirements

- **Subscribe to** `/cmd_vel` (`geometry_msgs::Twist`)
- **Compute wheel velocities** for a **differential drive robot**
- **Convert velocities to RPM**
- **Publish RPM values** for each wheel
- **Dynamically configurable parameters**

---

3. ROS 2 Parameters

Parameter Name	Type	Description
`wheelbase`	`double`	Distance between left and right wheels (m)
`wheel_radius`	`double`	Radius of each wheel (m)

`max_rpm`	`double`	Maximum allowed RPM for safety

---

4. Topics

Topic Name	Message Type	Role
`/cmd_vel`	`geometry_msgs::Twist`	Input: Robot velocity command
`/left_wheel_rpm`	`std_msgs::Float64`	Output: Left wheel RPM
`/right_wheel_rpm`	`std_msgs::Float64`	Output: Right wheel RPM

---

5. Gazebo Simulation Integration

- Implement a **Gazebo plugin or interface** to apply the computed RPM values to the robot's wheels.
- Ensure compatibility with **Gazebo Harmonic+**.

---

6. Testing Requirements

Run the node and test with:
sh
CopyEdit

```
ros2 topic pub /cmd_vel geometry_msgs/Twist '{linear: {x: 0.5},
angular: {z: 0.2}}'
```

-

Verify the correct RPM values using:
sh
CopyEdit

```
ros2 topic echo /left_wheel_rpm
ros2 topic echo /right_wheel_rpm
```

-
- **Test in Gazebo** by applying the computed RPMs to wheel joints.

Part 2: Waypoint Navigation (Python Script)

1. Script Requirements

- Create a Python script (`waypoint_navigation.py`) to:
    1. Subscribe to **odometry data** (`/odom` from `nav_msgs::Odometry`).
    2. Implement a **PID controller** to adjust **linear and angular velocity**.
    3. Navigate the robot to **two waypoints** sequentially.
    4. Publish velocity commands to `/cmd_vel`.
    5. Stop the robot once both waypoints are reached.

---

2. ROS 2 Parameters

Parameter Name	Type	Description
`waypoint_1_x`	`double`	X-coordinate of the first waypoint
`waypoint_1_y`	`double`	Y-coordinate of the first waypoint
`waypoint_2_x`	`double`	X-coordinate of the second waypoint
`waypoint_2_y`	`double`	Y-coordinate of the second waypoint
`kp`	`double`	Proportional gain for PID
`ki`	`double`	Integral gain for PID
`kd`	`double`	Derivative gain for PID

---

3. Topics

Topic Name	Message Type	Role
`/odom`	`nav_msgs::Odometry`	Input: Current robot position
`/cmd_vel`	`geometry_msgs::Twist`	Output: Velocity command

---

4. Testing the Navigation Script

Run the Python script with ROS 2 parameters:
sh
CopyEdit

```
ros2 run differential_drive_controller waypoint_navigation.py
--ros-args \
-p waypoint_1_x:=2.0 -p waypoint_1_y:=1.0 -p waypoint_2_x:=4.0 -p
waypoint_2_y:=3.0
```

1.
2. Verify that the robot correctly navigates to **both waypoints**.
3. Tune the PID parameters (`kp`, `ki`, `kd`) if needed.

Bonus (Optional)

- Implement **obstacle avoidance** using `LaserScan` (`/scan`).
- Use **Gazebo services** to reset the robot after reaching the goal.
- Add **unit tests** using `ament_pytest`.

Expected Deliverables

✅ **ROS 2 C++ package** for differential drive control
✅ **Gazebo simulation integration**
✅ **Python script for waypoint navigation with PID**
✅ **Configurable parameters for tuning**
✅ **Working demonstration in Gazebo**