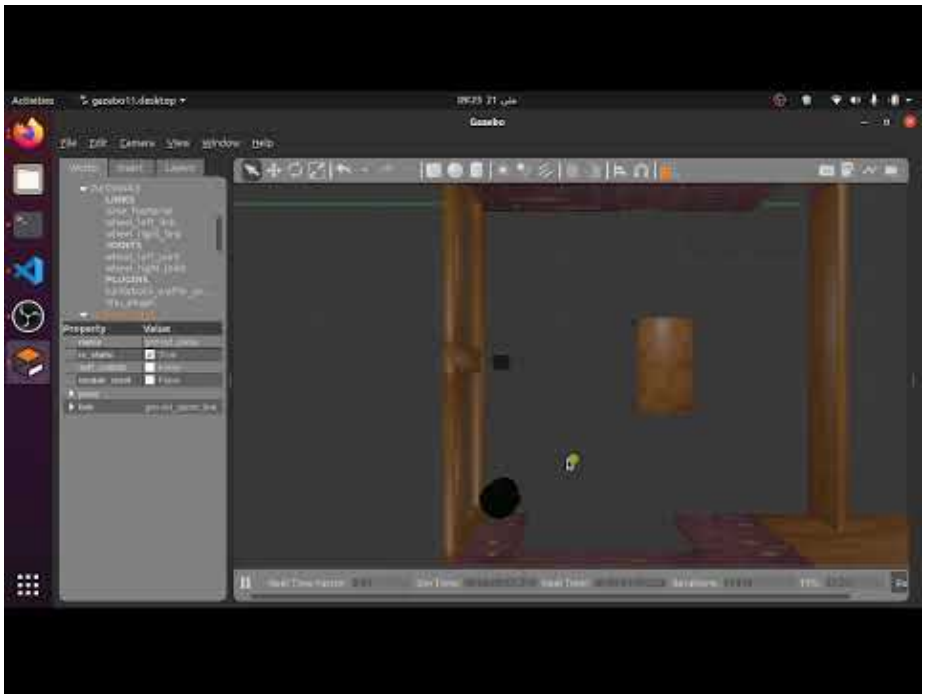
Mobile Robot Project Report

Jaysh Muhammad 450074

Wall Following with PID Control for TurtleBot3 in ROS Noetic



https://youtu.be/WSBhR1azYpI

This report details the implementation of a wall following algorithm for the TurtleBot3 in ROS Noetic, utilizing a PID controller to fine-tune the robot's path.

1. **ROS Environment:**
   - **ROS Distribution:** ROS Noetic Ninjemys
   - **Gazebo Simulator:** Gazebo 11
   - **Python:** Python 3
   - **Nodes:**
     - `wall_follow.py` (This node subscribes to laser scan data, calculates control commands, and publishes them to the robot's velocity controller)
2. **Libraries:**
   - **rospy:** Used for ROS communication, subscribing to topics, publishing messages, and handling ROS nodes.
   - **geometry_msgs:** Contains ROS message types like Twist, representing linear and angular velocities.

- **sensor_msgs:** Contains ROS message types like LaserScan, representing laser sensor data.
- **numpy:** Used for array manipulation and numerical operations on sensor data.

3. **System Architecture:**

- **Sensor:** The TurtleBot3's laser range finder provides real-time distance readings to nearby objects.
- **Perception:** The `wall_follow` receives laser scan data and processes it to detect the wall.
- **Control:** The PID controller calculates the robot's angular velocity based on the distance to the wall, aiming to maintain a constant distance from it.
- **Action:** The `cmd_vel_pub` publishes the calculated velocity commands to the robot's velocity controller, allowing it to move and rotate accordingly.

4. **Code Implementation:**

- **PIDController Class:**
  - Implements the Proportional-Integral-Derivative (PID) control algorithm.
  - Takes proportional gain (Kp), integral gain (Ki), and derivative gain (Kd) as parameters.
  - Computes the control output based on the error, integral, and derivative terms.
- **WallFollowerNode Class:**
  - Initializes the ROS node and subscribes to the laser scan topic.
  - Publishes velocity commands to the `cmd_vel` topic.
  - Defines the desired distance to the wall (`obstacle_threshold`), movement speed (`move_speed`), and rotation speed (`rotation_speed`).
  - `laser_callback` function:
    - Processes the laser scan data to detect the wall.
    - Calculates the error between the desired distance and the actual distance to the wall.
    - Uses the PID controller to compute the angular velocity.
    - Publishes the calculated velocity commands to the robot.

5. **Key Features:**

- **PID Control:** Precisely adjusts the robot's orientation to maintain a consistent distance to the wall.
- **Obstacle Avoidance:** The code checks for obstacles in front of the robot and adjusts its movement accordingly.
- **Flexibility:** The PID gains can be tuned to fine-tune the robot's response and achieve optimal wall following behavior.

6. **Running the code:**

1. Start ROS master: `roscore`

2. Launch TurtleBot3 simulation: `roslaunch turtlebot3_gazebo turtlebot3_house.launch`
3. Run the wall following node: `rosrun wall_follow_turtlebot wall_follow.py`

7. Conclusion:

This implementation provides a functional wall following system for the TurtleBot3 using PID control. It demonstrates the power of using PID control for robust and accurate robot navigation. The code can be further enhanced by incorporating more sophisticated obstacle avoidance strategies and adapting to different environmental conditions.