Name: Muhammad Akram

Roll NO:401618

Wall following robot using ROS and PID Control

Introduction: -

This project aimed to design and simulate a Wall Follower TurtleBot 3 robot using the Robot Operating System (ROS) and PID control. The robot was programmed to autonomously follow a wall at a constant distance, showcasing its navigation capabilities.

Objective: -

- Design a Wall Follower TurtleBot 3 robot in a simulation environment.

- Implement PID control to keep the robot at a constant distance from the wall.

- Assess the robot's performance in a simulated Environment.

Methodology: -

1. Robot Design: The TurtleBot 3 robot was designed and simulated using ROS and Gazebo.

2. Sensor Integration: A lidar sensor was incorporated into the robot to measure the distance from the wall.

3. PID Control: A PID controller was implemented to adjust the robot's velocity and steering angle, ensuring it maintains a constant distance from the wall.

4. Simulation: The robot operated in a Gazebo simulation environment with a predefined wall layout.

5. PID Tuning: After manual tuning, the following PID gain values were used: (K_p = 0.6), (K_i = 0), (K_d = 0.01).

6. **Performance:** The robot consistently maintained a distance of 1.3 meters from the wall and executed angular turns to avoid collisions.

Steps:-

1. **Install Ubuntu and ROS Noetic**
    - Install Ubuntu 20.04 (Focal Fossa).
    - Install ROS Noetic as the framework for this project.

2. **Setup Catkin Workspace**
    - Ensure you have an existing catkin workspace to create your project.
3. **Create Wall Follower Node**

- Within your project folder, create an executable script or node for the wall follower TurtleBot. The script link is available on my GitHub account. Ensure the script has executable and readable permissions.

4. **Necessary Installations Before Simulation**
   - Update and upgrade your system:
     sudo apt-get update

   sudo apt-get upgrade

   - Clone the required repositories for TurtleBot3 (for Noetic, use the `-b noetic-devel` branch):

```
cd ~/catkin_ws/src/

git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git -b noetic-devel

git clone https://github.com/ROBOTIS-GIT/turtlebot3.git -b noetic-devel

cd ~/catkin_ws && catkin_make
```

   - Install the TurtleBot3 simulation packages:
     cd ~/catkin_ws/src/ git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git cd ~/catkin_ws && catkin_make
   - Edit the `.bashrc` file to include useful aliases:
     gedit ~/.bashrc
   - Add these lines:
     alias burger='export TURTLEBOT3_MODEL=burger' alias waffle='export TURTLEBOT3_MODEL=waffle' alias tb3fake='roslaunch turtlebot3_fake turtlebot3_fake.launch' alias tb3teleop='roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch' alias tb3='roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch' alias tb3maze='roslaunch turtlebot3_gazebo turtlebot3_world.launch' alias tb3house='roslaunch turtlebot3_gazebo turtlebot3_house.launch' source /opt/ros/noetic/setup.bash source ~/catkin_ws/devel/setup.bash export TURTLEBOT3_MODEL=waffle export SVGA_VGPU10=0

5. **Create Project Directory**

   - Run the following commands to create your project directory:

```
cd ~/catkin_ws/src
catkin_create_pkg my_turtlebot_pkg rospy geometry_msgs sensor_msgs
```

6. **Move and Set Up Python Script**
   - Move your Python script to the project directory and make it executable:

```
mv /path/to/tb3_wall_follower.py ~/catkin_ws/src/my_turtlebot_pkg/src
chmod +x ~/catkin_ws/src/my_turtlebot_pkg/src/tb3_wall_follower.py
```

   - Rebuild your catkin workspace:
     cd ~/catkin_ws && catkin_make

7. **Run the Simulation**

- Set the TurtleBot3 model and launch the Gazebo simulation:

```
 export TURTLEBOT3_MODEL=waffle
roslaunch turtlebot3_gazebo turtlebot3_stage_1.launch
```

8. **Execute Python Script**
   - Run the Python script from Visual Studio Code to initiate the node.

Results: -

- The TurtleBot 3 effectively maintained the desired distance from the wall using the PID controller.
- The robot adapted to various wall layouts, demonstrating stable and consistent behavior.
- The simulation results confirmed the robustness and reliability of the PID control implementation for wall-following tasks.