Lab 2 Saftey Node

Lab 3 - Wall Following Node - ROS2 Documentation

This ROS2 node implements wall following for an autonomous vehicle using LaserScan data and AckermannDriveStamped messages to control the vehicle's speed and steering angle. The node subscribes to `/scan` for LiDAR (**Light Detection and Ranging**) data and publishes to `/drive` for vehicle control.

Node Overview

- **Node Name**: `wall_follow_node`
- **Purpose**: Perform wall following based on LaserScan data using a PID (**Proportional-Integral_Derivative**) controller to compute error and adjust the vehicle's driving commands.

Dependencies

The following ROS2 packages are required for this node:

- `rclpy` : ROS2 client library for Python.
- `sensor_msgs` : Message types for sensors, specifically using `LaserScan` for LiDAR.
- `ackermann_msgs` : Message types for Ackermann steering, specifically using `AckermannDriveStamped` for vehicle control.

Subscriptions and Publications

Subscriptions

- **Topic**: `/scan`
  - **Message Type**: `sensor_msgs/LaserScan`
  - **Purpose**: Receives LiDAR scan data for processing distances and angles to compute the wall-following error.
  - **Callback Function**: `scan_callback`

Publications

- **Topic**: `/drive`
  - **Message Type**: `ackermann_msgs/AckermannDriveStamped`
  - **Purpose**: Publishes vehicle control commands (steering angle and speed) based on the computed PID error from the LiDAR data.

Key ROS2 Functions

1. Node Initialization

This function initializes the ROS2 Python client. The node is instantiated with the name `wall_follow_node`.

```
rclpy.init(args=args)
```

2. Creating Subscriptions

The `create_subscription` method subscribes to the `LaserScan` topic to receive LiDAR data. The callback function, `scan_callback`, is triggered whenever a message is received.

```
self.sub_scan = self.create_subscription(LaserScan, lidarscan_topic, self.scan_callback, 1)
```

3. Creating Publishers

The `create_publisher` method sets up a publisher for the `AckermannDriveStamped` topic. This publishes drive commands (steering and speed) to control the vehicle.

```
self.pub_drive = self.create_publisher(AckermannDriveStamped, drive_topic, 1)
```

4. Callback Function for LiDAR Data

The `scan_callback` function processes incoming LiDAR data (`LaserScan` messages). The key components processed are:

- **ranges**: Array of distances measured by the LiDAR.
- **angle_min, angle_max, angle_increment**: Angular properties of the LiDAR scan. The processed data is used to compute the error, which is then passed to the PID controller for vehicle actuation.

```
def scan_callback(self, scan_msg):
```

5. PID Controller

The `pid_control` function implements a PID controller to adjust the vehicle's steering angle and speed based on the calculated error from the wall. The proportional, integral, and derivative (PID) terms are tuned for smooth and accurate wall following.

```
def pid_control(self, error):
```

6. Main Function

The `main` function initializes the ROS2 node and enters a loop to continuously process LiDAR data and publish drive commands. The node is shut down and destroyed when the program terminates.

```
def main(args=None):
    rclpy.init(args=args)
    wall_follow_node = WallFollow()
    rclpy.spin(wall_follow_node)
    wall_follow_node.destroy_node()
    rclpy.shutdown()
```

Message Types

`sensor_msgs/LaserScan`

This message provides data from a 2D LIDAR, including:

- **ranges**: The array of distance measurements (in meters) from the LIDAR.
- **angle_min, angle_max**: The start and end angles of the scan (in radians).
- **angle_increment**: The angular difference between two consecutive range measurements.

`ackermann_msgs/AckermannDriveStamped`

This message sends drive commands to control the vehicle. Key fields include:

- **steering_angle**: The angle at which the vehicle should turn.
- **speed**: The forward velocity of the vehicle.

Key Concepts

Wall Following

The node uses LiDAR data to follow a wall at a specified distance. The `get_range` and `get_error` functions calculate the car's distance from the wall, which is then used to compute the error.

PID Controller

The PID controller computes the steering angle and speed of the vehicle based on the error between the desired distance and the current distance from the wall.

Possible Future Improvements

- **PID Tuning**: Further tuning of the PID gains may be needed for optimal performance in different environments.
- **Robustness**: The code currently assumes ideal sensor data; adding handling for noisy or missing data could improve reliability.

Example ROS2 Commands

Running the Node

To run the `wall_follow_node`, use the following command:

```
ros2 run <your_package> wall_follow_node
```

Checking Topics

To view the topics the node is subscribing to and publishing, use:

```
ros2 topic list
```

Inspecting Messages

To inspect the messages being published on the `/drive` topic, use:

```
ros2 topic echo /drive
```