CS/ME/ECE/AE/BME 7785

Lab 5

Due: March 17^{th}, 2023, 4pm

Overview

The objective of this lab is to get you familiar with the mapping, localization and path planning capabilities of the ROS2 navigation stack. The final goal is to write a script to make a simulated and real robot autonomously navigate to a series of global waypoints. Unlike previous labs, this is not one in which you will be programming algorithms directly, but rather understanding previously written code/algorithms and tuning the parameters to suit your goals.

We encourage you to use all available resources to complete this assignment. This includes looking at the sample code provided with the robot, borrowing pieces of code from online tutorials, and talking to classmates. You may discuss solutions and problem solve with others in the class, but this remains a team assignment and each team must submit their own solution. Multiple teams should not jointly write the same program and each submit a copy, this will not be considered a valid submission.

1 Lab Instructions

1.1 Gazebo Simulation

For this lab, you will be required to leverage Gazebo simulation environment, which can be used for code development and initial testing. A reminder, the tutorial on simulating the Turtlebot in Gazebo and initial modifications to the simulation environment can be found in Lab 2. In addition to this initial setup, for this lab, we have created a Gazebo environment similar to the one you will find in the lab to test your navigation code. The files and directions on how to use them are available here,

https://github.gatech.edu/swilson64/7785_Lab5_Gazebo_Files

Note: This maze environment will be identical to the one used for your Final project.

Your team will be expected to perform the lab in simulation and on the robot. To perform this lab in simulation, follow the same steps as in **??** and **??**, but with Gazebo running instead of bringing up the real Turtlebot. Gazebo will subscribe and publish the same messages as the real robot.

1.2 Create a map through teleoperation

Generate a map of your environment using the instructions found at:

http://emanual.robotis.com/docs/en/platform/turtlebot3/slam/

make sure you select the instructions for ROS2 Foxy and don't use the ROS1 Kinetic instructions that the web page defaults to.

> ***Note:*** If during any of the steps your PC or Turtlebot throws errors of the form,
>
> `ERROR: cannot launch node of type [example/example]:  example`
>
> You are missing that <example> package, install it with the command
>
> `sudo apt-get install ros-foxy-example`.
>
> You can tab complete to find the correct name or Google the name of the package to find the correct instillation command for ROS Foxy.

In later stages of the lab, you will have your robot navigate to a predefined goal. To ensure that everyone uses (approximately) the same global coordinates, start your robot with its wheels on the blue tape marks on the floor, facing in the direction of the arrow. Then drive your robot around to complete the map. You can also save this file wherever you would like in step 9.4.1 by changing the directory and file name after the `-f` command.

1.3 Localization, Path Planning, and Navigation

To start, use the map you've generated to have the robot navigate to a point you specify in the rviz GUI. Instructions found at:

https://emanual.robotis.com/docs/en/platform/turtlebot3/navigation/

make sure you select the instructions for ROS2 Foxy and don't use the ROS1 Kinetic instructions that the web page defaults to.

> ***Note:*** If you changes the save location of your map files, make sure they are the same as defined after the `map_file:= ...` command in step 10.1. This navigation is using a flavor of particle filter for localization and A*/Djikstra's for path planning.

After you verify everything is working through the GUI in RViz, you can determine points in your map's coordinate frame using publish point in the RVIZ GUI and using the following command,

```
ros2 topic echo /clicked_point
```

you should notice the topic is publishing the the clicked position. However, the Navigation Stack requires a pose consisting of 3 positional coordinates (x, y, z), and 4 orientation coordinates (x, y, z, w). These four orientation coordinates are a way of representing a rotation called a Quaternion. To publish a navigation goal to the robot rather than using the RVIZ GUI, use the command line argument,

```
ros2 topic pub /goal_pose geometry_msgs/PoseStamped '{header: {stamp: {sec: 0,
nanosec: 0}, frame_id: "YOUR_MAP_NAME"}, pose: {position: {x: 3.9, y: -0.85,
            z: 0.0}, orientation: {x: 0, y: 0, z: 0, w: 1.0}}}'
```

the message being published here is a PoseStamped geometry message.

https://docs.ros.org/en/diamondback/api/geometry_msgs/html/msg/PoseStamped.html

You will need to use this message type to pass your script's goal points to the NavStack for this lab.

1.4 Drive to global waypoints

On the day of the lab, 3 random goal points within your map will be chosen for your robot to navigate between. You should be able to put these point into your script through a file or change them easily in your code.

2 Parameters that should be tuned

When using the navigation stack, there are a few parameters you can adjust to get more consistent performance. All of the parameters are found in the `turtlebot3/turtlebot3_navigation/param` directory, and they're loaded by the navigation launch file. All you have to do is change the parameters in the .yaml files (you can open these with a text editor), no compiling needed.

Full details about the Nav2 Stack can be found at https://navigation.ros.org/. In particular the Configuration Guide and Tuning Guide may be useful. Within the configuration guide, some parameters that may be useful to understand and tune are,

- Costmap 2D - This provides the environment the robot plans within. The parameters within this section deal with how "Dangerous" your robot will believe walls are and plan accordingly.
- DWB Controller - This configures the low level controller that is attempting to follow the path generated by the global planner. Similar to a Dynamic Window Approach, this is an optimal control algorithm and changing the cost coefficients, planning window, and other parameters will change the behavior of the robot tremendously. From previous years experience, the DWB controller does not perform well in a constrained environment (like the maze). You may find more success using the embphPure Pursuit controller. Feel free to experiment with any of the available low level controllers.
- Frequency of Planners - In the controller and cost map parameters, there are frequency parameters. Recall discrete control and Lab 3 where the sampling time impacts performance of controllers. Make sure these frequencies are within the range of your sensors (LIDAR).

- For passing waypoints into the Nav Stack from a script, use the `'/goal_pose'` topic which takes a *PoseStamped* message type.
- To determine if the robot has been reached you can use any part of the Nav Stack feedback. One method of doing this is subscribing to the `/navigate_to_pose/_action/feedback` topic which uses the `NavigateToPose_FeedbackMessage` message type (https://github.com/ros-planning/navigation2/blob/main/nav2_msgs/action/NavigateToPose.action).

3 Grading

Show an instructor the generated map of the simulated maze	20%
Show an instructor the generated map of the maze	20%
Navigate to 3 waypoints specified at grading time in simulation (10% per waypoint)	30%
Navigate to 3 waypoints specified at grading time (10% per waypoint)	30%

4 Submission

You have two required submissions for this lab.

1 Your ROS package in a single zip file called `TeamName_Lab5.zip` uploaded on Canvas under Assignments–Lab5.

2 A live demonstration of your code to one of the instructors. This can be done anytime before the due date at the top of this lab. Class will meet in the lab room on the due date to allow everyone to demo their navigation. If you demo before the due date you do not need to come attend class that day.