CS/ECE/ME/BME/AE 7785

Lab 5

Due: November 5th, 2021, 3pm

1 Overview

The objective of this lab is to get you familiar with the mapping, localization and path planning capabilities of the ROS 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.

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.

2 Lab Instructions

2.1 Gazebo Simulation

For this lab, you will be required to leverage Gazebo simulation environment 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. The signs on the wall are associated with Lab 6 and the Final so you can ignore them for now.

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 subsection 2.2 and subsection 2.3, but with Gazebo running instead of bringing up the real Turtlebot. Gazebo will subscribe and publish the same messages as the real robot.

2.2 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/

> ***N*ote:** 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-melodic-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 Melodic.

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.

2.3 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:

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

> ***N*ote:** 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* for path planning.

After you verify everything is working in RViz, echo the topic that is responsible for handling the 2D Nav Goals you are creating through the RViz GUI through the command,

```
rostopic echo /move_base_simple/goal
```

you should notice the topic is publishing the map frame it is navigating in as well as the goal pose. The pose consists of 3 positional coordinates (x, y, z), and 4 rotational coordinates (x, y, z, w). These four coordinates are a way of representing a rotation called a Quaternion. To publish to this topic rather than use the GUI, use the command line argument,

```
rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped
   'header: stamp: now, frame_id: "YOUR_MAP_NAME", pose:
position: x: 1.0, y: 0.0, 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 scripts goal points to the NavStack for this lab.

2.4 Drive to global waypoints

Read in `global_waypoints.txt` and navigate the robot to each waypoint in turn. Specifically, the file will contain 3 waypoints not known to you until grading time. An example file is provided with this assignment, but feel free to try other waypoints. Your code should have the robot drive to the first waypoint, stop for 2 seconds, then go on to the second waypoint and so on. The waypoint file lists seven values per line, corresponding to the 3D position and the orientation quaternion for the waypoint, in that order.

3 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 various .yaml files (you can open these with a text editor), no compiling needed.

Recommended parameters to adjust:

- **dwa_local_planner_params.yaml**
  - **xy_goal_tolerance, yaw_goal_tolerance:** Increasing the goal tolerance will allow the robot to stop when it's "close enough" to a goal point. This can prevent the robot spinning around in circles for a while when it gets close to a goal to get to the exact correct point, provided your solution can handle reaching a larger area as a goal rather than a very specific point/angle.

  - **path_distance_bias, goal_distance_bias:** Increasing these will cause the robot to more strongly follow the planned path or move towards the goal. Think of it like the switching controller we discussed in the lecture.
  - **max_vel_x, min_vel_x, max_rot_vel, min_rot_vel:** These all set bounds on how fast the robot will move. Fast moving robots may shake and mess up localization, so reducing maximum velocities may improve performance.

- **costmap_common_params_burger.yaml**
  - **inflation_radius:** This will determine how large to make obstacles. Increasing this will prevent the robot from driving close to walls.
  - **cost_scaling_factor:** This will determine how much the robot should be repelled by obstacles. Increasing this will make the robot prefer not to drive close to walls, but will allow it if necessary.

4 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	30%
Navigate to 3 waypoints specified at grading time	30%

5 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.