Homework 3: Turtlebot3 Teleoperation and Navigation

Assigned: 10/29/2020 **Due:** 11/20/2020

Introduction

The purpose of this homework is to give you a chance to learn how to write ROS code. We're going to look at two problems: teleoperation and navigation, and we're going to do it with the Turtlebot3. Obviously you need to have ROS running on a system that you can use, preferably Noetic or Melodic. You should also hopefully be making use of a machine with a reasonable amount of processing power. Gazebo uses a decent amount of resources. Assuming you have ROS installed, you need to get the Turtlebot3 Gazebo simulator running. If you haven't done that yet, here are some resources:

- https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/
- https://automaticaddison.com/how-to-launch-the-turtlebot3-simulation-with-ros/

Once you've got that running, return to this document and read the instructions for Part I and II.

Provided Code

In order to aid you in your work, we are providing you with some code. For Part I there is a skeleton of a C++ or Python teleoperation node that you need to complete. For Part II, we provide a stand-in for the testing node that your node will be sending service calls to, so that you can test that aspect of things. The node that we provide does not represent any of the testing that will be done, it simply provides the service that can be called. All of this code is in a zip file along with a README with instructions on use. Make sure to download and use it!

Part I: Teleoperation

Using the provided skeleton code, teleoperation.py or teleoperation.cpp, write a teleoperation node which moves the Turtelbot in response to key presses. The provided code should do all of the key-press registering for you (please let me know if you have issues), all you need to do is publish the movement messages to the proper topic (which is well documented).

The only requirement for this part is that your node creates motion in the robot based on the following key mapping. For your teleoperation node, the mappings are:

W is forward motion.

S is backward motion.

A is count-clockwise rotation.

D is clockwise rotation.

X is emergency stop.

Whether you implement velocity ramping (slow speed up), velocity decay (speed slows after key is no longer pressed), controls for increasing the keypress step size, controls for increasing or decreasing the max speed, is up to you. Please try to do ay least one of these.

Please have your node print out the key mapping when it starts, including any additional key functions.
This is a relatively easy piece of work. It mostly serves as a test that you understand basic ROS principles and have gotten everything to work. Between this and the submission format, you should have an easy guaranteed 30 points.

Part II: Navigation

For the second part of the homework, you must write a ROS node which allows the Turtlebot to navigate to a selected point in the environment, which you will do without knowledge of a map, and without hitting any obstacles. The exact algorithm you use in order to navigate to the point is up to you, though you should not use entirely random motion. We will be testing this in the gazebo world launched by the *turtlebot3_world.launch* launch file, in the *turtlebot3_gazebo* package. The starting point in the world may be changed, so do not assume your starting point will remain as is.

Your navigation node must achieve the following behaviors:

- Accept 2D Navigation goals from RViz on the topic */move_base_simple/goal*. If a goal comes in and a navigation is not currently in progress, a new navigation must start with no further input. You may ignore all new navigation goals until your robot has reached the final goal.
- Navigate to the goal, using information about its local environment from its LIDAR sensor and position information from the simulator.
- Avoid hitting obstacles along its way. For this, please use the Turtlebot3's simulated LIDAR sensor in Gazebo.
- Detect once it has reached its goal. This may be done by checking the goal against the position information given by the simulator on the topic */odom*. Note that */odom* gives displacements with relation to the origin of the world.
- Once it has reached the goal, call the */csci_5551/goal_reached* service. This service is of the type std_srvs/Trigger.srv. This service will be provided by the testing software, and is also provided by the stand-in node that you have been given.

As you can see in the Rubric, you will receive points for every time your navigation successfully reaches the goal (and sends the service call indicating as much), you will lose points if the robot hits obstacles, and you will lose points if your node does not receive goals on the appropriate topic and send service requests to the appropriate service.

Navigation Algorithm Tips

For your algorithm design, you might want to start by looking at the classic Bug Algorithms, which are explained well in the following resources:

- McGuire, Kimberly N., G. C. H. E. de Croon, and Karl Tuyls. "A comparative study of bug algorithms for robot navigation." Robotics and Autonomous Systems 121 (2019) `https://www.sciencedirect.com/science/article/abs/pii/S0921889018306687` (If you're having access issues, `https://arxiv.org/abs/1808.05050` is the same paper).
- Slides from the University of Maryland `https://spacecraft.ssl.umd.edu/academics/788XF14/788XF14L14/788XF14L14.pathbugsmapsx.pdf`

Bug 1 or Bug 2 might be a good starting point, but as long as your algorithm works, it's fair game! The only case in which we will assign no points for algorithm complexity is if you are just doing random motion.

Extra Credit 3: Navigation with Mapping and Localization

For the extra credit associated with this homework, implement a new navigation algorithm which takes into account mapping information. For your mapping algorithm please use gmapping. The same expectations from before apply in terms of navigation goals, randomness of starting point, not hitting obstacles etc. However, in this problem you must not use your local sensor besides in its use for building the map. Use the map built by gmapping to plan your path, avoid obstacles, and reach the goal. You should not expect the mapping to be launched by us (launch it in your *navigation_extra_credit.launch* file. If you'd like an extra challenge, try using a proper localization node instead of Gazebo's ground truth for your robot localization information.

Submission Format

Getting your submission right is worth 10 points. Pay attention to the following information.

Your submission for this homework will be a zipped file (.zip format) containing all of your code for the homework, uploaded to Canvas. Inside that zipped folder, we must find your code in the form of a ROS package, named [your x500]_hw3 . The expected form is as follows:

```
[your x500]_hw3/
        package.xml
        CMakeLists.txt
        README.txt
        src/
                Any code for the two nodes you wrote, Python or C++.
        launch/
                teleoperation.launch
                navigation.launch
```

As far as our expectations for the content of these files:

- *package.xml* – We expect that you will have generated this file, and filled out all relevant fields such as author, version number, license, etc.
- *CMakeLists.txt* – We expect that you will have generated this file, and that it will lead to a successful compilation of your code.
- *README.txt* In the readme, put your name, your x500, a list of any software you used in your code that might need to be installed (ROS packages or non). It is not recommended that you use any software that requires anything more than a simple pip or apt install. Your README file should also contain a text description of your navigation algorithm. Any additional information you want to provide here is also welcomed.
- *src/* Your Python or C++ code for your two nodes. If you intend to use a different language, please contact Michael first to talk about it.
- *launch/* We expect two launch files: *teleoperation.launch* and *navigation.launch*. These should launch your nodes (and only your nodes, not Gazebo or RVIZ) for the specified parts of the problem.

Any compilation or running issues will attempt to be resolved, but please, for the sake of your TA's sanity, do not overload your code with unnecessary, annoying to install dependencies.

EXTRA CREDIT SUBMISSION: A separate submission will be created for the extra credit. PLEASE DO NOT SUBMIT YOUR EXTRA CREDIT TO THE HW3 SUBMISSION. For the extra credit, submit in the same way (a zipped ROS package), just update your navigation portion of things to make use of a map.

Rubric

The grading for this homework is listed below.

Correct submission format (package, launch files, etc)	10 points
Part I: Teleoperation	20 points
Node fulfills all listed capabilities	*20 points*
Part II: Navigation	70 points
Node implements all required topics/services	*20 points*
Robot reaches goal out of 5 trials (4 pts. each)	*20 points*
Robot avoids hitting obstacles	*20 points*
Pathfinding algorithm complexity	*10 points*
Total	**100 points**

Completing the Extra Credit for this homework will count towards the EC counter (worth 3 points at the end of the semester if a student completes all of them).

How To Get Help

- If your question is about a specific error you're having with a piece of code, whether it's in the installation process or while writing code, Google it first. Please do not flood your TA's with questions that Google can answer, it makes it harder to answer the questions that Google can't answer.
- If you're having a problem, and you've worked on it a bunch, and Google has been no help, please ask your TA's through the approved methods (office hours and email).
- If you have a question about requirements or bounds of the homework, please contact Michael Fulton (a TA for this course) specifically.

Some Leads

If you're having trouble, check these out...

- Launch File Syntax
- Writing Publisher and Subscriber (Python)
- Writing Publisher and Subscriber (C++)
- Creating a Package
- Message type of the /cmd_vel topic
- Message type of the /scan topic
- Message type of the /odom topic