AA 274A: Principles of Robot Autonomy I
Section 7: ROS Parameters

Our goals for this section:

1. Learn how to use ROS parameters
2. Learn how to dynamically reconfigure node parameters
3. Work on any remaining bugs in your robot planners and controllers

1 Setting up and using ROS parameters

Often there are parameters that are useful across multiple nodes. Such parameters would ideally be shared between nodes to ensure that their values throughout your robot stack are in sync. Additionally, there may be parameters whose values depend on the situation the robot is being launched into, and thus it would not be ideal for them to be hardcoded and modified for each situation. These are some of the reasons for the ROS parameter server.

Launch the robot stack as described before in Section 5, and then try the following command:

```
rosparam list
```

You will see that there are already a large number of parameters on the ROS parameter server. Where exactly are those being set? Open the `turtlebot3_nav_sim.launch` file and you will see that most of the lines are actually setting parameters! Thus, the value of these is set in a launch file and different launch files could be written to initialize these parameters differently for a different task.

We can also use `rosparam get` to see the current value of parameters from the terminal, providing another way to get information about the current state of your robot stack. Try using this on a few parameters.

1.1 Creating our first ROS params

Now let's give ROS params a try for our navigator node. The values for maximum rotational velocity of the robot are not navigator-specific, and thus they could conceivably be shared across nodes.

1. Make sure that your custom launch file `my_nav.launch` launches the `navigator.py` node directly and launches `turtlebot3_nav_sim.launch`. Make sure that you've commented out the line inside `turtlebot3_nav_sim.launch` which launches the `navigator.py` node.
2. Next, modify the launch of the navigator node inside `my_nav.launch`. Make it create parameters `v_max` and `om_max` within the navigator node namespace and give them the values listed in `navigator.py`. Take a look at the gmapping section of `turtlebot3_nav_sim.launch` for an example of how to do this.
3. Next, modify `navigator.py` to get the value of these parameters from the ROS param server rather than hardcoding them in the script. Take a look at `supervisor.py` for an example of how to do this.

Note the difference between `self.rviz = rospy.get_param("rviz")` and `self.pos_eps = rospy.get_param("~pos_eps", 0.1)`.

1.2 Parameterizing launch files

What if we have another task where we want to run the same stack but with the robot limited to a more conservative speed? One possibility would be to make another copy of `my_nav.launch` and change the values of the parameters inside. However, a more elegant and extremely useful solution is to parameterize the launch file itself.

Open `turtlebot3_signs_sim.launch` in `asl_turtlebot` and you will notice that it is calling a `gazebo_ros` launch file while passing in some arguments. It additionally has some arguments of its own with default values that could be *overridden* if it is called by another launch file that passes in different values for those arguments. As robot stacks become more complex, you can imagine how crucial it is to be able to hierarchically call trees of launch files, using such arguments to repurpose each launch file and ultimately parameters and nodes for the task at hand.

Now using `turtlebot3_signs_sim.launch` as an example, modify `my_nav.launch` again to use launch file arguments to set the value of the `v_max` and `om_max` parameters. Then create a second launch file `my_nav_slow.launch` which calls `my_nav.launch` with max velocity values that are at half the speed.

Problem 1: Test these launch files with your robot sim and paste the contents of your `my_nav.launch` and `my_nav_slow.launch` files into your submission.

2 Dynamically reconfiguring node parameters

ROS params give a great way to synchronize parameters across your stack on node startup, but what if you want to modify parameters while your nodes are running? There is a `ros param set` command, but if your nodes only use ROS params to initialize internal node variables and don't continue checking them, updating the ROS params during operation won't result in any changes.

Thankfully, we have another tool called `dynamic_reconfigure`. We have already set this up for some parameters in the navigator script. Start the navigator and run

```
rosrun dynamic_reconfigure dynparam list
```

You will see that dynamic parameters are set up for the pose controller gains in `turtlebot_navigator`. Use the `dynparam get` and `dynparam set` to get the current value of the one of the gains and to set it to a different value (see the `dynamic_reconfigure` documentation).

Reconfiguring parameters like this through the command line is quite tedious, so thankfully there is an alternative. Run

```
rosrun rqt_reconfigure rqt_reconfigure
```

This will provide an extremely convenient interface for changing your robot's parameters on the fly! Give it a try on the pose controller gains.

2.1 Setting up dynamic parameters

Now let's try extending this to more parameters in our navigator. Unlike normal ROS params, dynamic parameters are created in special `cfg` files. You can find one for the navigator at `asl_turtlebot/cfg/navigator.cfg`. Check the `dynamic_reconfigure` tutorials to see what the arguments in the `gen.add` function mean. Check the `dyn_cfg_callback()` function in `navigator.py`. Now, add a dynamic parameter for any one parameter from `navigator.py` — this could be the trajectory smoothing parameter, the desired velocity, or anything else that you would like to change on the fly!

Problem 2: Test this on your robot sim and paste the contents of your `navigator.cfg` and `navigator.py` files into your submission.

3 Tuning and debugging your planners/controllers

You now have a large number of tools for tuning and debugging your robot stack! Spend any remaining time practicing using these tools to track down bugs in your planners and controllers and improve the performance of your robot.

Take note of any inefficiencies in the robot's current stack that might be improved upon by your group during the final project!