Overview

The goals for this section:

1. Learn how to use ROS Parameters.
2. Apply heading controller in hardware.

Setup

Task 0.1 — Cleanup. Remove directory `~/autonomy_ws` if it exists. This may be leftover from a previous section.

Task 0.2 — Directories. Create the base directory structure for your group's ROS workspace, `~/autonomy_ws/src`.

Task 0.3 — Clone and Branch. Clone your group's GitHub repo to the `src/` directory, and create a new branch called `s4_inperson`.

Task 0.4 — Copy homework repo. We will be using the code you wrote in HW1 to control the TurtleBot. Copy the `autonomy_repo` ROS package (of any of your group members) inside of your group's git repository.

Task 0.5 — Build. Navigate back to the root of your workspace, `~/autonomy_ws/`, and build your ROS workspace using `colcon build` .

Section 2 Cont: Full TurtleBot Bring Up

Let's test out your custom launch file on the actual robot! Determine the ROS_DOMAIN_ID of the laptop you are using for this section using this command:

```shell
echo $ROS_DOMAIN_ID
```

Call a CA over to assign a robot to you based on your domain ID!

CAs will move your TurtleBot to the map on the ground, and turn on your TurtleBot for you.

Then, use the following workflow to start up the TurtleBot.

```Shell
# Use SSH to connect to the Turtlebot

ssh aa274@<name_from_ca>.local



# Enter the password given by the CA



# Start the ROS environment

micromamba activate ros_env



# Bring up the Turtlebot

ros2 launch asl_tb3_driver bringup.launch.py
```

Task 1.1 — Available Topics. Once this is all running, which ROS2 topics are available? You can run `ros2 topic list` from a new terminal window on the laptop.

Checkpoint — Call a CA over to demonstrate that your TurtleBot is running!

Task 1.2 — SimtoReal. Run the launch file you created previously, and see the robot moving! To stop the robot, publish a kill message from a new terminal.

Checkpoint — Call a CA over to demonstrate this.

Testing your HW

Task 2.1 — Launch simulator. Firstly, we will test out your HW1 code before you can test it out on the actual robot. Just like in HW1, start your TurtleBot simulator by:

```Shell
ros2 launch asl_tb3_sim root.launch.py
```

Task 2.2 — Launch your heading controller. In a new terminal, source your workspace, and then run:

```
Shell
source ~/autonomy_ws/install/setup.bash
ros2 launch autonomy_repo heading_control.launch.py
```

Task 2.3 — Generate plot. Finally in another new terminal, navigate to the root of your workspace, source your workspace again, and then run:

```
Shell
cd ~/autonomy_ws
source install/setup.bash
ros2 run autonomy_repo p3_plot.py
```

Common error: If you get an error after running `p3_plot.py` regarding your directory being incorrect, open `p3_plot.py` from your `~/autonomy_repo/scripts` folder and edit the path in **line 63** to match the path of your `~/autonomy_repo/plots` folder like below:

```
Shell
filename = Path("src/autonomy_repo/plots/p3_output.png")
```

Checkpoint — Call up a CA and show them your robot turning and your resulting plot `p3_output.png`!

ROS2 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 hard coded and modified for each situation. These are some of the reasons for the ROS2 parameter server.

For this case, we will define the heading controller gain as a ROS2 parameter. Navigate to your `heading_controller.py` script in `~/autonomy_ws/src/<repo>/autonomy_repo/scripts`.

Task 3.1: Declare the heading controller gain as a ROS2 parameter in the `__init__` method of the `HeadingController` class. Name the parameter `kp` and set it to a default value of 2.

Task 3.2: Use the `@property` decorator that defines a property `kp` (of type `float`) of the `HeadingController` class, and gets the real-time parameter value of the controller gain. See here for an example from the `BaseController` class.

Cheat Sheet:

```python
# Declares a parameter named 'my_int' and sets it to a default value of 7
self.declare_parameter("my_int", 7)

# Retrieves the value of the parameter named 'my_float'
self.get_parameter("my_float").value
```

Gain tuning in Sim

Task 4.1 — Launch sim. Launch the simulator and the heading controller node (i.e., Tasks 2.1 and 2.2).

Task 4.2 – List ROS Parameters. In a new terminal, run:

```shell
ros2 param list
```

Task 4.3 – Identify control gain. In the output of the command, find the controller node, and identify the control gain parameter. Then to display the type and current value of the parameter, run:

```shell
ros2 param get <node_name> <parameter_name>
```

Task 4.4 – Control to goal. Set a new goal pose in RViz by selecting the `2D Goal Pose` button on the top toolbar and then clicking and dragging in the environment to set a goal heading.

Task 4.5 – Change control gain. Now to change the proportional control gain, run:

```shell
ros2 param set <node_name> <parameter_name> <value>
```

Task 4.6 – Test new gain. In RViz, have the robot navigate to a new goal pose. Notice the change in behavior with the new control gain.

Checkpoint — Show the CA your updated parameter value and your robot turning in sim.

Full TurtleBot Bring Up

Time to test this out on the actual robot! Determine the ROS_DOMAIN_ID of the laptop you are using for this section using this command:

```shell
echo $ROS_DOMAIN_ID
```

Call a CA over to assign a robot to you based on your domain ID!

CAs will move your TurtleBot to the map on the ground, and turn on your TurtleBot for you.

Then, use the following workflow to start up the TurtleBot.

```shell
# TODO: close any terminals running the sim in Gazebo before
using the hardware

# Use SSH to connect to the Turtlebot
ssh aa274@<name_from_ca>.local

# Enter the password given by the CA
# Start the ROS environment
micromamba activate ros_env

# Bring up the Turtlebot
ros2 launch asl_tb3_driver bringup.launch.py
```

Task 5.1 — Available Topics. Like last time, run `ros2 topic list` from a new terminal window on the laptop to verify that your TurtleBot is running!

Task 5.2 — SimtoReal. Launch the heading controller node. Specify goal poses in RViz, and try changing your controller gain using `ros2 param set`.

Checkpoint — Call a CA over to demonstrate this.

Clean Up

Clean up the workstation.

1. Push your code (`add`, `commit`, and `push`) to your `s4_inperson` branch!
2. Create a PR either from the GitHub web page or by running `gh pr create` in your terminal.
3. Verify that you've pushed correctly by checking if the code you wrote is in the repository on GitHub.com
4. Delete the `~/autonomy_ws/` .
5. Log out of GitHub on the workstation.
6. Close all terminals and windows!

Checkpoint — Call a CA over to sign you out!