CS/ME/ECE/AE/BME 7785

Lab 2

Due: Spetember 29, 2023 at 4 pm

Overview

The objective of this lab is to get you familiar with writing ROS code for the Turtlebot 3. In this lab you will create your own ROS2 package containing two communicating nodes that enable the robot to turn in place to follow an object. You can also create one debugging node that allows your computer to display the processed image from the Turtlebot. You can use your image processing code from Lab 1 that enables the robot to identify the location of an object in the frame or create a new one for this task.

Network lag is an important consideration when running robots. If you run code on your laptop, each image must first be streamed from the robot over Wi-Fi, which will add significant delays and make your robot very slow to react to changes in the environment. You can start the lab by running code offboard on your laptop while you are debugging. Once your code is in a stable form, you should then change to running all files on-board the robot. You *should not* have images passed from the Turtlebot to your laptop for processing during the demo, it is okay to have the debug node running on your computer to visualize your image processing outputs. To move folders from your computer to the robot, you may use the `scp` (secure copy) command. For example,

scp -r <Directory on your cpu to copy> burger@<ip-of-burger>:<Directory to copy to on robot>

Additionally, the robots have GIT installed to pull and push code to and from repositories if you prefer to use version control for your code. Alternatively, you can use VScode remote ssh and there is a guide on how to set this up availabe on Canvas.

We strongly 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 other teams, but 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.

Lab

For this lab create a new package called **TeamName_object_follower** (refer back to the ROS2 tutorials from Lab 0 if needed). For your `Team Name` you can make one up or use the `Team Number` you were assigned with your lab teams. Note, package names must begin with a letter so if your `Team Number` was 1 you could name your package `team1_object_follower` or `one_object_follower`. Some potentially useful dependencies include `sensor_msgs`, `std_msgs`, `geometry_msgs`. You may structure your software solution in any way you would like. We suggest adding two nodes to the package:

find_object: This node should subscribe to receive images from the Raspberry Pi Camera on the topic `/image_raw/compressed`. Example code in Python:

```
self._img_subscriber = self.create_subscription(CompressedImage, '/image_raw/compressed',
                        self._image_callback, qos_profile)
```

Note: The `self.<variable>` is present because this subscriber is created within a class. Additionally, the `qos_profile` is a networking setting that is discussed later in this document. For more information see the example file provided with this lab `view_img_raw.py` and `view_img_raw2.py`. If you would like to clone the packages that use these scripts, you can clone the following ROS2 package into your ROS2 workspace on your computer,

https://github.gatech.edu/swilson64/camera_viewer

If you wish, you can use GIT to clone this repository,

```
git clone https://github.gatech.edu/swilson64/camera_viewer.git
```

From this point you should be able to use the same code from Lab 1 to take this image and identify the location of your object in the frame (whatever object you chose). If you would like, you can bring in any object you would like to track and use that instead. All that is required is code that reliably identifies and outputs the location of the object in your camera frame. Once you've located the object in an image, this node should publish the pixel coordinate of the object. To do this, you can create your own message type (refer to ROS tutorials) or use an existing message type such as the `geometry_msgs/Point` message. To help debug this, it may be useful for this node to also publish the processed frame for a node run on your computer to subscribe to and display (or use rqt_image_viewer).

To view compressed image topics on your computer you will need to install an additional transport package. You can do this with the following command on Linux,

```
sudo apt-get install ros-humble-image-transport-plugins
```

rotate_robot: This node should subscribe to the object coordinate messages from your **find_object** node and publish velocity commands to the robot:

```
self._vel_publish = self.create_publisher(Twist, '/cmd_vel', 5)
```

Configure your code to have the robot turn to follow the object in the image (turn right when the object is on the right of the image, turn left when it is on the left, and stand still if the object is directly in front of the robot). The robot should not drive forward or back, only turn.

For this lab, the object can be at any distance from the robot that you choose within reason. You algorithm cannot require the object to take up the entire frame and there must be some tolerance to the distance (i.e. you cannot tell us the object must be exactly 30cm from the robot to work, you can say it should be around 30cm from the robot).

Simulation

For this lab, the Gazebo simulation environment can be used for code development and initial testing. The tutorial on simulating the Turtlebot in Gazebo can be found at,

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

Note: When cloning the repository to your computer in step 6.1.1.1 make sure you clone the ROS2 Humble branch into your current workspace (ros2_ws or whatever you named it, you can install it to a turtlebot3_ws but will need to source and build that new workspace appropriately). This can be done with the command,

```
git clone -b humble-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
```

You may notice the simulated Turtlebot3 Burger does not have a camera. To remedy this, we have created an updated model of the Turtlebot with parameters similar to the raspberry pi camera. The updated model files and directions on how to incorporate them is available here,

https://github.gatech.edu/swilson64/turtlebot3_sim_update

If you wish, you can use GIT to clone this repository,

```
git clone https://github.gatech.edu/swilson64/turtlebot3_sim_update.git
```

Tips, Tricks, and Suggestions

- Launch files allow you to launch several nodes all at once. For example instead of running the turtlebot bringup and object follower code separately (requiring two or more ssh windows), a launch file will do both of these at once. It may be useful to create a launch file for this project and future labs. Instructions on how to do this can be found in the ROS2 tutorials here,

  https://docs.ros.org/en/humble/Tutorials/Intermediate/Launch/Creating-Launch-Files.html

- To capture images from the Raspberry Pi camera attached to the robot, please use the `v4l2_camera_node` for image acquisition in this lab

  ```
  ros2 run v4l2_camera v4l2_camera_node -ros-args -params-file ./v4l2_camera.yaml
  ```

  This ROS2 node grabs images from the camera on the robot and publishes them to the `/image/compressed` topic. The parameters that effect the resolution of the camera are found in the `v4l2_camera.yaml` file in the run command above. If you choose to operate at a different resolution, please make a new yaml file. **Do not change the default yaml file**. Alternatively, you can use a custom launch file we have created that brings up the robot with the camera, `ros2 launch turtlebot3_bringup camera_robot.launch.py`.

Networking and Quality of Service Profiles

The computational ability of the Raspberry Pi is much weaker than your computer's, meaning you most likely cannot process a high resolution image in real time. The launch file provided brings up the camera at 320×240 resolution.

Images may be captured faster than your robot can process them even at reduced resolution. Additionally, ROS2's networking has Quality of Service (QoS) settings that define how data should be passed. For example, ROS has a queue system where it will store a time series of images so none are missed. In some real time applications, this can be a problem as your robot will be acting on old data. It is useful to add a queue size argument in your subscriber/publisher initialization to limit the amount of stored data your callback function will act on. Additionally, due to networking lag, connections may be dropped or hang if the QoS profile is set as `"reliable"` when the WiFi connection is not steady. Details about QoS profiles can be found,

https://docs.ros.org/en/humble/Concepts/About-Quality-of-Service-Settings.html
and
https://docs.ros2.org/latest/api/rclpy/api/qos.html

The following code snippets can be used as a starting point to generate different QoS profiles within the `__init__` function your subscriber is declared in,

```
from rclpy.qos import QoSProfile, QoSDurabilityPolicy, QoSReliabilityPolicy, QoSHistoryPolicy

custom_qos_profile = QoSProfile(
            reliability=QoSReliabilityPolicy.BEST_EFFORT,
            history=QoSHistoryPolicy.KEEP_LAST,
            durability=QoSDurabilityPolicy.VOLATILE,
            depth=1
        )

sub = Subscriber(
            self,
            Image,
            "cool_image_topic",
            qos_profile=custom_qos_profile
        )
```

or

```
from rclpy.qos import QoSProfile, QoSDurabilityPolicy, QoSReliabilityPolicy, QoSHistoryPolicy

custom_qos_profile = QoSProfile(depth=1)

custom_qos_profile.reliability = QoSReliabilityPolicy.BEST_EFFORT
custom_qos_profile.history = QoSHistoryPolicy.KEEP_LAST
custom_qos_profile.durability = QoSDurabilityPolicy.VOLATILE

sub = Subscriber(
            self,
            Image,
            "cool_image_topic",
            qos_profile=custom_qos_profile
        )
```

Grading Rubric

The lab is graded out of 100 points, with the following point distribution:

Successfully run all code^o on the robot	15%
Robot rotates in the direction of the object consistantly*	50%
Communicate information between your two nodes	35%

^{o} All code does not include any debugging code you write for your personal computer.
* Consistantly means it does not frequently rotate the wrong direction due to noise/image delay, the robot will get false and noisy readings occasionally but we will not deal with these problems in this lab.

Submission

1. Perform a live demonstration of you running the robot to one of the course staff by the listed deadline.
2. Put the names of both lab partners into the header of the python script. Put your python script and any supplimentary files, in a single zip file called Lab2_LastName1_LastName2.zip and upload on Canvas under Assignments–Lab 2.
3. Only one of the partners needs to upload code.

We will set aside class time and office hours on the due date for these demos, but if your code is ready ahead of time we encourage you to demo earlier in any of the office hours (you can then skip class on the due date). Office hour times are listed on the Canvas homepage.