

CS/ME/ECE/AE/BME 7785

Lab 2

Due: February 11, 2022 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 ROS package containing two communicating nodes that enable the robot to turn in place to follow an object. You will 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 **roscore** and all files on-board the robot. You *should not* have images passed from the Turtlebot to your laptop for processing during the demo. To move folders from your computer to the robot, 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>
```

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 ROS 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 **rospy**, **roscpp**, **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 `/raspicam_node/image/compressed`. Example code in Python:

```
img_sub = rospy.Subscriber("/raspicam_node/image/compressed", CompressedImage, callback)
```

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 (either diamond or ball, whichever 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 user `rqt_image_viewer`).

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

```
pub = rospy.Publisher('/cmd_vel', Twist, queue_size=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. Your 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 11. 1. make sure you clone the branch made for ROS Melodic. This can be done with the command,

```
git clone -b melodic-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 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

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.
- To capture images from the Raspberry Pi camera attached to the robot, please use the raspicam_node launch file for image acquisition in this lab. You can also use the turtlebot3_camera_robot.launch file in the turtlebot3_bringup package that we have created to launch the robot and camera simultaneously.
- The computational ability of the Raspberry Pi is much weaker than your computer's, meaning you most likely cannot process a full resolution image in real time. To capture images at a reduced frame-rate and resolution, you may modify/create new launch files associated with the raspicam_node which should be located in
“\$(find raspicam_node)/launch/camerav2_AAxBBB.launch”.
- Images may be captured faster than your robot can process them even at reduced resolution. In this case, 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 (*queue_size = N*) in your subscriber/publisher initializations to limit the amount of stored data your callback function will act on. For subscribing to the raspicam_node image topic, $N=1$ will only apply the callback function to the most recently captured frame.

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 consistently*	50%
Communicate information between your two nodes	35%

^o All code does not include any debugging code you write for your personal computer.

* Consistently 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 supplementary 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.