Institute for Aerospace Studies
UNIVERSITY OF TORONTO

LABORATORY III

LIDAR Mapping with Wheel Odometry

ROB521 Mobile Robotics and Perception
Spring 2023

1 Introduction

This is the third laboratory exercise of ROB521-Mobile Robotics and Perception. The course will encompass a total of four labs and a design project, all of which are to be completed in the scheduled practicum periods. Each of the four labs will grow in complexity and intended to demonstrate important robotic concepts presented during the lectures. Our robot of choice: *Turtlebot 3 Waffle Pi* running the operating system *ROS*.

2 Objective

The objective of this lab is to develop wheel odometry pose estimates and use them to construct an occupancy grid map of the environment. In particular, you are

- *To learn about the robot's hardware and its suite of sensors*
- *To design and perform an experiment to calibrate the robot's wheel radius and baseline*
- *To implement a dead-reckoning robot motion estimator using wheel encoder data*
- *To construct and update an occupancy grid map using live scan measurements and pose estimates*

2.1 Lab Deliverables

In this (and future) labs, look for deliverables that will be in **bolded red text** throughout the manual, and follow the instructions for each. Additionally, a summary of the deliverables and their mark distribution is at the end of the document. **In this lab, you will submit a pdf lab report and a video demonstrating the deliverables. You will also need to demonstrate your mapping algorithm working on a Turtlebot during the in-person lab session.**

In your lab report, for each deliverable, include a short description of what your code is doing, and what the robot does as a result of running the code. Provide context that will allow the TA's to understand what your robot is doing, in order for them to properly gauge your success. Importantly, state whether you were able to complete the deliverable. If not, explain why you weren't able to complete the task. Do not write more than a paragraph for each deliverable.

Finally, include a copy of your code for the TA's to look over. The code itself will not be marked, but it must be presented to demonstrate that each group has independently written their solution. If the code does not look complete, TAs will be run the code themselves to confirm if the deliverables have been completed.

3 Getting Started

3.1 TurtleBot Topics

In the first lab we discussed the ROS topics subscribed to and published by our ROS Gazebo simulation. In this lab we will use three topics: `cmd_vel`, `odom`, `scan`, and `sensor_state`.

The `cmd_vel` topic uses a `twist` message type. The contents of this message are shown by Figre 1 used to set the linear and angular velocity of the robot.

```
linear:
  x: -0.000766990473494
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: -0.000899488280993
```

Figure 1: Example Twist message

The `odom` message provides a computed solution for the turtlebot3 based on the same wheel odometry method presented in class combined with estimates from the imu, and is accurate up to wheel slip (with some corrections from the imu) and teleportation by human carrier. The message contains the components shown in Figure 2. The pose is expressed with 7 parameters, a vector translation and quaternion rotation relative to the initialization frame at launch, whereas the velocity (twist) is expressed with 6 parameters, 3 linear and 3 angular.

The `scan` message contains the lidar scan data, updated at 5 Hz. An example is shown by Figure 3. A range measurement of inf in simulation means that the LIDAR detected no return signal - on the actual robot the reading is 0.0. Therefore, you can conclude that there are no objects between the robot and the max range measurement in that direction.

Finally, the `sensor_state` topic contains the raw sensor values for a range of sensors (Fig. 4). Of our concerns are two values `right_wheel` and `left_wheel` which are the number of encoder ticks that have been counted, by each wheel respectively, since the last published `sensor_state` message.

4 Assignment

4.1 Simulation

This section describes the implementation and simulation tasks. We strongly recommend you finish these tasks before coming to the lab session.

```
header:
  seq: 73
  stamp:
    secs: 1573792014
    nsecs: 715609947
  frame_id: "odom"
  child_frame_id: "base_footprint"
pose:
  pose:
    position:
      x: -0.153988704085
      y: 0.0590254813433
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: -0.371140569448
      w: 0.92857670784
  covariance: [0.0, 0.0, 0.0, ...]
twist:
  twist:
    linear:
      x: -0.000766990473494
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: -0.000899488280993
  covariance: [0.0, 0.0, 0.0, ...]
```

Figure 2: Example odom message

Task 1: Vehicle Calibration

Often, robots will come with factory measurements of important parameters, such as wheel radius and separation. *Lecture 14: Dead Reckoning and Wheel Odometry* demonstrated that these parameters are critical for wheel based odometry estimation. Occasionally, you may wish to verify these measurements so, you will design two experiments. One to verify the accuracy of the Tutlebot3's wheel radius and another to verify the wheel separation - these values are shown in Figure 5.

For this section, you will be designing the two experiments and implementing them in ROS. You will perform calibration on a real robot later in lab. During the lab, you will be provided tools to determine how many rotations a robot has performed or how far a robot

```
header:
  seq: 73
  stamp:
    secs: 1573792014
    nsecs: 715609947
  frame_id: "base_scan"
angle_min: 0.0
angle_max: 6.26573181152
angle_increment: 0.0174532923847
time_increment: 2.99000002997e-05
scan_time: 0.0
range_min: 0.119999997318
range_max: 3.5
ranges: [0.0, 0.0, 0.0, 0.0, 2.4110000133514404, ...]
```

Figure 3: Example scan message

```
header:
  seq: 419
  stamp:
    secs: 1677428862
    nsecs: 607415632
  frame_id: ""
bumper: 2
cliff: 197.0
sonar: 0.0
illumination: 541
led: 0
button: 0.
Torque: True
left_encoder: 20751
right_encoder: 29119
battery: 12.09000015258789
```

Figure 4: Example sensor_state message

has driven forward. Note that you can teleoperate the robot and have access to the encoder information provided in the `sensor_state` topic. The steps to complete these tasks are as follows:

- ***Design the experiments.*** *How would you drive a robot using teleoperation to determined the parameters?*
- ***Implement your method***. *To assist your development, code templates have been provided to you,* `l3_estimate_wheel_baseline.py` , `l3_estimate_wheel_radius.py` . *In both files, a node has been instantiated subscribing to the* `sensor_state` *and* `cmd_vel`

topic. The `sensorCallback` function processes the raw encoder data and provides you accumulated changes in encoder readings since the node starts. You will use this information and implement the estimator of wheel baseline and radius in the `startStopCallback` function. Note that on the top of each file, you will find some useful constants defined in ALL_CAPS.

- ***Sanity check**. Unfortunately, our Gazebo simulator has not been configured to simulate encoder. To check your implementation, a ROS bag (`sample_data.bag` has been provided to you. It was recorded on a real robot while it was commanded to move forward in a straight line (along the x-axis) for about 0.75 meters. It contains two topics, `sensor_state` and `cmd_vel` topic. To verify your code, first start your calibration node,*

```
$ rosrun rob521_lab3 l3_estimate_wheel_baseline.py
```

or

```
$ rosrun rob521_lab3 l3_estimate_wheel_radius.py
```

Next, play the ROS bag

```
$ rosbag play sample_data.bag
```

Check if there are any syntax errors and if your code outputs the expected values given how the robot was driven.

Congratulations! You have finished implementing the vehicle calibration task. Your next step is to test it on a real robot in your lab session. There are no deliverables for this section. You can proceed to the next simulation task.

Task 2: Estimate Robot Motion using Wheel Encoder

In this section of the lab, you'll implement your own dead reckoning pose estimator using wheel odometry data. In your lab session, you will compare your pose estimator with the TurtleBot onboard odomentry data in the `/odom` topic.

A template code has been provided to you, `l3_estimate_robot_motion.py` which creates a node that publishes the robot's pose and velocity in the form of a `nav_msgs/Odometry` message on a topic called `wheel_odom`. Now the message is empty. You are responsible for filling in the missing information, pose and twist. `Lecture 14: Dead Reckoning and Wheel Odometry` outlines the differential drive wheel model required for this task and the `/sensor_state` messages contains the encoder data required. The pose's parameters will be in reference to an inertial frame defined when the node initializes. The steps to finish this task are as follows

- ***Implement the two missing sections***
  - *Based on the conversion from ticks to radians, the radius and baseline determined in calibration and the wheel encoder counts, compute a translation in x and y and a rotation about the z axis for the Turtlebot. Note that you can use any integration method you like.*
  - *Calculate the velocity of the robot at using the latest pose change of the robot.*
- ***Sanity Check.*** *Again, you can not fully verify your code in simulation since this task also depends on the encoder data missing from our simulator. Instead, you will be using the ROS bag,* `sample_data.bag` *for a quick check.*
  First, start the estimation node,

```
$ rosrun rob521_lab3 l3_estimate_robot_motion.py
```

Then, play the bag

```
$ rosbag play sample_data.bag --pause
```

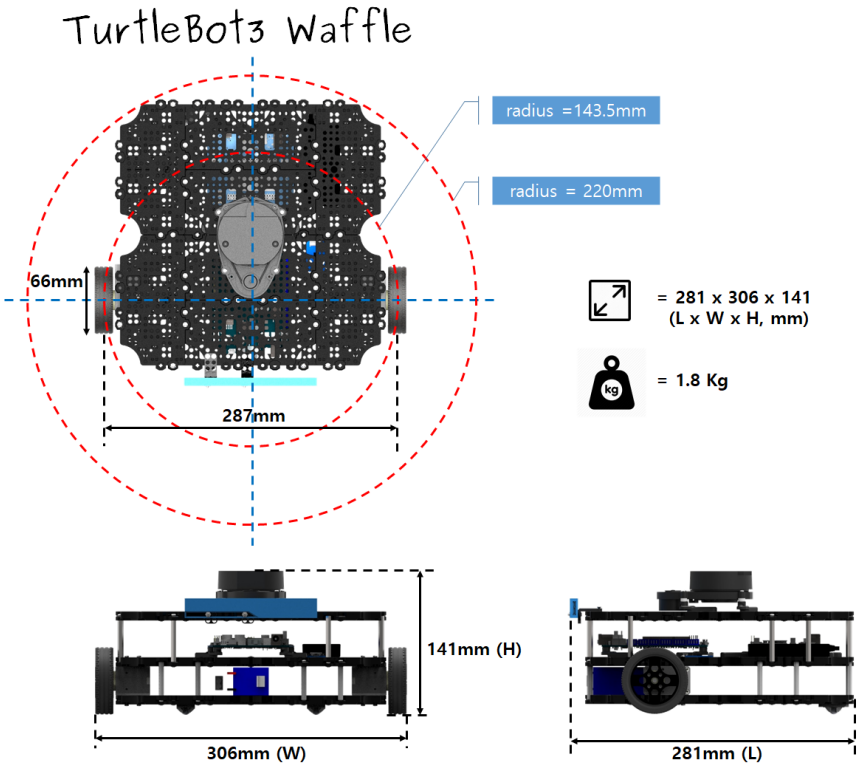

Figure 5: Waffle Pi dimensions.

Note that the bag does not contain the `odom` topic. You can use the fact the robot was driven forward (along the x-axis) for roughly 0.75 meter as a baseline. Note that your estimator does not need to match this number.

You will finish testing during your lab session. There are no deliverables for this section. You can proceed to the next simulation task.

Task 3: Construct an Occupancy Grid Map

In this section, you will be developing an occupancy grip map of the environment near the robot, based on the estimate of vehicle motion from the onboard odometry measurements. You will be testing in simulation.

Using the template code provided, you will create a node that publishes a grid map on the map topic. The lecture on Grid-based Mapping and Localization outlines the how to fill in a grid map using a sensor that scans the environment. You will need to fill in two functions in *l3_mapping.py* to create a grid map. The steps to completing this task are as follows:

- *If you are not using one of the lab computers you will need to download and place the lab2 folder in accordance with how your catkin workspace is constructed. Additionally, you will need to run two commands `catkin_make` and `rospack profile` from your catkin workspace folder.*
- *In the terminal, enter the command `roslaunch rob521_lab3 turtlebot3_world.launch`. This command will start the simulator with the turtlebot and the environment. You can reset the robot position in the gazebo simulator at anytime by pressing the reset models button in the gazebo simulator.*
- *In another terminal window, enter in `roslaunch rob521_lab3 mapping_rviz.launch`. This will launch an rviz window that will display your map and the live 2D lidar data. Note that the **x-axis of the map points in the same direction as the x-axis (forward direction) of robot when you start the node**.*
- *In a third terminal window, enter in the command `rosrun rob521_lab3 l3_mapping.py`. If this command does not work, you may need to navigate to the folder containing `l3_mapping.py` and run the command `chmod +x l3_mapping`. This command will give the `l3_mapping.py` file the permission to run as an executable. This command starts the mapping node that you will fill in, so it won't do much for now. You will need to end this program and restart it as you make changes to the `l3_mapping.py` file.*
- *In the `l3_mapping.py` file, the `scan_cb` function is automatically called whenever new lidar data is available. In the `scan_cb` function, you will call `ray_trace_update` for each element in `scan_msg.ranges`. **Pay attention to the input variables required by `ray_trace_update`**. For more information about the structure and contents of scan_msg messages, consult Figure 3 or google "sensor_msgs LidarScan".*

  - *For debugging, you may find it easier to start with only updating the map for a single or a few lidar rays.* ***Note that the first lidar beam in the message points directly in front of the robot (x-axis of the robot), and each subsequent beam moves in a counter-clockwise direction with an angle change equal to*** `scan_msg.angle_increment`.

- *Now you will complete the* `ray_trace_update` *function. This function will need to update the pixels in the map that are along the measured LIDAR ray. We have imported a function ray_trace that will return the indices of the pixels in the map that belong to the LIDAR ray. For more information on this function, consult scikit-doc. Use the method outlined in the lecture on Grid-based Mapping and Localization to update the log odds map (self.log_odds), and subsequently update self.np_map with actual probabilities stored as integers between 0 and 100. After you've updated* `self.np_map`*, we have included code to convert it to the form that ROS expects for the actual map message.*
- *If you run your node, you should notice that rviz will show that the lines of the map around the robot have been filled in as in Fig.6. If your code is correct, then the empty pixels between the robot and the object should be white, the pixels with the object in it should appear black, and the pixels behind the object (unknown) should appear grey.*
- *If you kill your mapping node, a figure, named map.png, will be stored under the lab3 folder. You will need to include it in your report once you finished testing.*
- *Now use* `roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch` *to launch the Turtlebot3 teleoperation nodes. Start to rotate the robot, does your map still make sense? If not, check the angle that you are passing into the ray_trace_update.*
- *Congratulations, you have a working grid mapping solution! Using the teleop functions, experiment with different driving motions and obstacle densities until you are able to build a map. During map creation, what failure modes do you notice? Note the simulated odometry measurements have noise added to them.*

For this section, the deliverables are the following:

- A video of your the rviz window as you map the environment. The start of the video should start with a map that has not been updated with any Lidar measurements. During the video, drive the robot in a way to map the closed perimeter of the simulation environment. (1 pt)
- In your report, you must include the following:
  - Include a picture of the mapped environment - similar to the video the mapped environment only needs to show the closed perimeter of the simulation environment. (0.5 pt)

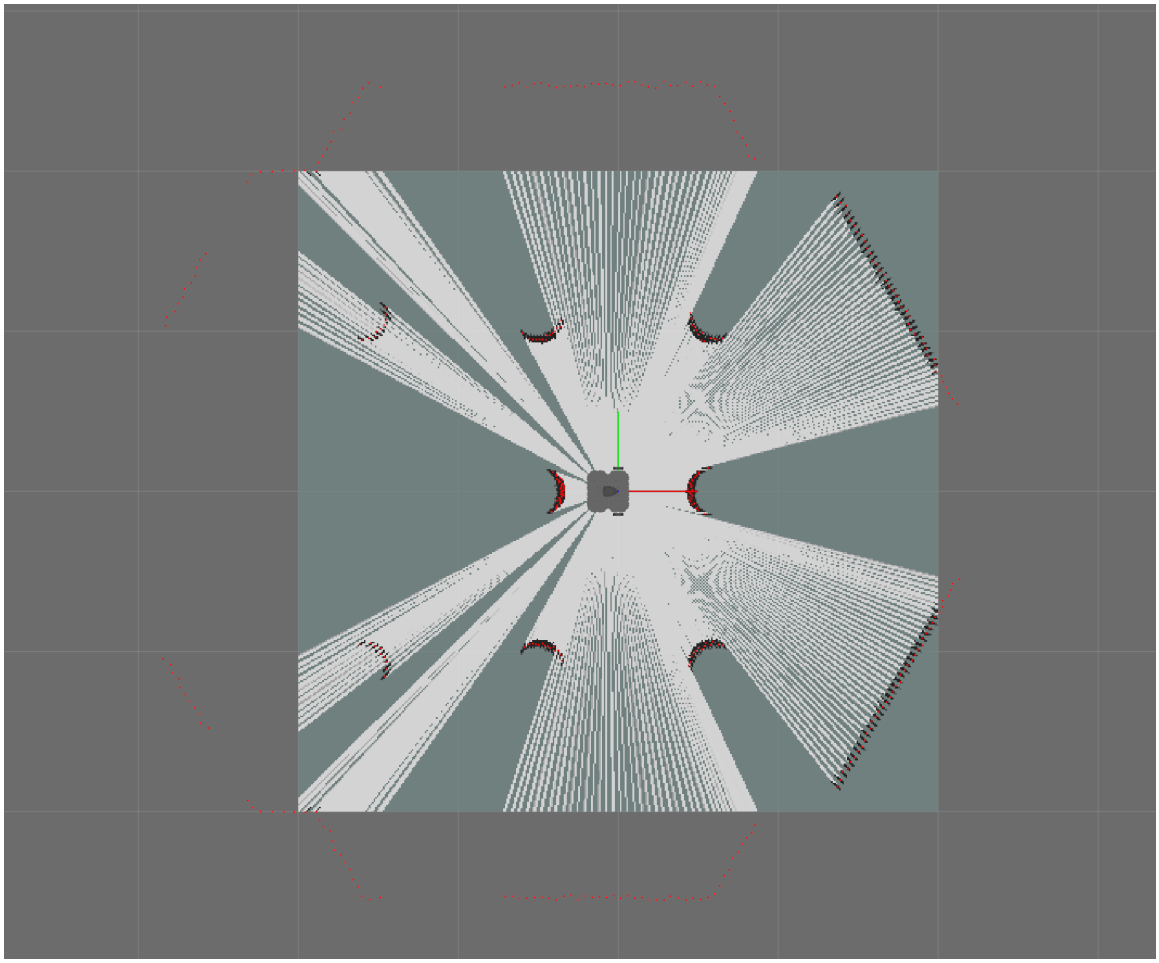

Figure 6: An example of Rviz showing Lidar measurements (red dots) and an occupancy grids map.

- Describe how your code works or should work. (1 pt)
- Explain a potential source of error in this mapping algorithm. (0.5 pt)

Helpful Notes Don't forget to use the constants defined in ALL_CAPS at the top of the file! This is where we define the ALPHA and BETA for updating log_odds, the MAP_DIM in width and height in meters, the CELL_SIZE indicating meters per pixel, NUM_PTS_OBSTACLE indicating how many pixels at the end of a lidar beam to consider part of the obstacle, and SCAN_DOWNSAMPLE to how many lidar beams to increment when you loop through them all (=1 means use all data, i.e. increment one at a time). You are free to experiment with other parameters for these fields, but you may notice degraded performance if you try to make the resolution too high.

4.2 Experiments

Tasks in this section are to be finished during the lab session.

Task 4: Calibrate the TurtbleBot

This is the first experiment you will do during the lab session. You will carry out the two calibration experiments you designed earlier and determine the wheel radius and baseline parameters. These parameters will be used for the next task to estimate robot motion.

In your report, describe your experiments. Your summary must address these points:

- What path the robot was driven in each experiment. (1 pt total)
  - (0.5 pts) How did you drive the robot to determine the wheel radius? How much did it rotate? How far did you drive forward?
  - (0.5 pts) How did you drive the robot to determine the wheel separation? How much did you rotate? How far did you drive forward?
- How does your code work or should work? How are the parameters determined? (0.5 pt total)
- What values did you get for the wheel radius and baseline? (0.5 pt total)
- Does these values match those given in the data sheet? Identify one possible source of uncertainty or bias that made your answer differ from the factory calibration. (1 pt total)
  - Identify the source of uncertainty or bias.
  - How does the source of uncertainty or bias affect your measurement?
  - How could you mitigate this source of uncertainty or bias?

Task 5: Motion Estimation

In this section, you will test your motion estimator on a real robot. You will drive the robot using teleoperation and compare your estimates (the `/wheel_odom` topic) against the one provided by the TurtleBot (the `/odom` topic).

In particular, you will need two experiments. First, you drive in a circle with radius approximately 1m and return to the same starting point. Confirm that the pose is close to the origin through visualization in rviz and by uncommenting the print statements in `l3_estimate_robot_motion_solution.py`. Repeat for a more complex path and identify motions that lead to significant failures of odometry.

To run experiments, first on the TurtleBot PC, run

```
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch
--screen
```

Then, on Remote PC, start your state estimation node

```
$ rosrun rob521_lab3 l3_estimate_robot_motion.py
```

Before you continue, check the outputs in the terminal. Both your and TurtleBot's estimates should be close to zeros.

To visualize your odometry estimate compared with the existing one from the turtlebot3 package, run

```
$ roslaunch rob521_lab3 wheel_odom_rviz.launch
```

which will start a visualization package called rviz. Rviz has been configured to allow you to visualize your `/wheel_odom` pose (marked `wo_base_link`) and the Turtlebot's `/odom` pose (marked `base_link`, with the robot model on top as well).

Now you can drive the robot,

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Once you close the estimator node, a rosbag `motion_estimate.bag` will be saved under your lab3 folder. It contains both your and the onboard pose estimates. A script has been provided to you to plot the estimated trajectory,

```
$ rosrun rob521_lab3 l3_plot_motion_estimate.py
```

You will need this figure for your report. Note that the data is written to the same rosbag. You may want to change the bag name for your second experiment in `l3_estimate_robot_motion_solution.py`. You will need to change the name accordingly in `l3_plot_motion_estimate.py` for visualization.

For this section, the deliverables are the following:

- In your report:
  - Include two plots from the two experiments comparing your odom estimates with the onboard one. (1 pt)
  - Briefly discuss the results (1 pt)
    - (0.5 pt) How is your estimation compared to the onboard one? Name one possible source of error that accounts for the differences.
    - (0.5 pt) How are the estimates compared to the actual trajectory you observed? Name one possible source of error that accounts for the differences.

Task 6: Mapping

For this task, you will test your mapping algorithm on a real robot. A maze will be set up in Myhal 570 and you will be asked to create a map for it.

To run experiments on a real TurtleBot3, first on the TurtleBot PC, run

```
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch
--screen
```

Then, on Remote PC, run

```
$ roslaunch rob521_lab3 mapping_rviz.launch
```

Also, on Remote PC, start your mapping node

```
$ rosrun rob521_lab3 l3_mapping.py
```

Finally, to teleoperate your robot,

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

For this section, the deliverables are the following:

- Live mapping demo for your TA (1 pt)
- In your report,
  - Include a picture of a map of the environment. (0.5 pt)
  - Explain a potential source of error that did not present in simulation. (0.5 pt)

5 Concluding Remarks

This lab exposes you to the implementation challenges of getting two commonly used algorithms working in the ROS environment and on a reliable and well-designed robot. It hopefully solidified the limitations of occupancy grid mapping based on odometry pose estimate, where errors accumulate over time and must be undone gradually with repeated new measurements. You should now be very excited for the next lab, localization and mapping, where we develop more sophisticated methods of correcting for error accumulation over time.

6 Deliverable Summary

The deliverables are listed below for a total of 10 points.

- A video of your the rviz map as you map the environment in simulation. At the start of your video, the map should be completely uninformed - no Lidar mapping updates should have occurred. During your video, drive the robot in a way to map the closed perimeter of the simulation environment. (1 pt)
- Live mapping demo for your TA (1 pt)
- Report Part 1: Vehicle Calibration
  - What path the robot was driven in each experiment. (1 pt total)
    - (0.5 pts) How did you drive the robot to determine the wheel radius? How much did it rotate? How far did you drive forward?
    - (0.5 pts) How did you drive the robot to determine the wheel separation? How much did you rotate? How far did you drive forward?
  - How does your code work or should work? How are the parameters determined? (0.5 pt total)
  - What values did you get for the wheel radius and baseline? (0.5 pt total)
  - Does these values match those given in the data sheet? Identify one possible source of uncertainty or bias that made your answer differ from the factory calibration. (1 pt total)
    - Identify the source of uncertainty or bias.
    - How does the source of uncertainty or bias affect your measurement?
    - How could you mitigate this source of uncertainty or bias?
- Report Part 2: Motion Estimation
  - Two plots from the two experiments comparing your odom estimates with the onboard one. (1 pt)
  - Briefly discuss the results (1 pt total)
    - (0.5 pt) How is your estimation compared to the onboard one? Name one possible source of error that accounts for the differences.
    - (0.5 pt) How are the estimates compared to the actual trajectory you observed? Name one possible source of error that accounts for the differences.
- Report Part 3: Occupancy Grid Mapping
  - Include a picture of the mapped environment - similar to the video the mapped environment only needs to show the closed perimeter of the simulation environment. (0.5 pt)
  - Describe how the sections you coded works or should work. (1 pt)
  - Explain a potential source of error in this mapping algorithm. (0.5 pt)
- Report Part 4: Mapping on a Real Robot

- Include a picture of a map of the environment. (0.5 pt)
- Discuss the results. Explain a potential source of error that did not present in simulation. (0.5 pt)

7 Additional Resources

1. Introduction to ROS, ROB521 Handout, 2019.
2. Robotis e-Manual, `http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/`.
3. "SSH: Remote control your Raspberry Pi," The MagPi Magazine, `https://www.raspberrypi.org/magpi/ssh-remote-control-raspberry-pi/`.
4. Official ROS Website, `https://www.ros.org/`.
5. ROS Wiki, `http://wiki.ros.org/ROS/Introduction`.
6. Useful tutorials to run through from ROS Wiki, `http://wiki.ros.org/ROS/Tutorials`.
7. ROS Robot Programming Textbook, by the TurtleBot3 developers, `http://www.pishrobot.com/wp-content/uploads/2018/02/ROS-robot-programming-book-by-turtlebo3-developers-EN.pdf`.