Institute for Aerospace Studies
UNIVERSITY OF TORONTO

LABORATORY III

LIDAR Mapping with Wheel Odometry

ROB521 Introduction to Robotics
Spring 2022

1 Introduction

This is the third laboratory exercise of ROB521-Autonomous Mobile Robotics. 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 an experiment to calibrate the robot's wheel radius and baseline*
- *To construct and update an occupancy grid map using live scan measurements and pose estimates*

2.1 Lab Deliverables

Since students cannot demonstrate functionality to the TA during in-person labs, as is usually done, you will be required to demonstrate functionality by recording screen captures and submitting a report.

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**.

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, TA's will have access to all of the computers, and will be able to run the code themselves to confirm that 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`, and `scan`.

Subscribed Topics

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

3.1.1 Published Topics

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.

Note: 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.

Finally, the `scan` message contains the lidar scan data, updated at 5 Hz. An example is shown by Figure 3.

Note: 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.

```
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

4 Assignment

Task 1: Vehicle Calibration Verification Experiment Design

Often, robots will come with factory measurements of important parameters, such as wheel radius and separation. *Lecture 7: 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 4. For this section, you will not need to code these experiments in ROS - you will only be designing the experiments.

```
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

For your experiment, the testing environments and the robots have been setup for you. The first environment lets you accurately measure how far the robot has traveled in a straight line. In the second environment, you are able to accurately measure how many rotations a robot has performed. The robot subscribes to the `cmd_vel` topic. Additionally, the robot publishes on a topic `encoders`, which contains two values `right_wheel` and `left_wheel`. The values `right_wheel` and `left_wheel` are the number of encoder ticks that have been counted, by each wheel respectively, since the last published `encoders` message. The equations required to calculate the wheel radius and separation using the change in encoder ticks are defined in *Lecture 7: Dead Reckoning and Wheel Odometry*.

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

- What path the robot will drive in each experiment. (1 pt total)
  - (0.5 pts) How will you drive the robot to determine the wheel radius? How much will you rotate? How far will you drive forward?
  - (0.5 pts) How will you drive the robot to determine the wheel separation? How much will you rotate? How far will you drive forward?
- How many encoder ticks do you expect to count from each wheel in your experiments? Make sure to highlight any assumptions that you make. The Turtlebot3's left and right encoder counts have a TICK2RAD conversion of 0.001533981 (aka the number of radians per tick, or 2*pi / 4096). (1 pt total)
  - (0.5 pts) How many encoder ticks do you expect to count in your wheel radius experiment?

  - (0.5 pts) How many encoder ticks do you expect to count in your wheel separation experiment?

- Identify at least two sources of uncertainty or bias that could make your answer differ from the factory calibration. (3 pts total)
  - (0.5 pts) Identify the source of uncertainty or bias.
  - (0.5 pts) How will the source of uncertainty or bias affect your measurement?
  - (0.5 pts) How could you mitigate this source of uncertainty or bias?

Task 2: Construct an Occupancy Grid Map

In the final 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. 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 *l2_mapping.py* to create a grid map. The steps to completing this task are as follows:

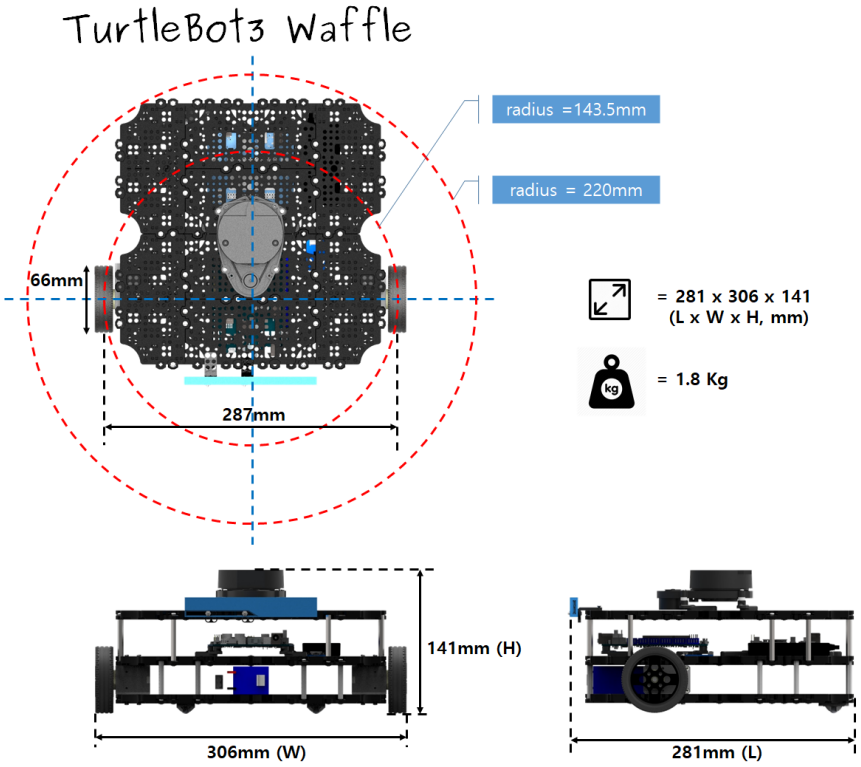

Figure 4: Waffle Pi dimensions.

- *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_lab2 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_lab2 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_lab2 l2_mapping.py`. *If this command does not work, you may need to navigate to the folder containing* `l2_mapping.py` *and run the command* `chmod +x l2_mapping`. *This command will give the* `l2_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* `l2_mapping.py` *file.*
- *In the* `l2_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, 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* `https://scikit-image.org/docs/dev/api/skimage.draw.html#skimage.draw.line`. *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. 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.*
- *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 map 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. (1 pt)
  - Describe how the sections you coded works or should work. (1 pt)
  - Explain a potential source of error in this mapping algorithm. (1 pt)
  - Explain how you could correct or prevent the source of error that you previously identified. (1 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.

5 Concluding Remarks

This lab exposes you to the implementation challenges of getting two commonly used algorithms working in the ROS environment with 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 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. 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)
- Report Part 1: Vehicle Calibration
  - What path the robot will drive in each experiment. (1 pt total)
    - (0.5 pts) How will you drive the robot to determine the wheel radius? How much will you rotate? How far will you drive forward?
    - (0.5 pts) How will you drive the robot to determine the wheel separation? How much will you rotate? How far will you drive forward?
  - How many encoder ticks do you expect to count from each wheel in your experiments? Make sure to highlight any assumptions that you make. The Turtlebot3's left and right encoder counts have a TICK2RAD conversion of 0.001533981 (aka the number of radians per tick, or 2*pi / 4096). (1 pt total)
    - (0.5 pts) How many encoder ticks do you expect to count in your wheel radius experiment?
    - (0.5 pts) How many encoder ticks do you expect to count in your wheel separation experiment?
  - Identify at least two sources of uncertainty or bias that could make your answer differ from the factor calibration. (3 pts total)
    - (0.5 pts) Identify the source of uncertainty or bias.
    - (0.5 pts) How will the source of uncertainty or bias affect your measurement?
    - (0.5 pts) How could you mitigate this source of uncertainty or bias?
- Report Part 2: 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. (1 pt)
- Describe how the sections you coded works or should work. (1 pt)
- Explain a potential source of error in this mapping algorithm. (1 pt)
- Explain how you could correct or prevent the source of error that you previously identified. (1 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`.