Lab III: LiDAR Mapping With Wheel Odometry

University of Toronto
ROB521

March 2nd, 2022

Alvin Pane [1004281118]
Isobel Lees [1003912484]
Thanos Lefas [1003927334]
Alaynah Malik [1003099032]

Introduction

The objective of this lab was to learn about the robots hardware and sensors by using wheel odometry pose estimates to construct an occupancy grid map of a simulated environment. Odometry is the process of estimating motion and position from wheel encoders. The lab is split into 2 tasks. The first task involves designing an experiment to calibrate the robot's wheel radius and wheel separation. The second task was to create and update an occupancy grid map through simulated LiDAR scan measurements and odometry pose estimates.

Task 1: Vehicle Calibration Verification Experiment Design

Part A: Verify Wheel Radius

Experimental Design

The Dead Reckoning and Wheel Odometry lecture suggests that we drive the robot a certain distance, x_{true} (ex. 10 m) and using the wheel encoders measure $\Delta\varphi_r(kh), \Delta\varphi_l(kh)$ (the angle rotated during the kth incremental time period). Because there is no rotation, the angle $\theta = 0$ (the heading of the robot in the inertial frame), and the forward kinematics model below:

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} r/2 & r/2 \\ r/2b & -r/2b \end{bmatrix} \begin{bmatrix} \dot{\varphi}_r \\ \dot{\varphi}_\ell \end{bmatrix} \begin{matrix} \text{right wheel speed} \\ \text{left wheel speed} \end{matrix}$$

can be rearranged to obtain an equation to calculate the radius, r.

$$\underbrace{x_{\text{true}}}_{\text{from tape measure}} = \sum_k \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} r/2 & r/2 \\ r/2b & -r/2b \end{bmatrix} \underbrace{\begin{bmatrix} \Delta\varphi_r(kh) \\ \Delta\varphi_\ell(kh) \end{bmatrix}}_{\text{from wheel encoders}}, \quad r = 2\frac{x_{\text{true}}}{\sum_k (\Delta\varphi_r(kh) + \Delta\varphi_\ell(kh))}$$

The only slight modification we might make to this procedure is driving the robot forward for a specified time and then measuring the distance (because this is easier to code in ROS). Another method (perhaps to verify the radius calculated via the method specified in lecture) would be to drive the robot at a specified wheel rotational velocity (ex. 0.1 rad/s) for a specified time, then measure the distance and use this to calculate the wheel radius. However, this second method relies on the turtlebot being able to very accurately control the rotational velocity of its wheels, so the first method is far more feasible. The experiment should be repeated a few times to check for consistency.

Expected Ticks

The tick2rad conversion ratio is $2 * \pi / 4096 = 0.001533981 \, rads/tick$. Thus there are $1/0.001533981 \approx 652$ ticks in one radian (one wheel radius), and 4096 ticks in one full rotation of the wheel. The distance traveled is equal to $x_{true} = 2\pi r(number\ of\ rotations)$. So the number of encoder ticks expected from each wheel during the experiment is $ticks = 4096\frac{x_{true}}{2\pi r}$, where r is the wheel radius. If we tell the robot to travel 10m, given the wheel radius is 33 mm, we would expect 1.975×10^5 ticks.

Part B: Verify Wheel Separation

Experimental Design

To perform this experiment, we will need a method to drive the robot to perform 10 pure rotations. This can be done through marking a point on a wall, and then attaching a laser pointer to the robot that lines up with this point. We can then publish a purely rotational velocity twist and rotate the robot until the laser pointer lines up with the mark on the wall 10 times, ensuring 10 full rotations. This gives us a value for θ_true of N * 2π = 20π. Similar to the test for wheel radius, the forward kinematic model can be rearranged with x and y set to 0 (no translation) to obtain an equation to solve for the wheel separation, b, in terms of the wheel radius (r), θ_true, and the wheel encoder measurements.

$$b = \frac{r}{2}\frac{\sum_k (\Delta\varphi_r(kh) - \Delta\varphi_\ell(kh))}{N \times 2\pi}$$

$\Delta\varphi_r(kh), \Delta\varphi_l(kh)$ can be measured directly from the encoders topic, which return the number of ticks counted by each wheel respectively. Since the wheel separation is constant, we anticipate that it should not vary between experimental runs. Thus, it would be prudent to repeat this experiment multiple times, for both left and right rotations, to verify that we are indeed getting a result that is consistent.

Expected Ticks

When rotating in place, we anticipate the motion of the wheels to travel along the circle with diameter equal to the wheel separation, b, obtained from the above equation. We can get the circumference of the circle from the expression b*π. As we are rotating 10 times, the total distance traveled by the wheels will be equal to 10 times this circumference, 10*b*π. The tick2rad conversion given is $2\pi/4096$ rad/tick , meaning $4096/2\pi r$ ticks/m. We can translate this into an expression for the number of ticks in terms of the wheel separation: $ticks = (4096/2\pi r) * 10\pi b = 20480 * b/r$. If the wheel separation is 287 mm, and wheel radius is 33 mm we would expect 1.78×10^5 ticks. It is also important to note that the robot is turning on the spot, and so the left and right wheels will be traveling in opposite directions.

Part C: Sources of Error

The major source of uncertainty in this experiment is wheel slipping. When a twist command is being issued, it is possible that the rotation of the wheel will not fully translate to motion of the robot, due to unequal floor contact and variable friction. This would likely cause the distance travelled by the wheels to seem greater than it is meaning that the number of ticks recorded would be less than they should for the distance or rotation measured. Thus wheel radius and wheel separation would be calculated to be larger and smaller respectively than their true values. This uncertainty could be estimated by performing the calibration on a specific surface; however, different types of motion can result in various slipping models, and thus it is more appropriate to consider this a non-deterministic model. The best way to mitigate this uncertainty is probably to prevent slipping in the first place - use a hard surface (i.e. not loose like sand), with a high coefficient of friction (i.e. tarmac, not ice).

A form of bias in this setup is the possibly unequal wheel diameters. A difference in diameters may result in an additional rotational motion and result in significant accumulated positioning errors. Depending on the tested motion, this could make the calculated wheel radius and separation incorrect. It is possible to account for this error by testing the degree of curvature corresponding to a linear motion command, and adjusting the kinematic model accordingly.

Another potential source of uncertainty is the encoders. The turtlebot uses absolute encoders which may have small quantization errors resulting in imprecise measurements (although they won’t miss a count, like incremental encoders). However, this error is very small, and encoders these days are highly accurate so it is unlikely to be a problem.

Some other errors are listed below. One issue is limited resolution during integration of motion, which could be improved by using a better model (ex. Simpson's Rule instead of trapezoidal approximation). Another is a variation in the contact point of the wheel which is a non-deterministic error and thus can not be accounted for except by improving the quality of the apparatus. There will also be minute variations caused by random factors such as variations in the floor surface, wind, electrical-mechanical systems within the robot, and even human error in the measurement of the distance traveled by the robot. These can be dealt with by running multiple experiments (a good idea in any case) and taking the average.

Task 2: Constructing an Occupancy Grid Map

Mapped Environment

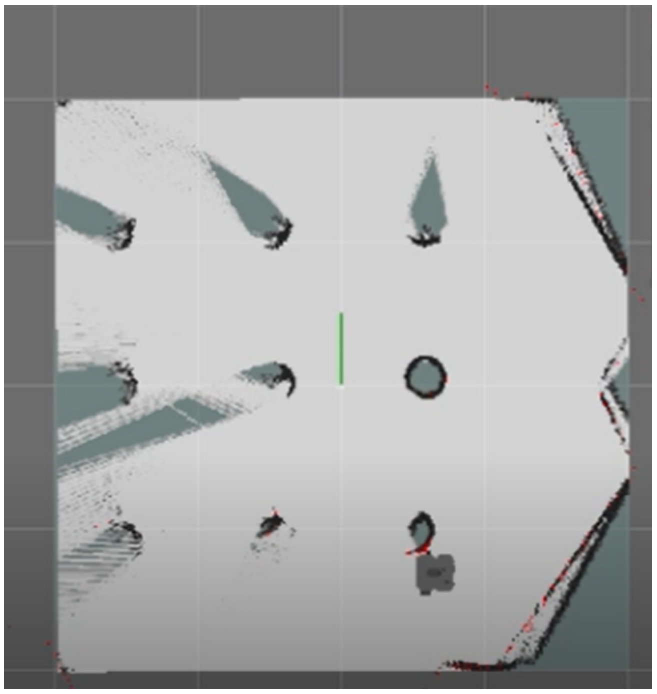

Figure 1: Picture of Mapped Environment

Description of Code

The scan_cb function was completed by first calculating the robot position on the map in pixels. Then we iterated through all LiDAR measurements available (in scan_msg.ranges). For each, its range value was converted to pixels, and the angle the LiDAR beam made with the horizontal x-axis was calculated. With this information, the ray_trace_update function could be called for each measurement, to update the map and the log_odds values.

The ray_trace_update function first calculated the ending position of the LiDAR beam in pixels. Using the starting and ending position the ray_trace or line function was used to determine the cells that the beam passed through. Each of these cells (as long as they were within the bounds of the image) was determined to be free space until the three (or NUM_PTS_OBSTACLE) cells at the end, which were considered to be an obstacle. Using these designations, the log_odds could be updated for each cell in accordance with lecture formulas:

$$l(c_k|y^t, x^t) \approx l(c_k|y^{t-1}, x^{t-1}) + \ln\left(\frac{p(y_t|c_k, x_t)}{1 - p(y_t|c_k, x_t)}\right)$$

$$l(c_k|y^t, x^t) \approx l(c_k|y^{t-1}, x^{t-1}) + \begin{cases} \alpha \text{ if cell in FOV and near measured range} \\ -\beta \text{ if cell in FOV and closer than measured range} \\ 0 \text{ otherwise} \end{cases}$$

The map probability could then be calculated from the log odds:

$$p(c_k|y^t, x^t) = \frac{\exp\left(l(c_k|y^t, x^t)\right)}{1 + \exp\left(l(c_k|y^t, x^t)\right)}$$

Finally, the updated map and log_odds were returned to the scan_cb function.

Robot Behavior

We seem to have completed the objectives of this deliverable as the robot behaved as expected. When travelling through the environment the robot only maps what is has a visual on and it does so by having the map show obstacles as black, with a pixel thickness of 3 (NUM_PTS_OBSTACLE). The empty pixels in between the robot and obstacles show up as white and the unmapped areas remain grey.

Error Sources

The mapping depends on the position of the robot, as determined by wheel odometry which accumulates error as the robot travels over time. Thus, the error in the map will increase as the robot continues to travel around the space. Odometry is a predictive source of data, so by combining it with a corrective data source, such as GPS, we can use sensor fusion (perhaps via a Kalman Filter) to remove some of the error and achieve the optimal combination of sensor data. If indoors or if the application requires high accuracy, we could try mounting a positioning system (similar to GPS, except instead of satellites we fix receivers at known coordinates in the environment).