COSC 081/281 - Principles of Robot Design and Programming (Term: Spring, Year: 2023)

Assignment: PA4 - Report

Author: Carter Kruse

Date: May 10, 2023

Implementation / Description

Overview

The purpose of this assignment is to expand familiarity with the real robot, develop basic behaviors, and identify the accuracy of different odometric schemes, together with mapping. Familiarity with ROS, publishers, and subscribers is a prerequisite. The following report will be split into sections corresponding to the different elements of this assignment.

Section 1

The program includes three functions that are responsible for the translation and rotation of the robot. The *translate* function takes one argument (the distance to be traveled in meters). The relative/absolute *rotate* functions each take one argument (the angle of rotation and the target angle, respectively).

To appropriately translate/rotate the robot, we determine the time needed for the robot motion, according to the linear and angular velocities, respectively. After implementing these basic behaviors, the following patterns are performed.

- Straight Line (1m)
- Rotation (Relative) (± 30º)
- Square (Side - 1m) → An extra rotation is included to end at the same orientation.

Following this implementation, the code was run for the robot several times for each pattern. To determine the results, manual estimation involved using a measuring tape and protractor (respectively), along with results from the laser sensor on the robot distance traveled/rotation. The odometry metrics were further collected to gain a deeper understanding as to the motion of the robot.

To achieve the measurement through the laser sensor, an environment is set up to allow the robot to perceive the shapes and the robot is positioned accordingly. This provides the relevant information to characterize the noise of the motion model. The average/mean error and variance were calculated to further characterize the data.

Link → Data

The link above includes the collected raw data and the processed results. To determine the results from the rotation using the laser scan, trigonometry is utilized, according to the image on the right. For specifics, view the formulas for each of the cells. The values correspond to the appropriate raw data.

In particular, for the *square* implementation, there are a few items of significance for the testing of this. When performing translations, followed by rotations, followed by translations, there is propagation of error, which indicates that at every time step, the robot moves further from its expected course. This reflects the idea of "drift" in a robot, which may be due to friction or other factors, and highlights the significance of using a closed-loop control (or various other methods) to ensure the robot does not go off course.

The data collected for the straight line and rotation display the following results. The manual measurements had the greatest variance, followed by the laser scan, followed by the odometry, indicating the relative levels of consistency. Further, in instances, the manual data more precisely approximated the commanded movement, indicating that the odometry and laser scanner may be prone to error in measurement.

Qualitative Observations

In testing, there were various qualitative observations that confirmed the previously published quantitative results. In particular, the robot tends to drift when following a straight line, either to the left or to the right, depending on the relative strength of each of the motors. Further, when performing rotations, the robot is nearly always slightly off, as there is a significant amount of friction between the wheels and the surface on which the robot is placed.

Section 2

The program implements occupancy grid mapping, using the laser scan data, as the robot moves around. The occupancy grid is with respect to the 'odom' reference frame, and the resultant map is published via a message through the relevant publisher and topic. As indicated by the documentation, the occupancy grid has three states: unknown, free space, and obstacles.

Prior to testing with the actual robot, the program was implemented in the simulation, which allowed for the appropriate refinement. The world used matches that of a previous world from a past programming assignment. The ECSC was partially mapped using the real robot, which will be discussed at a later point in time. To map the real world, teleoperation was used to perform the mapping, as opposed to creating an autonomous navigation algorithm.

Visualize Occupancy Grid

To visualize the resulting occupancy grid that was produced, individuals are able to either utilize *rviz* or the script written that runs on *matplotlib*. This script was designed with the following goal in mind: allow the robot to finish its movement and save the occupancy grid, so that it may be viewed at a later point in time. To use this, the script may be run in the same folder as the files; it does not require the use of ROS.

Design Decisions

In developing the program, various design decisions were made, primarily to ensure the accuracy/completeness of the code. Specifically, the linear velocity and angular velocity had to be closely set, so that the robot would rotate or move as expected, allowing room for error.

In the implementation of this particular assignment, the linear/angular velocity was used alongside time in an open-loop style that allows the robot to operate (mostly) as expected. By using this method, as opposed to a closed-loop, we are unable to ensure that the error in the robot's movements does not carry forward, which greatly impacts the positioning of the robot. This was what was discussed in the implementation of the *square* algorithm, which simply uses four translations, each followed by a rotation of 90º (pi / 2).

The linear velocity was set to a reasonable speed for the ROS Turtlebot, at 0.1 m/s, and the angular velocity was set to pi/8 rad/s.

Main Class

The main class, while relatively simple, contains elements that are slightly more complex, which will be explained as follows. This accounts for each of the Python scripts, as they are based on similar methods. The constants are initialized at the top of the class, followed by an initialization function for the class, which is used to set the publishers, subscribers, linear velocity, angular velocity, current position and orientation (*x*, *y*, *yaw*), scan measurement, and occupancy grid (map) info/data. The resolution for the map is further provided.

The *move()* and *stop()* methods are responsible for creating and publishing the Twist messages corresponding to given linear and angular velocities. The *translate()*, *rotate_rel()*, and *rotate_abs()* methods are simple structures that implement the open-loop design previously mentioned. The *odom_callback()* method is responsible for handling the Odometry data, setting the position/orientation of the robot.

The *laser_callback()* method functions differently, depending on the ROS node we are considering. The ROS node corresponding to the basic testing involves simply determining the laser scan measurement directly in front of the robot. The ROS node corresponding to the occupancy grid mapping involves a ray

tracing algorithm that determines where the obstacles and free space are, according to the data from the laser scanner. This information is published to the appropriate topic, which allows the map to be created as the robot moves around the environment.

The *run()* method handles the movement of the robot, whether it be with the actual robot or in simulation, along with publishing relevant information about the laser scan data and odometry data in the case of manipulating the actual robot. While a finite state machine may have been implemented, this was not done for this particular assignment.

The following link contains all of the media from the testing/experimentation, including relevant videos and images. Each image/video should be relatively straightforward to understand in the context of this assignment.

Link → Media

Evaluation

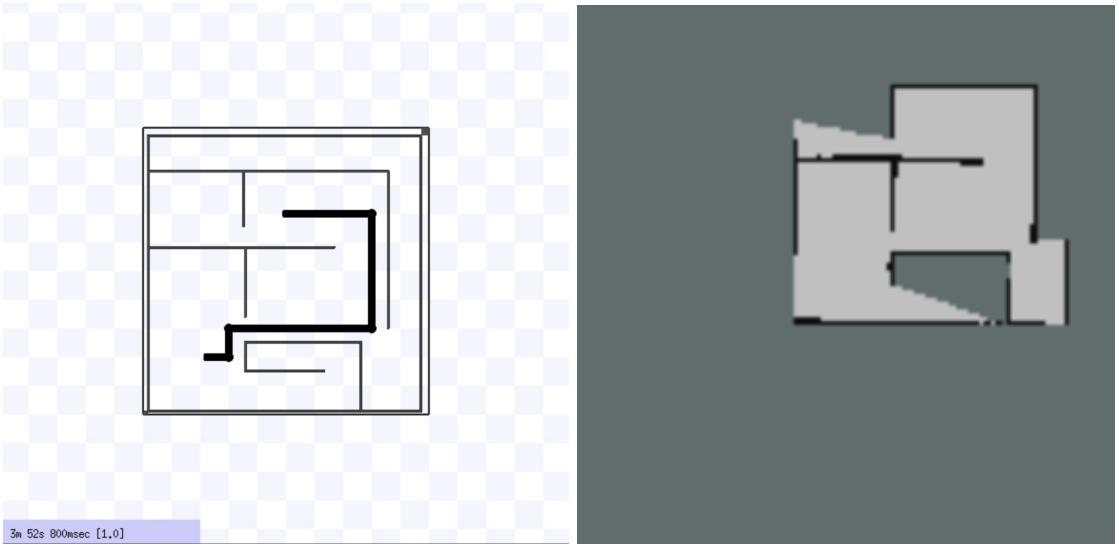
To verify the accuracy/completeness of the program, various experiments were conducted, with given parameters, highlighted as follows.

The primary objective of this assignment was to determine the functionality of the robot, which involved collecting data on simple translational and rotational motion of the robot. This was discussed previously, with the media folder and the data serving as indication of the "accuracy" of the program. In simulation, the robot performs as expected, as there are no elements (of the real world) that impact the functioning of the robot.

The secondary object of this assignment was to map the unknown environment of a robot, via an occupancy grid message. This involved simulation and testing with the actual robot. As indicated previously, a ray tracing algorithm is used to allow the robot to consider its environment and determine where the obstacles and free space are.

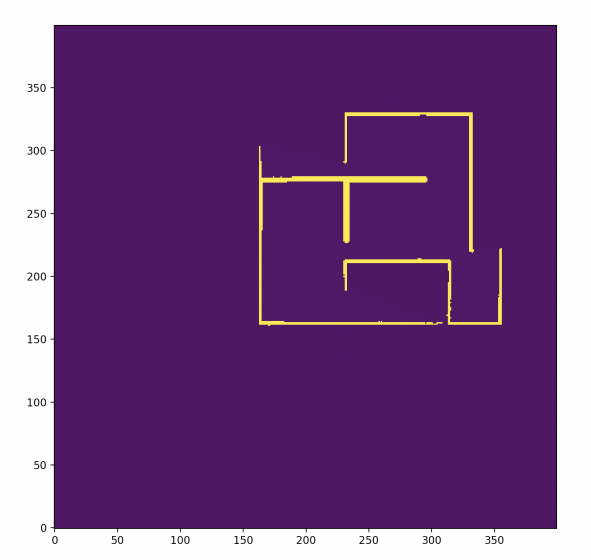
Simulation

The following experiments were conducted, along with the screenshots of the behavior of the robot and the resulting mapping. The main objective of these experiments (in simulation) was to ensure that the ray tracing algorithm functioned correctly. The following display the path of the robot in simulation, along with the occupancy grid (map) published to the appropriate topic and displayed using *rviz*.



In simulation, a reasonable result was produced, as the ray tracing algorithm worked as expected. The map is not necessarily produced at the end of execution of the program, but rather as the robot moves around the environment, which is exactly what was needed.

As indicated previously, to allow for further visualization of the map, *matplotlib* was utilized in conjunction with a separate Python script. The results of that are shown below. In this particular case, however, *matplotlib* is unable to display the unknown space (as with *rviz*), simply displaying the obstacles that are encountered by the robot as it traverses through the maze.

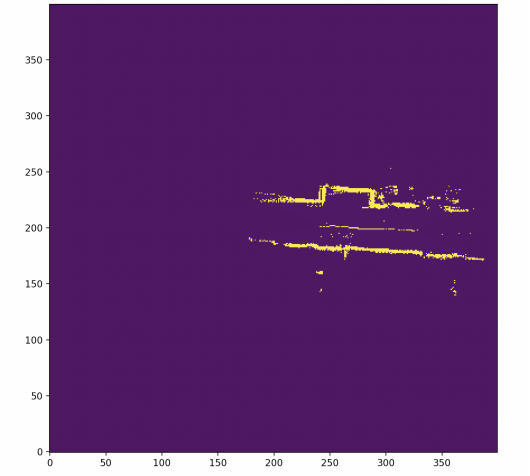


Real Robot

The assignment indicates that following the testing/experimentation in simulation, the algorithm should be implemented to map any part of the ECSC with the real robot. This ended up being significantly more challenging, due to various issues that will be discussed later. The following environments were mapped using the real robot.

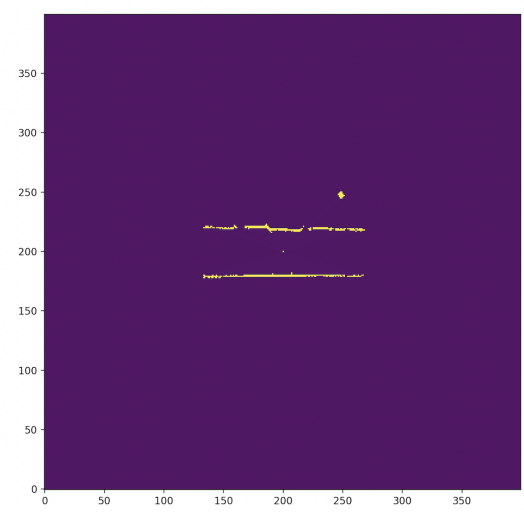
The map size was set to be 400x400, with a resolution of 0.05m. In this way, the robot was able to map a 20m x 20m environment, which was appropriate for this assignment.

Hallway w/ Cutout



The following *matplotlib* graph displays the results of the occupancy grid after passing through a hallway with a cutout to a classroom. This is quite clearly seen, though there appears to be a relative amount of noise in the data collected by the laser scanner.

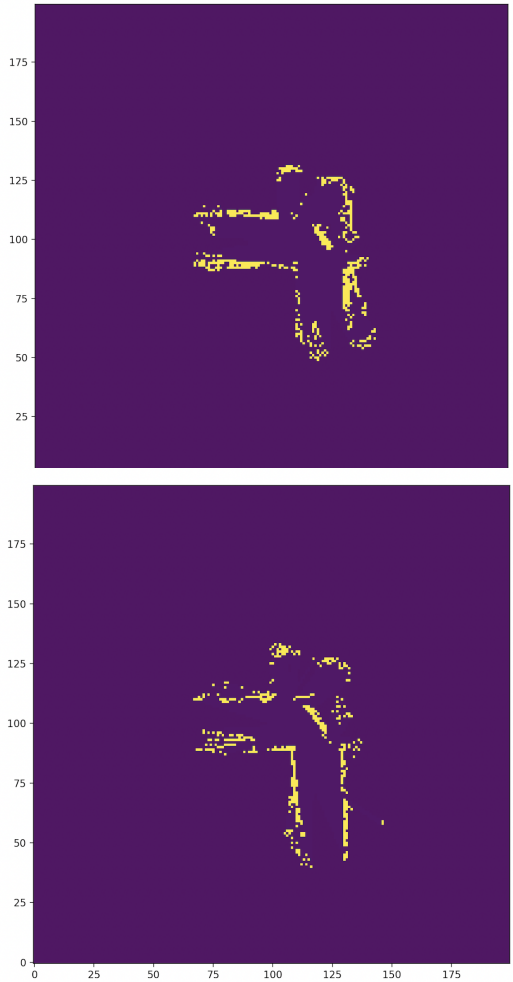
Simple Hallway



The following *matplotlib* graph displays the results of the occupancy grid after the robot was sitting stationary in a hallway of the ECSC.

In this scenario, there appears to be significantly less noise, which is a result of the robot not moving around, with the laser scanner in a stationary position. Regardless, there is still inconsistency with the physical world; the "blob" for lack of a better word located above the hallway displays an obstacle that was viewed through a window.

Corner



The following *matplotlib* graphs display the results of the occupancy grid after the robot moved around a corner of the ECSC, specifically with various open areas around the exterior of the corner. (See Media Folder)

For this particular environment, multiple passes were done to confirm the veracity of the occupancy grid. It seems that even though there may be noise in the data, there is relative consistency. Thus, we could construct an accurate map of the ECSC by performing multiple passes and comparing them with one another.

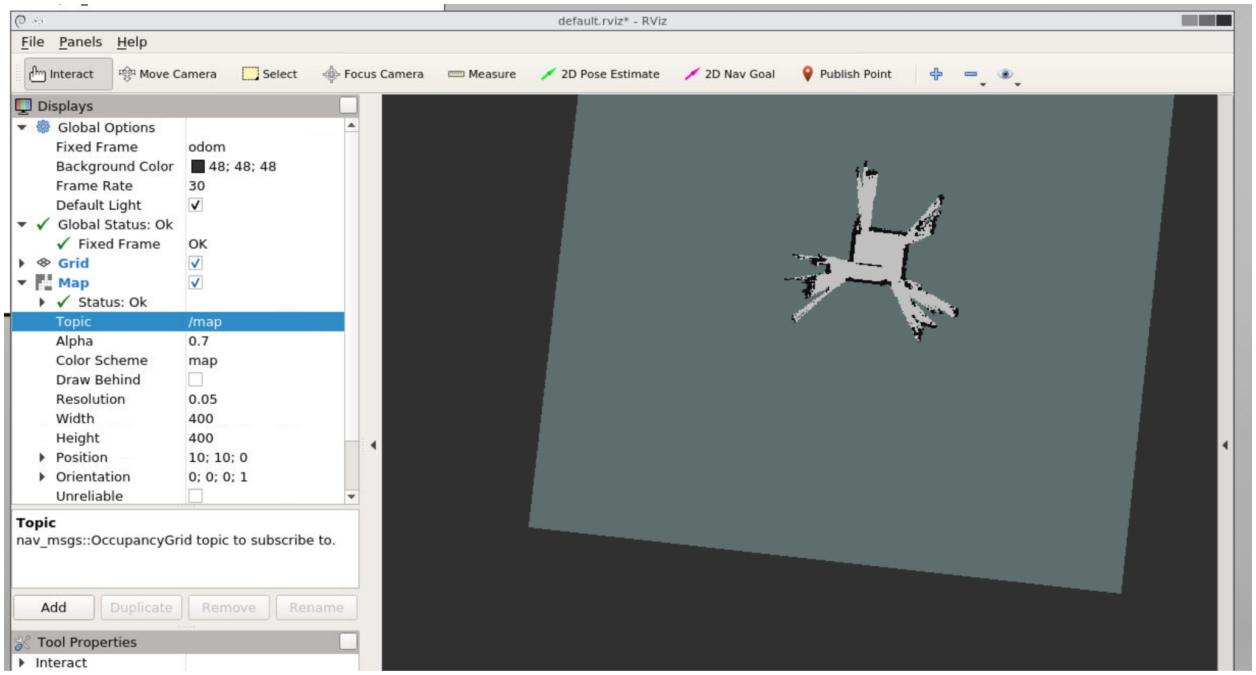
Using rviz

In conducting various tests with the robot, the use of *rviz* was difficult, considering that we were no longer using the simulation. Regardless, with a little bit of patience, the process was successful. The following images are a few of the plots that were generated by the ray tracing algorithm and mapped accordingly, displayed on *rviz*.

Simulated Environment → Robotics Lab

The following *rviz* visualization displays the result of the occupancy grid in the constructed set-up within the robotics lab. This set-up was the same that was used for the *square* simulation, and thus, is viewable in the media folder under the appropriate file.

As demonstrated, the ray tracing algorithm functions as expected, picking up the obstacles, as well as the gaps between obstacles that indicate there is free space at each of the corners of the bounded box.



Hallway

The *rviz* visualization displays the result of the occupancy grid in a hallway environment of the ECSC. Once again, videos/images were captured of the specific environment, and thus, are viewable in the media folder under the appropriate file.

In this given scenario, the robot is able to effectively map the cutout for a classroom, indicating the accuracy of the ray tracing algorithm.

Conclusion

The program functions as expected, according to the specifications.

The robot does not seem to encounter difficulties, in that it does not run into issues with the ray tracing algorithm (at least in simulation). In the actual world (working with the robot), there seem to be issues of synchronization between the odometry and the laser scan, which create noise in the occupancy grid map data. This further explains why quick motion of the robot tends to yield inaccurate results.

To further enhance the mapping algorithm, as indicated previously, it would be effective to perform multiple passes over a given area, and compare the results from each pass. In this way, there would be a more accurate approximation of the real world, given the limited robot sensors that we were using.

Credits

Kevin Cao (May 10, 2023) → High-level discussion regarding the ray tracing algorithm and the various testing methods utilized.

OccupancyGrid Message (Documentation)
http://docs.ros.org/melodic/api/nav_msgs/html/msg/OccupancyGrid.html