ECE5242 Project 3: SLAM

Wheel-odometry based mapping code checkup: 3/22/2022 at 9:25am on Canvas, <NetID>_project3_odom.zip
Wheel-odometry based mapping report checkup: 3/24/2022 at 11:59pm on Canvas, <NetID>_project3_odom.pdf
Complete code due date: 3/31/2022 at 9:25am on Canvas, <NetID>_project3.zip
Complete report due date: Monday, 4/11/2022 at 11:59pm on Canvas, <NetID>_project3.pdf

In this project, you will implement mapping and localization in an indoor environment using information from wheel encoders and range sensors. You will integrate the wheel odometry information from a mobile robot with a 2D laser range scanner (LIDAR) in order to build a 2D occupancy grid map of the walls and obstacles in the environment. Training sets of odometry, inertial, and range measurements from a mobile robot will be provided for this project. The inertial measurements are not required, but can optionally be used to improve your localization from wheel odometry.

Training data and helper code download:
Now available at https://cornell.box.com/v/ECE5242Proj3-train
Test data release: 3/30/2022 2:00pm: https://cornell.box.com/v/ECE5242Proj3-test

These data sets contain ***timestamped*** sensor values, corresponding to the raw sensor readings. Lidar measurements have units of meters and radians. Download these files and be sure you can load and interpret the file formats. 'docs/platform_config.pdf' and `docs/README_Encoders.txt` detail the robot and encoders, and utility functions (load_data.py, test_load_data.py) can help you load and understand the data. docs/'suggestions.txt' gives many further tips on how to proceed.

Wheel-Odometry Based Mapping Code Checkup (due 3/22/2022 at 9:25am, <NetID>_project3_odom.zip)
Do not include training data when you submit.
The in-class presentations on 3/22/2022 will be based on training data.

Wheel-Odometry Based Mapping Write-up Checkup (due 3/24/2022 11:59pm, < NetID >_project3_odom.pdf)
Submit the current draft of your project report including the following sections: Introduction, Problem Formulation, Technical Approach, Results and Discussion. You should re-use (and add to) this draft when submitting your final write-up. **Make sure your result includes a picture of your resulting map of the environment and the robot's motion in that map.**

Complete Code (due 3/31/2022 9:25am, <NetID>_project3.zip)
Do not include training data when you submit.
The in-class presentations on 3/31/2022 will be based on newly-released test data.

Complete Write-up (due Monday, 4/11/2022 11:59pm, < NetID >_project3.pdf)
Submit the final draft of your project report including the following sections: Introduction, Problem Formulation, Technical Approach, Results and Discussion.
Make sure your result includes images from running SLAM on training and test data, including the environment and robot's motion in the map.
If you have video files, please include them! (eg: as a link to a file in google drive)

Project Steps:

1. First, you should trace the path of your robot using pure wheel odometry measurements (optionally, you can improve your path estimate by also using yaw gyro readings). ‘docs/platform_config.pdf’ and `docs/README_Encoders.txt` detail the robot and encoders, and utility functions (load_data.py and test_load_data.py) can help you load and understand the data. ‘docs/suggestions.txt’ gives further tips on how to proceed.

2. Make a 2D map of the environment using range readings. You should then be able to provide a visualization of the motion of the robot within your 2D map. At this point, you will have done the first phase of this project and should be ready for the checkup!

3. Next, you will need to simultaneously localize the robot pose while constructing the surrounding 2D map, using a pose filter and occupancy grid algorithm. Python files are provided in MapUtils (along with cython versions and binding instructions in MapUtilsCython) to help you experiment with 2D LIDAR scan matching.

4. TESTING. You should then make sure that your program can take input sensor readings from unknown environments. We will release the test data at 2pm on 3/30/2022 in order to give some time to run your program with test data. You should be able to show your results with proper visualization from an unknown test dataset.

Tips for tuning SLAM hyperparameters:

Robot width parameter: Use a value that worked well for odometry-only.

Number of particles: Increasing this can only make your algorithm get better. 5-10 is almost certainly not enough. Try searching in the 30-100 range.

Min effective number of particles before resampling: Search in the 30-100 range as well.

Grid discretization: The finer your grid is, the more particles you'll need in order to get fine-grained alignment, and if you go too fine it might make sense to start adding blurring to your lidar "hits" and updating nearby cells as well. Try searching in the range 0.05m-0.1m for the width of each cell in your grid.

X-Y-Theta noise at each timestep: Play with this value. If you make this too small, your particles won't spread out and your results will tie out with odometry-only. If you make this too big then your particles will spread out a lot, so you need a lot of particles to make sure your density coverage is high enough that you get some good particles. You can tweak this parameter by looking at a plot of the positions of your particles over time

and visually inspecting them to make sure that the particles have visible dispersion but stay as reasonable paths (after re-sampling). If your particles go all over the place, either your re-sampling isn't working or you have too much dispersion compared to your number of particles. If your particles all look visually like they're in the sample place exactly, you have too little noise at each timestep.

Log-odds increase for a "hit": Play with this value in the range 0-1

Log-odds decrease for a "miss": Play with this value in the range 0-1

Min/Max cap for map log-odd: Play with this value in the range 1-30

Minimum Lidar Range: You'll notice that the lidar scanner can detect parts of the robot itself. You might want to filter these out by only using lidar hits that are more than some threshold min_lidar away from the sensor (up to you to decide if you want to do this and if so, what value to use for min_lidar to filter out lidar hits closer than your minimal range). Optional.

Tip for testing full SLAM code:

If you aren't seeing much improvement from your SLAM algorithm, and aren't sure if your SLAM is working at all, try the following:

Run odometry-only with a robot width of something you know is not a great compensation for wheel slip (eg: a robot width that is smaller than optimal). Your map should not be pretty.

Now, run SLAM using that same (slightly wrong) robot width. If your SLAM algorithm is working properly, your SLAM algorithm should be able to correct for the bad robot-width estimate and give a great map. The corrected map should be at least as good as what you get with your best robot-width estimate using odometry-only. If your SLAM isn't able to compensate for that slightly-wrong robot-width estimate, then your SLAM algorithm is not working.