COMP 417 – Assignment 2

Mapping and State Estimation

Out: Oct 31st, 2019
Due: Nov 17th, 2019

Getting Started

This assignment requires you to run and modify several "ros nodes" – those are each a piece of software that lives within the Robot Operating System (ROS) world and communicates with existing robot software tools. The first thing you must do is to secure a working installation of ROS "melodic" (that's a version name). There are several approaches for this:

1. (**Recommended**) It is installed on the Trottier lab machines. We've tested the assignment on these machines, and there are only a few small notes to make sure you have a good experience doing the assignment there:
    a. The machines are shared, and if you're not careful, your ROS code can interfere with that of another student. It's best not to pick mimi (the shared server) to do this, but rather one of the lab machines. Check if someone's already running ROS on that machine with "$ ps aux | grep roscore".
    b. Some of our needed software will require live display on somewhat "heavy" 3D viewers. For those components, you probably want to go physically sit at the machine, as we have not found "X forwarding" over ssh to work well.
2. It can be installed directly on your own machine quite easily if it's Ubuntu 18.04. Find instructions on www.ros.org.
3. In case you don't run Ubuntu now, there are several ways to begin running it on your own machine, and then subsequently in obtain ROS:
    a. By installing a Virtual Machine software (such a Virtual Box) and running an Ubuntu 18.04 "guest" operating system within a sub-window. A VM that's particularly helpful for this can be found here: ROS Melodic VM Image. (this option gives a bit worse performance as both your current OS and Ubuntu must run… best if you have a modern and powerful laptop/desktop)
    b. By "dual-booting" Ubuntu as a native OS on your computer. Look up the instructions on Ubuntu's install page for this, but please be careful as you can lose data on your existing system!

Once you have a working version of ROS melodic, create a "ROS workspace" to get rolling:

$ source /opt/ros/melodic/setup.bash
$ mkdir -p <wherever_you_want>/comp417_ws/src
$ cd <wherever_you_want>/comp417_ws/src
$ git clone https://github.com/dmeger/COMP417_Fall2019.git
$ cd ..
$ catkin_make -j1

Now edit your **~/.bashrc** file so that the last line is:
source <wherever_you_want>/ comp417_ws/devel/setup.bash. This activates our A2 code in new terminals. Also type **$ source ~/.bashrc**, for the current shell.

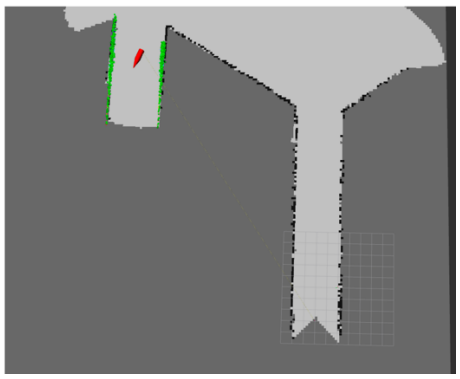
Question 1: Occupancy Grid Mapping (5 points)

In this exercise you are going to implement parts of the occupancy grid mapping system discussed in class. In particular, you are going to map an environment based on known odometry estimates and known 2D laser scans. The functionality that you need to implement is marked using to-do comments in the file:
estimation_assignment/python/occupancy_grid_mapper.py

To run your code, cd **path/to/comp417/estimation_assignment/** and execute the following commands on three different terminals:
$ roscore & rosrun rviz rviz
When rviz initializes, go to File > OpenConfig and then load the configuration file in **estimation_assignment/resources/comp417.rviz** which is going to start the visualization of laser scan messages, frames of reference, and debugging visualizations. Save this configuration file as the default in your **/home/username/.rviz/default.rviz**, so you won't have to do this every time you restart rviz.

$ roslaunch estimation_assignment occupancy_grid_mapper.launch odometry_orientation_noise_std_dev:=2.5 odometry_position_noise_std_dev:=0.1
In the roslaunch command given above, the standard deviation of the odometry orientation noise is given in degrees. In this example the true yaw of the robot is corrupted by random noise from $N(0,2.5^2)$. Similarly, the true position is corrupted by random noise from $N(0,0.1^2)$, which is expressed in meters.

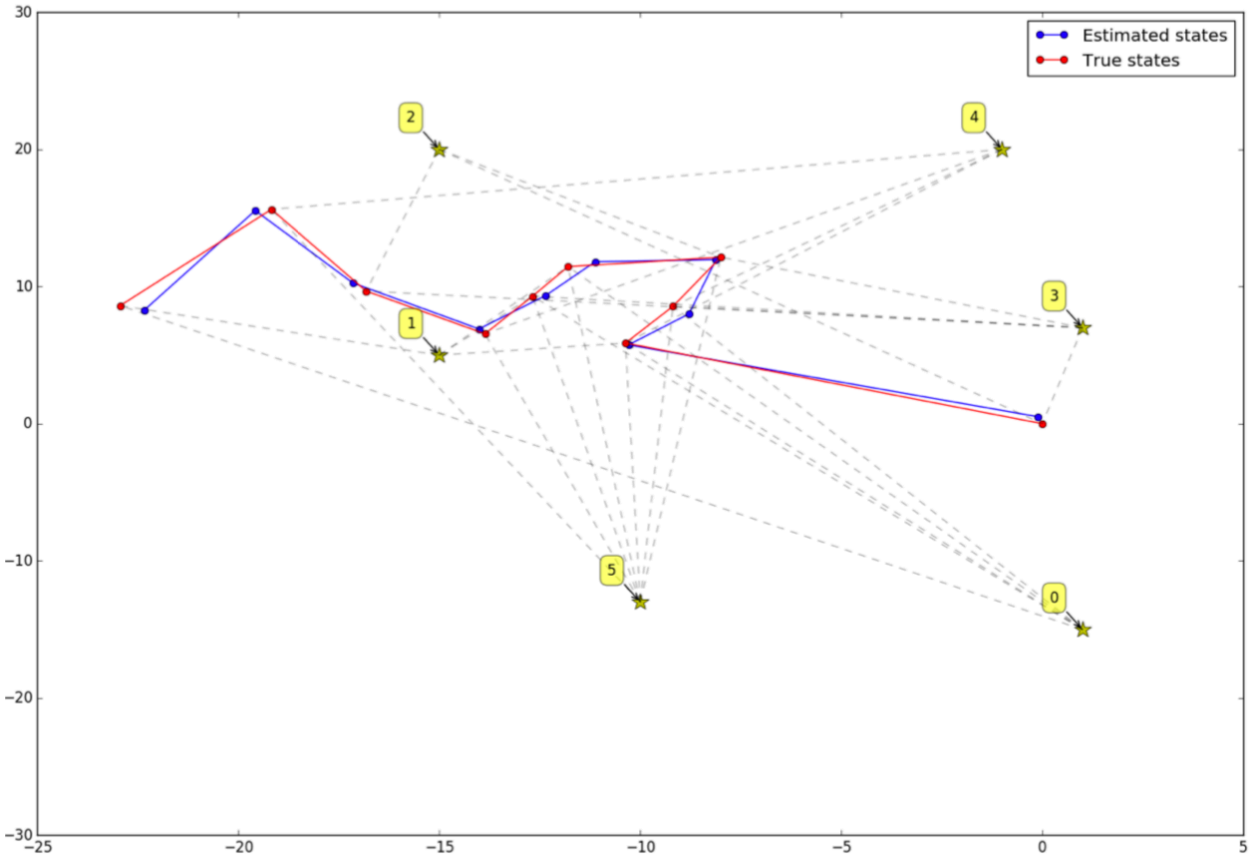
$ rosbag play laser_and_odometry.bag
laser_and_odometry.bag is a recording of laser scans and odometry data in the environment you used in Assignment 1. The expected result for perfect odometry looks similar to Figure 1.
What you need to submit: 5 images of maps produced by your mapper, with noise parameters (0, 0), (2.5, 0), (5, 0), (2.5, 0.1), (5, 0.1) respectively. Your images should be named map[0|1|2|3|4]_firstname_lastname.png. Also, submit a video recording of the rviz visualization for the odometry noise combination (0,0), demonstrating your map being built from beginning to end. Your video should be named **map_0_firstname_lastname.mp4/avi/ogg**.



2 Least squares localization (5 pts)

In this exercise you are going to solve the localization problem in a map of known landmarks. This one will be pure Python and not involve ROS. Assume your robot has 2D state $x_t = [p_x(t) \; p_y(t)]^T$ and an omnidirectional motion model $x_{t+1} = x_t + u_t \delta t + w_t$, where $w_t \sim N(0, \sigma_w^2 I)$ The robot is moving through an environment that has L static landmarks $l_1,\ldots,l_L$ whose positions are known. Occasionally the robot makes the following set of measurements: $z^{(i)} = ||x - l_i|| + n$ where $n \sim N(0, \sigma^2)$, for some of the available landmarks. Your goal is to estimate the sequence of states $x_{1:T}$ by minimizing a least squares cost function.

You can find starter code in **estimation_assignment/python/localization. py**. It provides the set of measurements made at each timesteps. Your task is to implement the cost function using numpy and the nonlinear least squares solver in scipy.optimize.least_squares. There are to-do comments in the code that explain in detail what you need to implement. After you are done with your implementation, the expected outcome is the following figure:



What you need to submit: your code in the file **estimation_assignment/ python/localization.py**.

P.S.: You can think of cell phone localization based on static cell towers as a particular application of this problem.

3 Monte Carlo Localization (6.25 points)

In this exercise you are going to implement the particle filter for locating a robot within an indoor map (also called Monte Carlo Localization), as discussed in class. Your robot is going to start by being completely lost in the environment, so the particles are going to be spread out uniformly at random in the world. Then, after many measurements, its position is going to be constrained and the particle are going to converge to a tight cluster (hopefully around the robot's true position!) The main mechanism for survivial of the fittest among particle will be: which particles are more likely to have generated the laser scans that the robot is observing now?

3.1 Starter Code

In order to keep this question cleanly separated from the previous ROS node, we have created a new folder for it called "**particle_filter_question**". You need to work on the file within that folder, **python/monte_carlo_localization_v2.py**. To run your code, change into the particle_filter_question folder and start 3 terminals. Run in order:

$ **roscore & rosrun rviz rviz**
(As before, change the config to be customized for this question. This time it's located at the particle_filter_question/resources folder).
$ **roslaunch particle_filter_question monte_carlo_localization_v2.launch**
$ **rosbag play -r 0.5 laser_controls_and_odometry.bag**

Note that the -r argument to rosbag reduces the playback rate by half, so that measurements are sent more slowly to your program.

As usual read through the whole code carefully to understand how it interacts with the data and creates its particle estimates. Then, find the #TODO comments and add code in each place. Test by stopping your node (ctrl-c on the roslaunch command), and re-running both your code and the bag file, for each change you make. You should not have to restart rviz each time (we hope!)

For this question, submit your code, and a video recording of the rviz visualization demonstrating your particles converging from beginning to end. Your video should be named mcl_firstname_lastname.mp4/avi/ogg

Grading:

- 2 points for the particle filter resampling
- 2 points for the laserr scan simulation for a given particle
- 2 points for particle prediction error and weight updates
- 0.25 points for the video

4 How to submit

Submit all your work in a file called estimation_assignment.zip that contains your extensions to the provided starter code, as well as all image and videos. Submissions will be done on MyCourses.