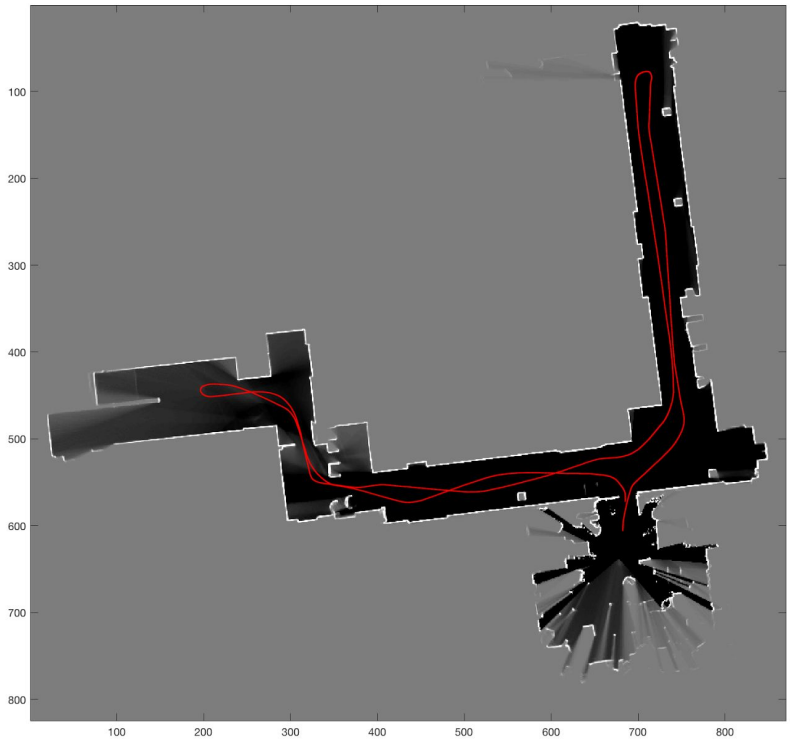
Programming Assignment Week 4

Particle Filter - Pose Tracking

1. Introduction

In this assignment you will be implementing a particle filter for pose tracking in 2D space. Imagine a robot wants to understand its position in order to determine where to next explore. The figure below depicts an example of robot localization from LIDAR measurements projected onto a map.



2. Instructions

Design your Particle Filter Model

1. **Dynamics Model** (= System Model)
    a. In order to move the particles in time, a distribution of odometry updates models the movement of the robot. The particle filter samples from this distribution many times to update each particle's position and orientation.
    b. Encoder measurements are not provided in the interest of simplicity. Instead, you can use a distribution with zero mean, but higher covariance.
        i. [Advanced] Alternatively, you can track the changes in position in time using a Kalman filter to modify the mean away from zero.
    c. Now consider the random noise added to particles, via the system noise covariance Σ_m . The simplest form is a diagonal matrix, with the four variances: Σ_m = *diag*([σ_x^2 , σ_y^2 , σ_θ^2])
        i. In general, you don't have to restrict the form, but a diagonal matrix is a good starting point if you do not have a strong reason not to do so.
        ii. You should sample at random from this multivariate normal distribution to add noise. NOTE: With a diagonal matrix, you can sample components individually, using randn.
        iii. [Advanced] Alternatively, you can use the estimated velocity of the robot, from the kalman filter, to expand or contract the noise as the velocity increases and decreases.
2. **Map Registration**
    a. The measurement data is provided as a set of LIDAR distance readings that occur at certain angles, in the local frame of the robot.
        i. This data is plotted in example_test.m
        ii. At this point, you should test methods to transform the *local* LIDAR data into the *global* map coordinates.

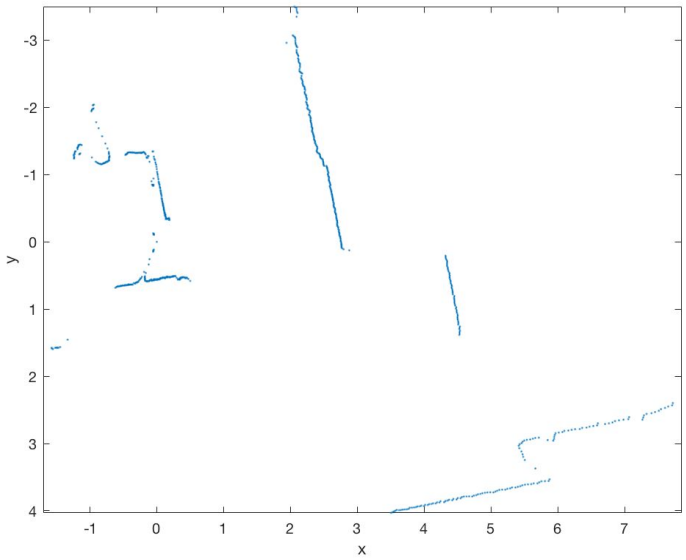


Fig 2. Local frame plot of the LIDAR data - in meters. You should convert to discrete map cell coordinates, discussed in (b.)

b. Scan matching against the map can be performed, where the global map coordinates are discretized based on the map scaling. For instance, x=1.5m, y=2.2m may be coordinate x=15, y=22 if there are 10 cells per meter.
    i. Consider using a subset of the lidar scan points in order to save on computation time.
    ii. [Advanced] The free space coordinates can be considered at this time, as well. Use the provided bresenham function. This will provide a set of cells for each ray, but can become computationally expensive. See example_bresenham.m from Week 3 for detailed use.
c. Determine your correlation scoring function. How will the correlation score increase when LIDAR free space measurement aligns with map free space? Decrease when the LIDAR measurement is on map free space? You should tune these values. The table below shows an example of of possible correlation updates, where columns show the state of the cell in the map and rows show the state observation of the cell from the LIDAR readings.

	Map Free	Map Occupied
LIDAR Free	+1	- 5
LIDAR Occupied	- 5	+10

3. Particle Re-weighting

a. At this point, with an ability to update a particle and measure its correlation to the map, you can make a set of particles to track the state of the robot. A particle will be an array of a position and orientation hypothesis (x, y, theta), coupled with a weight.
b. Start with 10 particles and tune to see the best number for this particular localization assignment, maybe 100 particles is best. Be careful - more particles means more computation time!
c. For each LIDAR measurement, you will compute the correlation score of every particle. At the start, each particle can be equally weighted, so for 10 particles, each particles will have a weight of 0.1. The new weight will be a product of the current weight and the correlation score. Be careful about the scaling of the weights - the numbers may get quite small! Renormalize all weight after each cycle so that the sum of the weights equals 1.
d. Calculate the effective number of particles, as discussed in Lecture 4.3. Resample particles if this number becomes too low. After resampling, you will have duplicates (in weight and position/orientation) of highly weighted particles, but then noise will be added based on the motion model, so no duplicates will remain, and the next map correlation step will re-weight these particles.
e. Resample the particles

Implementation

1. You will complete a function that takes the set of parameters, sensor data and given map as input, and returns a the positions and orientations of the robot. The signature of the function is given as:

```
function [ myPose ] = particleLocalization(ranges, scanAngles, map, param)
```

2. The param variable will include the map resolution (param.resol) as the number of cells per meter, map size (param.size) as the number of cells in the map, and origin of the robot (param.origin) as the starting cell coordinates of the robot in the map.
3. Your function should return the entire path of the robot, in a 3 by n matrix, where n is the number of LIDAR observations made. You will keep track of the x, y position components and the theta angular component, in that order, for each n observations
4. example_test.m is provided to help visualize your result.

Evaluation and submission

To submit your result to our server, you need to run the script runeval in your MATLAB command window. Please specify the path where the encrypted test data are located. A script will then evaluate your particleLocalization function and generate an output file, **SubmissionLocalization.mat**, to be uploaded to the Coursera web UI. You may submit your result multiple times, and we will count only the highest score towards your grade. The score reflects how close your predicted ball positions are to the observed ball positions.