ECE 276A Project 2
Particle Filter SLAM

Shiladitya Biswas
Dept. of Electrical and Computer Engineering
Universrity of California,San Diego)
California, USA

I. Introduction

Physical world is unpredictable and full of uncertainties. A flat floor might not be flat, a straight wall may not be straight, floor/ground friction will always vary from place to place. In the world of robotics we try to perceive and manipulate this physical world through computer generated signals which we can very well control at our own will. Similarly there will always be uncertainties, manufacturing fault etc in a robot hardware. These faults include faulty sensor, worn out rubber wheels (less friction), etc. Due to these uncertainties the robot's real state in the physical world is a function of both the computer signal(Known signal) and the environment state(completely unknown). For example, the computer sends instruction to the motor driver to rotate 5 time to travel 5m in the forward direction, but dues to worn out wheels and slightly steep floor the wheel couldn't generate the required traction and the robot only travelled 4 meters. Thus the robot controller and the real world goes out of sync i.e. the controller thinks the robot has travelled 5m forward but in reality it hasn't. This may lead to several fatal accidents (Self-Driven cars, rehabilitation robotics etc) and hence robots won't be safe to use. Thus it is evident that for a robot to manipulate/work safely in an environment, we need to take care of the unknown uncertainties and accordingly incorporate environment state estimation capabilities in the robot. Simultaneous Localization and Mapping (SLAM) is one such capability which allows a robot to simultaneously generate a Map of the unknown environment it is in and localize itself within the generated MAP. In this project we are given a set of sensor readings and Robot pose, using which we have to build the map of the environment it is. The project consists of 3 steps: Mapping, Prediction and Update. Mapping is achieved using the LIDAR data given. Prediction and update is performed by the Particle filter.

II. Problem Formulation

As discussed in the previous section, we are trying to solve a mapping and localization problem i.e. SLAM. The central goal of SLAM can be stated as follows: **Given a series of control signals and sensor measurements, estimate the unknown environment/Map and its pose relative to this map.**

I would like to thank Saurabh Mirani and Parikshit Jain for the helpful discussions on the project.

Mathematically speaking, given a dataset of the robot inputs $u_{0:T1}$ and observations $z_{0:T}$ we have to maximize the data likelihood conditioned on parameters Map m and state $x_{0:T}$:

$$\max x_{0:T}, m \log \text{ p}(z_{0:T}, u_{0:T1} | x_{0:T} \text{ , m}) \tag{1}$$

Where $x_{0:T}$ is pose trajectory of the robot from t=0 to T and $z_{0:T}$ is the set of all observations from t=0 to T. and m is the map.

Other than this we will also implement a Bayesia filter that keeps track of the the probabilities given below:

$$p_{t|t}(x_t) = p(x_t | z_{0:t}, u_{0,t-1}) \tag{2}$$

$$p_{t+1|t}(x_{t+1}) = p(x_{t+1} | z_{0:t}, u_{0,t}) \tag{3}$$

where $p_{t|t}$ represents the pdf of robot state at time t conditioned on all the observations upto time t and control input upto time t-1. Similarly, $p_{t+1|t}$ represents the pdf of robot state at time t+1 given all the observations and control inputs upto time t. In this project we implement a particle filter which is a type of Bayes filter to solve the localization problem.

From class we learnt that SLAM is a chicken and egg problem comprising of :

1) **Mapping:** Given lidar pose, scanning data head and neck angle, transform the laser scan lengths from the LIDAR Frame to the world frame using the robot configurations given to us. We then map the occupancy of the environment by considering the grid cells that the laser hit and the grid cells upon which the laser has travelled unblocked. We consider a discreet map m of the world which maintains the probability mass function of the the environment.
2) **Localization:** Given an occupancy map M of the environment, localize the robot and estimate its trajectory $X_{0:T}$. We first initialize the map obtained above with N particles, where each particles represents a probable pose (x,y,θ) of the robot in the map. Then we transform the laser scan lengths from the LIDAR Frame to the world frame for all the particles.We then obtain a map correlation value for each of the particles conditioned on the map generated in mapping step. The particle with the highest map correlation becomes the most probable location of the robot. We then save the coordinates of these particle to estimate the most probable trajectory of

T_ltoR

$$\begin{bmatrix} \cos(\theta_1)\cos(\theta_2) & -\sin(\theta_2) & \sin(\theta_1)\cos(\theta_2) & 0 \\ \sin(\theta_2)\cos(\theta_1) & \cos(\theta_2) & \sin(\theta_1)\sin(\theta_2) & 0 \\ -\sin(\theta_1) & 0 & \cos(\theta_1) & 0.48 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T_rtoW

$$\begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & x \\ \sin(\theta_3) & \cos(\theta_3) & 0 & y \\ 0 & 0 & 1 & 0.93 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T_full

$$\begin{bmatrix} \cos(\theta_1)\cos(\theta_2+\theta_3) & -\sin(\theta_2+\theta_3) & \sin(\theta_1)\cos(\theta_2+\theta_3) & x \\ \sin(\theta_2+\theta_3)\cos(\theta_1) & \cos(\theta_2+\theta_3) & \sin(\theta_1)\sin(\theta_2+\theta_3) & y \\ -\sin(\theta_1) & 0 & \cos(\theta_1) & 1.41 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fig. 1. Tranformation Matrices used

the robot. This part of the program is done by the particle filter .

Resampling: It is generally observed that in the localization step the particles with low weights becomes so small that a it becomes impossible to track their weights further. This phenomenon is generally called particle depletion. inorder to avoid this particle resampling need to be done. There are several methods of particle re-sampling. In this project I have used importance resampling for this purpose.

III. Technical Approach

A. Mapping

For each timestamp I extracted the laser data and the corresponding joint angles of the robot. The invalid scan data points i.e. lengths greater than 30m and less than 0.1m were removed. The valid data thus obtained were transformed from the lidar frame to world frame w.r.t the pose of the particle having the highest weight to obtain the best world coordinates or best estimate of the Map. The transformation matrices shown in figure 1 . Here θ_1 = Head angle and θ_2 =Neck angle. T). T_ltoR= Transformation matrix from laser to robot frame, T_rtoW= Transformation matrix from robot to world frame and Transformation from lidar to world frame T_full= $T\_rtoW \times T\_ltoR$. $x = scan\_length \times cos(scan\_angle)$ and $y = scan\_length \times sin(scan\_angle)$

Now, the best world coordinates obtained so far is in meters, hence this has to be converted to grid cell units. I used the numpy.ceil() function and other map parameters like length, breadth and map resolution to convert the length in meters to grid coordinates. A similar operation was done on the (x,y) position for the particles as well (since delta_pose data is in meters and radians). The best world grid coordinates obtained above and the highest weight particle pose (position in grid coordinates) were then fed to the skimage.line function to get the intermediate cell (i.e. the cells above which the laser has passed unblocked). The log odds of theses intermediate cells were then reduced by some constant amount 'K'. On the other hand the log odds of the cells having the best world grid coordinates were increased by the same constant 'K'. This updated log odd map was used to construct the occupancy grid map and the free space map.

B. Particle Filter

As already mentioned before, the particle filter is a special type of Bayesian filter. The particle filter uses a mixture of delta function with weight α_i to represent equations 2 and 3. The modified Bayesian filter equations 2 and 3 for the particle filter are as follows.

$$p_{t|t}(x_t) = \Sigma_{k=1}^{N_{t|t}} \alpha_{t|t} \delta(x_t; \mu_{t|t}^{(k)}) \tag{4}$$

$$p_{t+1|t}(x_{t+1}) = \Sigma_{k=1}^{N_{t+1|t}} \alpha_{t+1|t} \delta(x_{t+1}; \mu_{t+1|t}^{(k)}) \tag{5}$$

In our case the each delta function is a particle representing an arbitrary guess or hypothesis of the robot pose in the environment and α_i corresponds to the weights of each particles.

1) Prediction Step: We first initialize N particles with equal weights $\mu_i^t = (0,0,0)^T$. Each particle has weight $W_i = 1/N$. A zero mean Gaussian noise and the previous odometer reading ('delta_pose') were added to all the particles.

$$\mu_i^{t+1} = \mu_i^t + \mathcal{N}(0, \sigma^2) + delta\_pose^2 \tag{6}$$

2) Weight Update Step: With the predicted particle pose of each particles the laser data and joint angles at a give timestamp to get the world frame grid data points for each particle pose. These data points obtained and the updated occupancy map (obtained in the mapping section) Are then fed to the mapCorrlation function to get a quantitative weight of each particles. The mapcorrelation function gives the sum of the values of all the cells which the laser hits. Hence depending on the particle pose the sum will vary. The correlation values thus obtained for all particles and he previous particles weights were then normalized using the softmax function give below.

$$W\_new_i = \frac{W\_old_i . e^{corr_i}}{\sum_{i=1}^{N} W\_old_i . e^{corr_i}} \tag{7}$$

Now, μ_i^{t+1} is the new predicted pose of the robot. The updated weight of each of the particles represents the probability that the robot is in that particular pose depicted by the particle. Naturally, the robot is most likely to have a pose of the particle having the highest weight.

3) Resampling: Particle filters are always prone to Particle depletion. Particle depletion is a phenomenon in which most of the updates particle weights are very close to zero. This happens because the finite set of particle we initialized earlier are not accurate prediction/hypothesis of the robot state. Hence with time the weights of the particle having the most likely

robot pose increases and the weights of the particle having the least likely robot pose decreases and ultimately reaches zero. Hence resampling of the particles is needed to be done. Resampling is applied to the particles at time t if the effective number of particles:

$$N_{eff} = \frac{1}{\sum_{i=1}^{N}(W_i)^2} \tag{8}$$

For this project I used Sample importance resampling. The algorithm for this resampling strategy for a target distribution $p(.)$ given a proposal probability density function (PDF) $(.)$ is shown below.

1) Draw $W_1, W_2, W_2.....W_N$ from $q(.)$.
2) Compute importance weights W_i=$\frac{p(\mu_i)}{q(\mu_i)}$ and normalize over all W_i
3) Draw μ_i independently with replacement from $\mu_1, \mu_2, \mu_3.....\mu_N$ with probability W_i and add to the final sample set with weight $\frac{1}{N}$

In this project I chose N=100 and N_eff=35

C. Ground plane removal

In all the dataset it was observed that the robot looks at the floor through its camera. Since the camera is situated above the lidar, it is evident that there will be several laser readings in the scan data where the laser has reflected back from the floor. We need to eliminate these points, as they aren't really obstacles. In order to do so we again look at the lidar to world frame transformed points. We simply discard those points which have a z value less than 0.2 meters. And use the rest of the points as it is. This way the noise in map generation was due to stray laser rays from ground plane was successfully eliminated.

D. Timestamp Synchronization

This is a vital step for any SLAM problem. From the laser and joint angle data it was discovered that their timestamps didn't match. This is because the data on a the real robot arrives asynchronously and has time stamps. So I had to synchronize them by simply look up the closest joint timestamp in the past upon receiving a lidar scan. I ran a separate for loop to achieve this task, for a given timestamp in laser data I found the nearest timestamp in the Joint angle data by using numpy function argmin.

IV. Complete Working

We begin by setting parameters declaring 2 empty Maps namely Occupancy MAP and LogOdds MAP and their dimension, resolution. We then initialize N number of particles at the origin each having zero yaw angle. All the particles are initialized with uniform weight i.e. (1/N). The steps below go in a loop, hence they are enumerated.

1) We add odometry value and a zero mean Gaussian noise to the pose of theses particles. This operation randomly distributes the particle around the origin.
2) Then we find the map correlation of each of the particles with respect to the Occupancy MAP and store them in an array. (Initially the correlation values will be all zero because the Occupancy MAP is empty, as the loop goes on iterating the correlation values become nonzero)
3) Then we update the weights of all the particles using eq 7 and the correlation array obtained in step 2
4) The particle having the highest weight was then identified . The pose of this particle becomes our most probable estimate of the robot pose at that time stamp 't'.
5) We then take the laser scan corresponding to the same time stamp 't' and transform it into world coordinate points with respect to the particle having the highest weight.
6) The world coordinates W_best obtained in the above step and the best particle pose μ_best are then converted to grid coordinates using numpy.ceil function.
7) Next, we increase the values of the LogOdds MAP at the locations W_best location obtained in above step and use skimge.line function to decrease the values of the cells above which the laser has passed, (basically the cells between the world coordinate points w_best and μ_best. Both these increase and decrease are equal in magnitude. Hence the LogOdds MAP was updated successfully. The LogOdd map is then used to update the Occupancy MAP. The Occupancy MAP is set to '1' at locations where the LogOdd map values are greater than zero. The Occupancy MAP is set to '0' at locations where the LogOdd map values are less than or equal zero. This Occupancy MAP is then fed back to the mapCorrelation function in step (2) in the next iteration of the loop.
8) Next we check the N_eff particles of the system, if it is less than the set threshold then resampling of weights is done as discussed in section III-B3

The above steps are repeated for all the laser scans in the given data set. Finally we observe that the free spaces in the map have very less value and the occupied space (i.e. wall boundaries) have very high values. Which can then be plotted to see the map. Also as long as a

V. Result

The results of the particle SLAM implementation on all the datasets is shown below with red line representing the trajectory of the robot. All the maps are very clear. Although I feel the the clearest output was obtained in dataset 1 and 2.

In dataset 0, although the map formation is pretty accurate but the end point of the trajectory is a bit too deviated from the starting point. This slightly contradicts the RGB image of the dataset0, where the robot returns to the same door from where it started. I tried to increase noise in the system but this is the best map I got so far.

For dataset 1,2, since I don't know the real map, I can't really comment on the map generated. But looking at the trajectory of the robot in both the images I feel it is pretty accurate.

Similar to dataset 0, the algorithm performs well on dataset 3 as well, this was cross verified from the RGB data of Dataset 3.

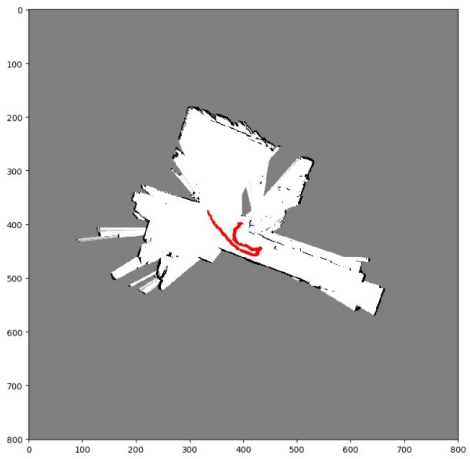

Fig. 2. Dataset0 Free Space Map

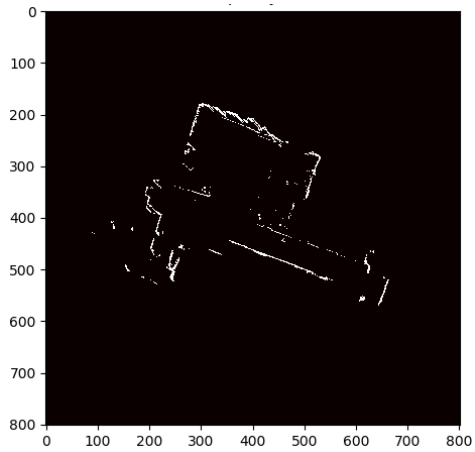

Fig. 3. Occupancy Grid

According to the RGB data of dataset 4 the robot comes back to starting position at the end. Hence I was expecting a loop closure. Clearly from the generated map we see that the trajectory loop isn't closed. So Loop closure didn't work.

Additionally, it was observed that SLAM is very sensitive to sudden change in orientation. Also, keeping too high noise variance for the yaw angle gave bad final output, the whole map got rotated. This is because the yaw angle in 'delta_pose' is in radians. Furthermore, higher the grid resolution higher is the time taken by the program to execute completely.

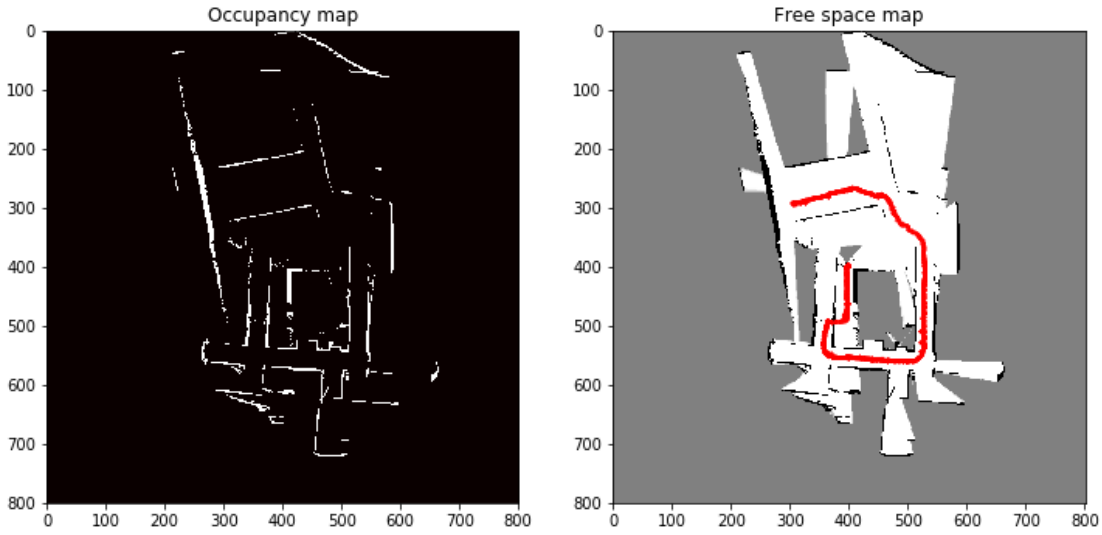

Fig. 4. Dataset1

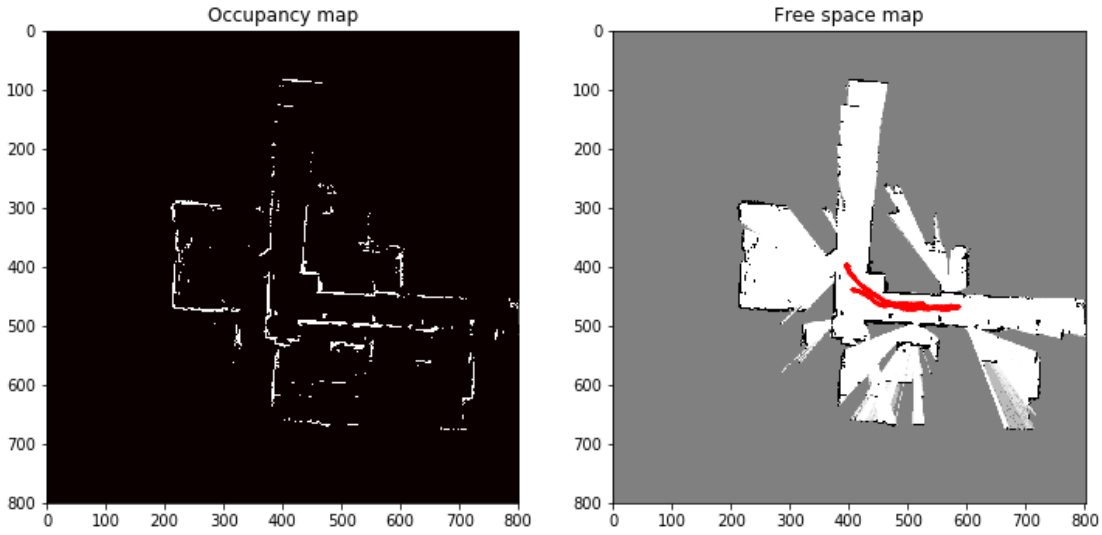

Fig. 5. Dataset2

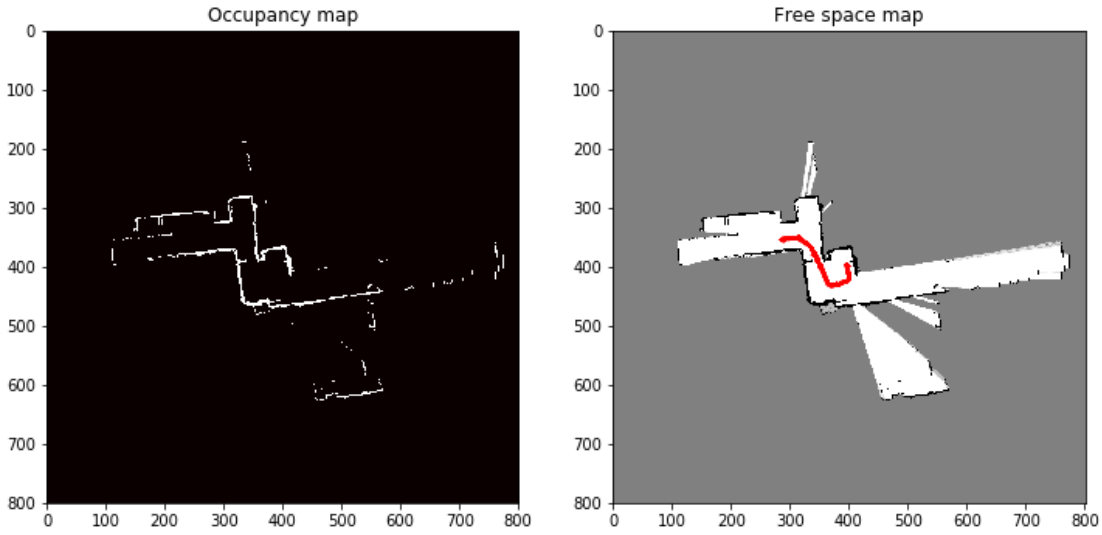

Fig. 6. Dataset3

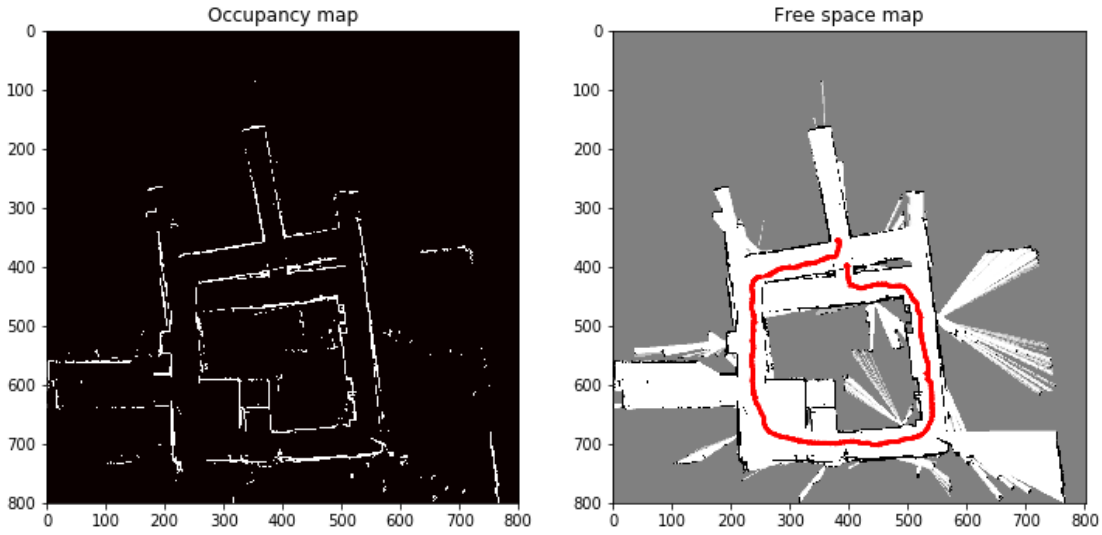

Fig. 7. Dataset4