Simultaneous Localization and Mapping using Particle Filters

Sai Krishnan Chandrasekar
Graduate Student, Robotics
University of Pennsylvania
Email: saikrish@seas.upenn.edu

***Abstract**: Simultaneous Localization and Mapping (SLAM) is a concept that is used to localize a robot and map an unknown environment of the robot at the same time. In other words, the robot both maps the surroundings (maps) and estimates where it is in that map (localization). In this project, SLAM was implemented using Particle Filters. Particle filter is a sequential Monte Carlo methodology. In this project, using on board sensor measurements from the robot, a 2D map of the surroundings of the robot is generated using particle filters based SLAM.*

1 Introduction

The main goal of this project was to estimate the map of the surroundings using Particle Filter based SLAM. The robot has to simultaneously map the unknown environment and localize itself in that environment.

Particle filters make use of "particles" that each have a weight and a pseudo random state initialization. The state was (x, y, θ). (x, y) is the location of the robot in the 2D world and θ is the orientation of the robot. As the robot moves and measures, the motion and sense updates are applied to the particles. In other words, the particles move and simulate the motion of the robot, except they're scattered over the map. As the robot moves, the likelihood of particles far away from the robot reduces, since their measurement readings do not correlate with that of the robot. Using this Sequential methodology, the robot is localized. The Map is built using the measurements of the particle with the most "weight" or likelihood.

The data provided was: Lidar measurements, IMU readings, Encoders and Kinect data. The Kinect information was not used in this project. The Particle filter was implemented using Lidar measurements, Encoders and IMU readings. This was coded up on Python and OpenCV libary was used for image processing.

2 Problem Formulation

The problem at hand was divided into 5 parts:

1. Preprocessing of data
2. Dead Reckoning
3. Localization Prediction
4. Localization Update
5. Mapping

3 Pre-Processing of data

3.1 Lidar:

The Lidar data consisted of absolute odometry, scans and IMU Roll Pitch Yaw. This data was for each timestep. The scans consisted of 1081 scans. The Lidar used was a Hukoyo Lidar. It had a range of -135° to $+135^\circ$. The resolution was roughly 0.252°, which resulted in 1081 laser scans. Each laser scan returned the distance the laser travelled before it hit an obstacle.

This distance was with respect to the Lidar's coordinate frame. The Lidar was placed 15 cm above the head and the head was placed 33 cm above the center of mass of the body. The center of mass was 93 cms above the ground. The head and neck could rotate as well. Their rotations were given by the encoders. Using these rotations and translations, each Lidar scan had to be transformed to the world frame. In other words, each distance vector had to be represented in the world frame. This would ensure that all the distance vectors were consistent and in the world frame, which is the only way to actually estimate how far the robot has travelled using Lidar measurements.

The scans (lidar data) were converted to the world frame using the formula below. Let z_t be the laser scan in the lidar frame.

$$z_t^w = {}_wT_b * {}_bT_h * z_t$$

Here:
z_t^w = Scan z_t in world frame
${}_wT_b$ = Homogeneous transformation matrix from body frame to world frame
${}_bT_h$ = Homogeneous transformation matrix from lidar (head) frame to body frame

3.2 Odometry:

The actual odometry of the state (x, y, θ) was provided. This didn't require too much preprocessing and was used directly for localization.

3.3 Kinect:

The Kinect data wasn't used in this project.

3.4 Time Synchronization:

The timestamps on the data from Joints and Lidar were all out of sync. The timestamps had to be synchronized. This was crucial for the Lidar measurements and the joint angles as the lidar scans had to undergo the corresponding rotation in order to be represented in the world frame. This was done by finding the nearest timestamp in the joint data for the corresponding lidar timestep. The corresponding rotation matrix was applied on the scan to rotate it to world coordinates.

4 Dead Reckoning:

Once the scans were converted to the world frames, dead reckoning was implemented. Here, the lidar data was used as is, without any particle filter implementation to build the map. Initially, a probability map with all regions marked as unexplored is generated.

Given the scan data (lidar data) in world frame, the invalid scans (scans beyond the map) were removed. The laser rays might have hit the floor but they must be considered free space. Hence, thresholding in the z axis was done to avoid this. Using the resolution of the map cells, the scan distances were converted into cells. Thus, the locations of the obstacles for each timestep in terms of cells are identified.

By calculating the convex hull of the scans, the cells on the boundary, which are the actual obstacles were identified. A value of $\log(9)$is added to these cells. All cells within the convex hull are free cells. A value of $\log(\frac{1}{9})$ is added to these cells. This is the "log-odds" maps. These values are cumulative and hence cells that are repeatedly classified as obstacles have higher log odds of being obstacles. Cells that repeatedly get classified as free cells have lower log odds of being obstacles. Unexplored regions remain with 0 log odds. The pdf of the map is recovered from the log odds using the formula:

$$p(x) = 1 - \frac{1}{1 + exp(probmap)}$$

Once the pdf is recovered, thresholds were applied.

If the cell had a probability value greater than 0.65, it was considered occupied.
If the cell had a probability value less than 0.35, it was considered free space.
If the cell had a probability value between 0.35 and 0.65, it was considered unexplored.

The output of dead reckoning for dataset 0 is shown in figure 1.

Fig. 1. map0 Dead Reckoning

5 Localization prediction

In this step, the particles are initialized and their locations are predicted at each instant. This is done by the following steps:

5.1 Initialization:

The particle poses are initialized as $(0,0,0)$ since in the global frame, the position where the robot starts is $(0,0,0)$. The particle weights, denoted by α are initialized as $\frac{1}{n}$.

5.2 Convert to local frame:

The odometry cannot be trusted. That's the basis of using particle filters for localization. However, the change in odometry can be trusted. The change in pose between the current step and previous time step is converted to the local frame by rotating through $R(\theta)^T$ where $R(\theta)$ indicates the rotation matrix given by θ. In this case, θ is the yaw in the previous time step.

5.3 Apply the motion model:

For each particle, using the yaw estimate of that particle from the previous step, each scan is converted into the global frame. The position of the particle is predicted by adding the change in position in the global frame with the position of the particle at the previous timestep. Noise drawn from an i.i.d gaussian distribution is added to represent uncertainty.

Thus, at the end of this step, we have n particles, whose positions are predicted based on the change in pose from the odometry data and their belief state in the previous timestep.

6 Localization update:

In the update step, for each particle, the following steps were performed:

6.1 Convert scan to world frame:

For each scan z_t, it was transformed to the world frame. The transformation was done by using the state of the particle computed in the predict state.

6.2 Remove invalid scan points:

Scan points that hit the ground were removed using a threshold for the z axis.

6.3 Correlate with map:

The "hit" cells are the cells where the lidar scan hit an obstacle. For each particle, these cells are correlated with the hit cells of the map at the previous time step. In other words, the robot cannot have moved by too much in one time step. Hence, the robot's obstacle environment at this instant should be highly correlated with it's environment in the last time step. This is done by identifying the obstacle cells of this time step and checking the number of obstacle cells at those indices in the map generated at the previous time step. This is simply done by summing up the cells in the previous map. This serves as a correlation index. Denoted by C_k.

6.4 Update Weights:

α_k denotes the weight for each particle k. α_k is initially initialized as $\frac{1}{n}$. However, we work in the log domain. w_k are calculated as $\log(\alpha_k)$. The weights are updated as follows:

The weight of each particle is added with the correlation index of that particle, C_k.
The weight of each particle, w_k is subtracted by the log sum exponent of w_k.
The new α_k is the resulting value weight of the particle.
This method is robust and is translation invariant due to the log sum exponent.

6.4.1 Resample Particles:

A threshold for number of particles is calculated as:

$$n_{eff} = 1/(\sum(\alpha_k))^2$$

If the number of effective particles falls below the threshold, this is considered to be a state of particle depletion. The particles are resampled then. A n_{thresh} value of 5 was used.

Resampling was done by a method called resampling wheel [1].

NOTE: I came across this method in an online udacity course taught by Dr. Sebastian Thrun. I adapted the code partly written by him for this project.

In this method, each particle can be thought of to occupy a sector of a circle corresponding to it's weight. Larger particles occupy a larger area of the circle. If we needed to sample n particles, this algorithm would be repeated n times.

Initially, a random index between 1 to n is generated. A value β is initialized to 0.

For each iteration, to β, is added a value equal to a value between 0 and twice the maximum weight.

While β is greater than the weight of the particle at that index, the weight of the particle is subtracted from β. Index is incremented by 1. The particle currently pointed to by β is appended to the new particles list.

As can be seen, particles with larger weights will get selected multiple times while particles with smaller weights will get selected fewer times or sometimes not.

At the end of the update step, the particle with the maximum weight is considered for building the map.

7 Mapping:

Using the scan for this particle, the map is built similarly to the procedure highlighted in dead reckoning. The obstacles are identified. Cells between the lidar and the obstacle are marked as free using log-odds. The "hit" cells are marked as obstacles using log-odds. The pdf is recovered from the probability map and the map is built.

8 Results:

The results are passable, but not perfect due to improper noise tuning.

The blue indicates robot path (odometry). The Red indicates particle trajectory.

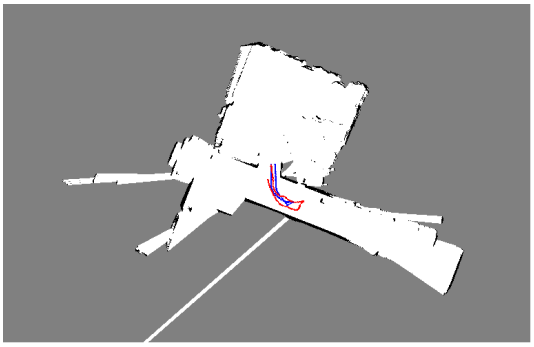

Fig. 2. SLAM output for map 0

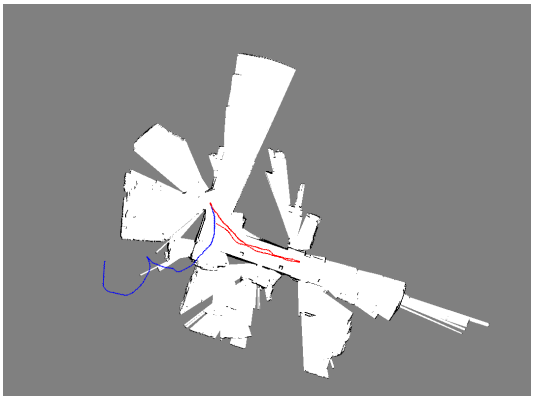

Fig. 3. SLAM output for map 2

9 Things to improve:

9.1 Noise Tuning:

As can be seen, in all the results submitted above, noise tuning wasn't perfect. Randomly drawn I.I.D noise of covariance 0.001 or 0.002 was implemented. Varying the noise covariances and the number of particles will improve the results significantly.

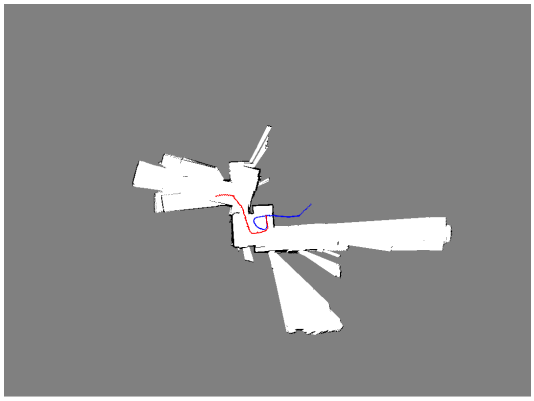

Fig. 4. SLAM output for map 3

9.2 Adaptive number of particles:

More number of particles are required during uncertain conditions to boost the confidence level. This can be seen particularly while the robot is turning corners. One way to resolve this would be to use an algorithm where the number of particles used are adaptive.

One way of implement an adaptive number of particles would be by varying the number of particles as a function of the confidence level of the model. If the confidence level of the model is high, fewer number of particles can be used. If the confidence level of the model is low, higher number of particles can be used to improve the prediction.

Another more established method is to vary the number of particles using the Kullback - Leibler Distance. This method is the KLD-Sampling: Adaptive Particle Filters [2]

9.3 Removing out of bound scans:

In the mapping step, scans who's "obstacle" cells were out of the boundary weren't removed. This resulted in a few invalid scans in the map. Removing invalid scans will improve the result further.

10 Conclusion:

The aim of the project was to simultaneously build the map and localize the robot in the map using Lidar, IMU and Encoder readings. This was done using particle filters as explained above. Due to lack of time, I wasn't able to implement texture mapping. However, this project was a great learning experience. SLAM is a concept that has fascinated me for a while and it was great to implement it on an actual project.

References

[1] Dr. Sebastian Thrun, *AI For Robotics*, Udacity course.

[2] Dieter Fox, *Fox, Dieter. "KLD-sampling: Adaptive particle filters."* NIPS. Vol. 14. 2001.