Partical Filter SLAM

Yahsiu Hsieh
Department of Electrical & Computer Engineering
University of California, San Diego
y4hsieh@eng.ucsd.edu

***Abstract*—This paper presented an implementation of simultaneous localization and mapping (SLAM) with particle filter.**

I. Introduction

Simultaneous Localization and Mapping (SLAM) is a topic of study that has been around for many years. It is the fundamental component in AI problem. Various applications can be found in motion planning, unmanned air vehicle, and self driving vehicle.

This project aims to simultaneously localize and map the robot in an unknown indoor environment using odometry data and LiDAR scanner. A particle filter based approach is implemented to achieve this objective. Particle filters effectively model a probability distribution (dynamic model) as a set of discrete states. Each particle has an associated weight which describes its confidence in its current estimate according to the observation. The THOR robot is used for this project.

The rest of paper is as follows. First we give the detailed formulations of SLAM problem in Section II. Technical approaches are introduced in Section III. And at last we setup the experiment, results are presented in Section IV.

II. Problem Formulations

A. Frame Transformation

The frame transformation problem is that: given a scan data $D \in \mathbb{R}^{1\times l}$ (l is the length of the scan data) in LiDAR frame, try to transform the scan data into world frame $\mathbb{W} \in \mathbb{R}^3$.

B. Localization and Mapping

The robot position $\mu_t \in \mathbb{R}^3$ represents a possible robot 2-D position (x, y) and rotation θ at time t.

a) Localization

The localization problem is that: given an initially unknown environment, we want to find an initial starting position μ_0 of our robot. As we gather more and more measurements from the sensors, the certainty of our robot position μ_t increase.

b) Mapping

The mapping problem is that: given the measurements we obtain from the sensors, we want to construct a reasonable map around the robot. As we gather more measurements from the sensors, the map would gradually become perfect.

III. Technical Approaches

In this section we discuss about algorithms and models used in this project.

A. Frame Transformation

The received LiDAR scan data need to be transformed from LiDAR frame to world frame in order to construct 2D occupancy map. Equation (1) shows the overall transformation matrix.

$$ {}_WT_L = {}_WT_{BB}T_{HH}T_L \tag{1}$$

where $\mathbb{W}$, $\mathbb{B}$, $\mathbb{H}$, $\mathbb{L}$ represents world frame, body frame, head frame, and LiDAR frame respectively. Moreover, the transformation matrix can be calculated:

$$ {}_WT_B = \begin{bmatrix} cos(\tau) & -sin(\tau) & 0 & x_{diff} \\ sin(\tau) & cos(\tau) & 0 & y_{diff} \\ 0 & 0 & 1 & 0.93 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

where x_{diff}, y_{diff} are the absolute position of the robot in the world frame, τ is the torso yaw of the robot.

$$ {}_BT_H = \begin{bmatrix} cos(\psi)cos(\theta) & -sin(\psi) & cos(\psi)sin(\theta) & 0 \\ sin(\psi)cos(\theta) & cos(\psi) & sin(\psi)sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 & 0.33 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

where ψ, θ is the robot neck and head angle, respectively.

$$ {}_HT_L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.15 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Finally, we can transform scan data in $\mathbb{L}$ to $\mathbb{W}$

$$ \begin{bmatrix} x_W \\ y_W \\ z_W \\ 1 \end{bmatrix} = {}_WT_{BB}T_{HH}T_L \begin{bmatrix} x_L \\ y_L \\ 0 \\ 1 \end{bmatrix} \tag{2}$$

B. Mapping

To implement the mapping function, we need four input: the old log-odds map, a scan data in the world frame $\mathbb{W}$, the state of the robot, and the resolution of the map.

Before mapping, the LiDAR scans neglected for invalid ranges. This is done for removing scan data that are too close (0.1m), too far (15m), or hit the ground (z-coord < 0.1m). Then, transformed the robot position and the scan points into map frame to perform Bresenham Algorithm. The grid map is initialized with zeros, the map is then updated by adding

4logγ into occupied cells and minus logγ into free cells, γ was kept as $\frac{0.9}{0.1}$. The values of the log map were limited to 100 to prevent certainty.

C. Dynamic Update

For every partical $\mu_{t|t}^k$, $k = 1 \dots N$ compute:

$$\mu_{t+1|t}^k = f(\mu_{t|t}^k, \mu_t + \sigma_t) \tag{3}$$

where f is the motion model, $\mu_t = (x_t, y_t, \theta_t)$ is the relative odometry input, and $\sigma_t \sim N(0, \sigma)$ is a 2-D Gaussian motion noise.

More specifically, the motion of all the particles are updated according to the odometry data:

$$\begin{bmatrix} x_t^i \\ y_t^i \\ \theta_t^i \end{bmatrix} = \begin{bmatrix} x_{t-1}^i \\ y_{t-1}^i \\ \theta_{t-1}^i \end{bmatrix} + \begin{bmatrix} dx \\ dy \\ d\phi \end{bmatrix} + \begin{bmatrix} N(0, \sigma_x) \\ N(0, \sigma_y) \\ N(0, \sigma_\phi) \end{bmatrix} \tag{4}$$

where, $d\phi = d\phi_i - d\phi_{i-1}$

(x, y, θ) is the pose for each particle i for the t^{th} iteration, and ϕ is the yaw of the robot.

Equation (4) is the motion model f for the particles. The position of each particle is updated by its previous position add the difference in x and y from the odometry data. Gaussian noise with a variance of σ is also added to the position of each particle. The orientation θ of each particle is updated by the difference in the yaw procured from the odometry data. σ_x, σ_y, and σ_ϕ are kept as (0.001, 0.001, $0.01 \times \frac{\pi}{180}$)

D. Measurements Update

Transform the scan z_{t+1} to the world frame using $\mu_{t|t}^k$, for $k = 1 \dots N$ and find all cells y_{t+1}^k in the grid corresponding to the scan. The weights of the particles are updated and resampled if required.

a) Weight Update

The weights of the particles are set based on the LiDAR hits according to the pose for each particle. The particle whose map corresponds highly to the previous log odds map is given more weight. In this case, the weights are calculated by summing up over the LiDAR hits or occupied cells. The current weight vector is multiplied by its previous weight which is used as a prior here to use the previous information. This is then normalized to sum up to 1. Equation (5) and (6) shows our weight update and laser correlation model.

$$p_h(z_{t+1} |\, \mu_{t|t}^k, m) \propto exp(corr(y_{t+1}^k, m)) \tag{5}$$

$$corr(y, m) := \sum_i 1(m_i = y_i) \tag{6}$$

b) Resampling

Resampling is done if the following condition is satisfied:

$$N_{effective} = \frac{1}{\sum_i^N w_i^2} < \gamma \times N \tag{7}$$

where N is the number of particles, and $\gamma = 0.3$

When resampled, the weight vector is set to $\frac{1}{N}$. The γ should be less. If it's too large, resampling is done really frequently and at any given time, there will be a few set of particles in the cluster. See Algorithm 1 for more details.

Algorithm 1 Resampling Wheel

Input: Particle weight W
Output: List of chosen resampled particle index τ

```
N ← number of particles
τ ← []
index ← uniformly chosen from U(0, N − 1)
while i < N do
    β ← β + U(0 . . . 2 × w_max)
    while w_index < β do
        β ← β − w_index
        index ← (index + 1) % N
    end while
    τ ← append index
    i ← i + 1
end while
return τ
```

IV. Results

In this section I will present the results of dead-reckoning and particle filter SLAM for 5 different datasets, including their corresponding paths and log-odds maps.

A. Dead-Reckoning

In order to check the correctness of the motion model, dead-reckoning is implemented. We only one particle prediction with zero motion noise and the new LiDAR measurement to update the log-odds map. The results are shown in below.

B. Particle Filter

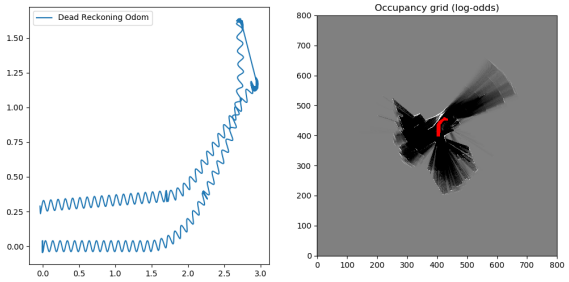
In the particle filter, we have total 128 particles. Every prediction and update iteration, we then choose the particle with the largest weight to be the best particle. The results are shown in below.

C. Conclusion

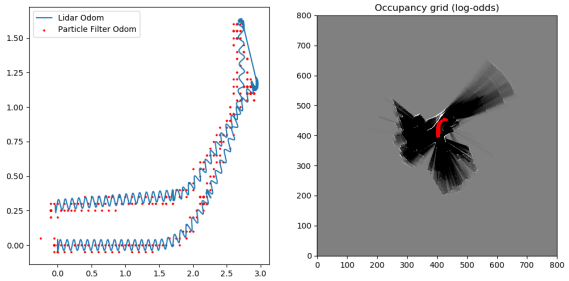
From the results above, we can see that particle filter SLAM seems not that different to the deck-reckoning. Although particle filter does pick the best particle state at each time stamp, the particle filter does not improve the result so much. Below are some possible reasons:

- The number of particles (in this case 128) are still too small, leading to the similarity to dead-reckoning. However, the time penalty will rise significantly.
- The variance of our motion model is too small. However, with small amount of particles, larger variance may lead to even worse result.
- Different resampling method may help improve the result.

In conclusion, the result of particle filter SLAM is good, but could still be improved with the methods mentioned above.

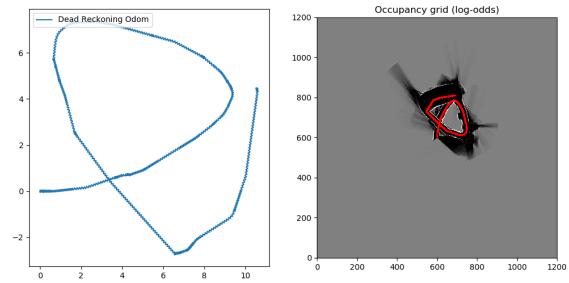


(a) Dead-Reckoning.

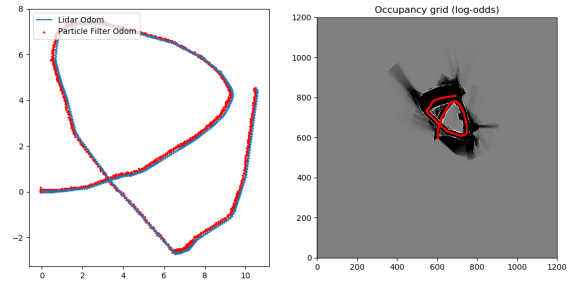


(b) Particle Filter SLAM.

Fig. 1: Dataset 0

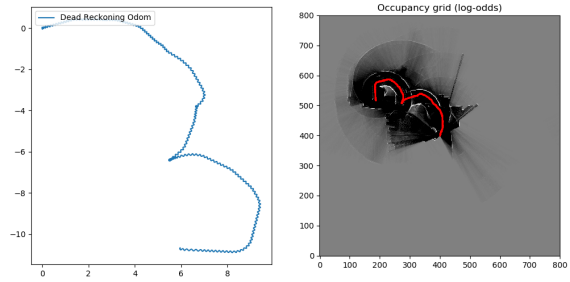


(a) Dead-Reckoning.

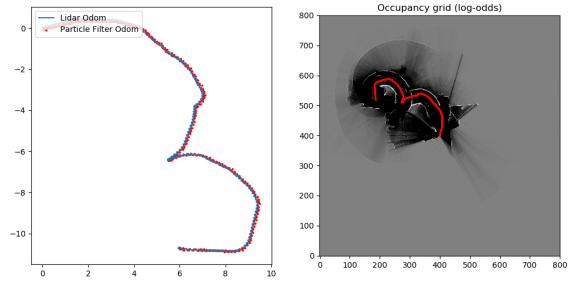


(b) Particle Filter SLAM.

Fig. 2: Dataset 1

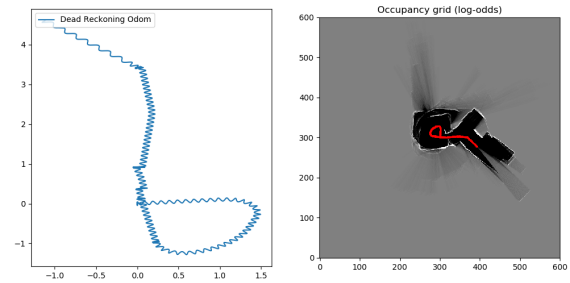


(a) Dead-Reckoning.

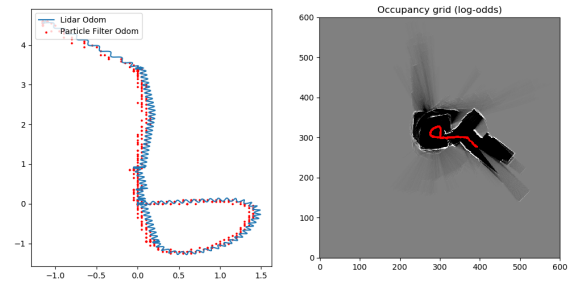


(b) Particle Filter SLAM.

Fig. 3: Dataset 2

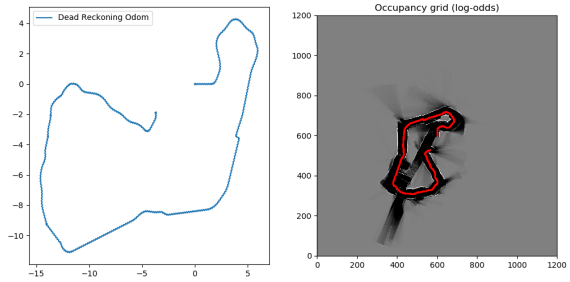


(a) Dead-Reckoning.

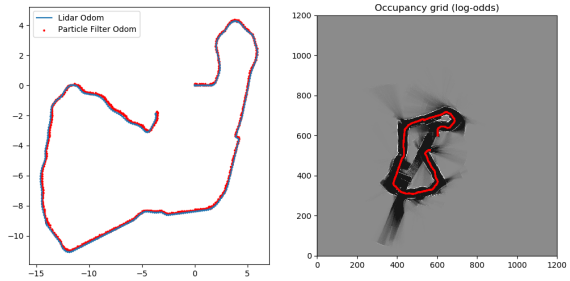


(b) Particle Filter SLAM.

Fig. 4: Dataset 3



(a) Dead-Reckoning.



(b) Particle Filter SLAM.

Fig. 5: Dataset 4