Particle Filter SLAM

Jiaming Lai[1]

I. INTRODUCTION

Particle Filter SLAM contains three part: (1) use Lidar odometry data to estimate current robot state and perform particle filter predict step; (2) based on the prediction result, update particle and corresponding weight, resampling if needed; (3) generate occupied map. In general, this is a simultaneous localization and mapping task. This problem is quite important for the following reasons: (1) an autonomous vehicle or robot should be able to localize itself given sensor data, for example, 2D Lidar scan and RGB-D camera frame, etc; (2) the vehicle or robot should also generate environment map, since this map could help correct localization and navigation.

In our case, we propose particle filter slam method to localize robot and generate environment map. In every iteration, given a set of lidar odometry data, 2D Lidar scan data and joint data, we estimate robot body pose in world frame and perform particle filter prediction step. We then perform update step to generate new particle and weight, and estimate robot body pose in world frame. In the end, we use robot body pose as well as lidar scan to compute lod-odd map, and then update our occupied map.

Data set was divided into two part: date_0, data_3 and data_4 contain RGB and depth information, hence these set could be used for generating texture map. date_1 and data_2 don't contain RGB and depth information, hence these are only used for generating occupied map.

The rest of paper is arranged as follows. First we give the detailed formulations of particle filter SLAM problem in Section II. Techinical approaches are introduced in Section III. And in the end we setup the experiment, results and discussion are presented in Section IV.

II. PROBLEM FORMULATION

A. Robot

1) *Set-up:* The center of mass keeps 0.93 meters above ground. Robot head is 0.33 meters above center of mass and lidar is 0.15 meters above head. Robot overview please see Fig. 1.
2) *Robot Body Pose:* In our case, robot body pose is represented by center of mass pose in world frame. We simplify our model that the center of mass keeps 0.93 meters above ground. We also ignore Raw and Pitch rotation of robot body. The robot body pose in world frame is represented as following:

$$T_w = [x, y, yaw]^T \in \mathbb{R}^3$$

One of our objective is estimate robot body pose T_w given sensor and map information.
3) *Joint Angle:* As we can see in Fig. 1, the head motor and neck motor could perform Pitch and Yaw rotation respectively. Those angles are provided in our data set. The joint angles are represented as following:

$$J = [\theta_n, \theta_h]^T \in \mathbb{R}^2$$

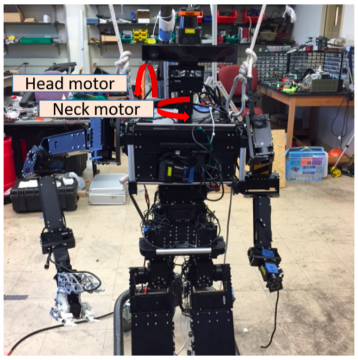

Fig. 1: Robot overview

B. Lidar

In our case, lidar provides two information:

1) *delta pose:* robot body relative odometry between last reading, represented by:

$$\Delta\mu = [\Delta x, \Delta y, \Delta yaw]^T \in \mathbb{R}^3$$

2) *lidar scan:* 2D lidar scan data with range from -135^o to 135^o, represented by:

$$S_l \in \mathbb{R}^{1\times 1081}$$

C. Particle Filter

Given the sensor and robot information above, we perform particle filter algorithm in every iteration. We use it to correct robot body pose and update environment map.

1) *Particle:* Given the number of particle N, particle filter try to maintain a set of particle in every iteration. The set of particle is represented as following:

$$\mu = [\mu_1, \mu_2, ..., \mu_N]^T \in \mathbb{R}^{N\times 3}$$

where $\mu_i = [x_i, y_i, yaw_i]^T \in \mathbb{R}^3$.
2) *Weight:* Weights are updated in update step. The set of weights is represented as $W = [\alpha_1, \alpha_2, ..., \alpha_N]^T \in \mathbb{R}^{N\times 1}$.

*This work was an project of ECE276A, University of California San Diego

[1]Jiaming Lai is with Master of Department of Electrical and Computer Engineering, University of California San Diego, 9500 Gilman Dr, La Jolla jil136@ucsd.edu

D. Map

After particle filter update step, we maintain a map given lidar scan data. Given the grid number on x and y axis, the map is represented as $\{M \in \mathbb{R}^{N_x \times N_y}\}$. More detail about updating map please see techinical approaches in Section III.

III. TECHNICAL APPROACH

In this section, we will discuss about the algorithms and methods used in our project.

A. Algorithm Overview

Suppose we have K frames of lidar scan and our down sample rate is 2, the overview of our method is show as following.

Algorithm 1: Algorithm Overview

```
Result: map, robot track
initialization;
generate map with 1st lidar scan;
for it = 2,3,...,K do
    estimate new robot pose;
    particle filter predict step;
    if it % 2 = 0 then
        particle filter udpate step;
        estimate new robot pose;
        update map;
        if N_ff > threshold then
            resample step;
        end
    end
end
```

The following subsections would introduce each step in detail.

B. Estimate Robot Pose

We performs this operation at the beginning of every iteration and also after particle filter update step. The initial robot pose is set as $[0,0,0]^T$. In estimate step, we choose the particle with maximum weight as our new robot pose. Hence a new robot pose in world frame is estimated as

$$T_w = \mu_{i^*} = [x_{i^*}, y_{i^*}, yaw_{i^*}]^T$$

where

$$i^* = \max_i(\alpha_i), \alpha_i \in W$$

At the beginning of every iteration, we will perform estimate step and add the new robot pose to our robot track record.

C. Particle Filter

1) *Parameter Initialization:* The initial state of all particle is set as $[0,0,0]^T$. The following are some hyper-parameters name and corresponding meaning.
   - *Number of particle:* number of particles we use. In experiment section, we would try 1200 and 2000 particles respectively. More detail please see Section IV.
   - *Down sample rate:* in order to lower computing time consume, we perform n predict steps and then one update steps. This parameter n is down sample rate. By default, we set it as 2.
   - *Noise:* scalar giving noise level when performing predict step. We should adjust it to get better performance. By default, we set it as $3e-2$.
   - *Resample:* scalar giving threshold of N_{ff} deciding whether to perform resampling. By default, we set it as 25% of the number of particle we use.
2) *Predict Step:* This step use robot body relative odometry from lidar to predict new particle set. Given $\Delta\mu$, a new set of particle is predicted by

$$\mu_{new} = [\mu_1 + \Delta\mu, \mu_2 + \Delta\mu, ..., \mu_N + \Delta\mu]^T$$

We also add noise to every new particle. Noise is generate from a standard Gaussian distribution and multiply with hyper-parameter 'Noise' scalar we say above.

3) *Update Step:* This step update weight of every particle. Given a set of lidar scan S_l, a generated map in previous step and particle set, we first transform lidar scan S_l in lidar frame to S_b in body frame using joint angles $J = [\theta_n, \theta_h]^T$. Then, for each particle, we transform lidar scan S_b in body to S_w in world frame, and then compute correlation value between S_w and map $\{M \in \mathbb{R}^{N_x \times N_y}\}$. Hence we get a set of correlation value of each particle

$$corr = [c_1, c_2, ..., c_N]^T \in \mathbb{R}^N$$

Using softmax function, we get probability distribution

$$p_h = [\frac{exp(c_1)}{\sum exp(c_i)}, \frac{exp(c_2)}{\sum exp(c_i)}, ..., \frac{exp(c_N)}{\sum exp(c_i)}]^T$$

Finally, we update particle weight as following

$$\alpha_{new} = [\frac{\alpha_1(p_h)_1}{\sum \alpha_i(p_h)_i}, \frac{\alpha_2(p_h)_2}{\sum \alpha_i(p_h)_i}, ..., \frac{\alpha_N(p_h)_N}{\sum \alpha_i(p_h)_i}]^T$$

4) *Resample Step:* We perform this step when N_{ff} is lower than a threshold we set. N_{ff} is computed as following:

$$N_{ff} = \frac{1}{\sum_{i=1}^N \alpha_i^2}$$

Our resample strategy is **Stratified and Systematic Resampling**. The strategy is shown in Fig. 2.

D. Map Update

In this section, we maintain a lod-odd map using lidar scan data. Note that, before we perform map update operation, we should estimate current robot pose first. Given a lidar scan in lidar frame S_l, we transform it to lidar scan S_w in world frame using current robot pose and joint angles. We use **cv2.drawContours** to get mask of lidar scan cover area in map. The grids in this contour interiors are regarded as free cell while grids outside this contour interiors are regarded as occupied. The initial log-odds of the map is set as $\lambda_{i,0} = 0$, where i represents grid. In No.t iteration, the map log-odds are updated as following:

$$\lambda_{i,t} = \lambda_{i,t-1} - log(4), if\ grid\ i\ is\ regarded\ as\ free$$

$$\lambda_{i,t} = \lambda_{i,t-1} + log(4), if\ grid\ i\ is\ regarded\ as\ occupied$$

Stratified (low variance) resampling

Input: particle set $\left\{\boldsymbol{\mu}^{(k)}, \alpha^{(k)}\right\}_{k=1}^{N}$
Output: resampled particle set
$j \leftarrow 1$, $c \leftarrow \alpha^{(1)}$
for $k = 1, \ldots, N$ **do**
    $u \sim \mathcal{U}\left(0, \frac{1}{N}\right)$
    $\beta = u + \frac{k-1}{N}$
    while $\beta > c$ **do**
        $j = j + 1$, $c = c + \alpha^{(j)}$
    add $\left(\boldsymbol{\mu}^{(j)}, \frac{1}{N}\right)$ to the new set

Fig. 2: Resample strategy

The map is shown using **plt.imshow** with **cmap=gray**. Example from data_0 is shown in Fig. 3. The black grids represent free area, and the white grids represent occupied area.

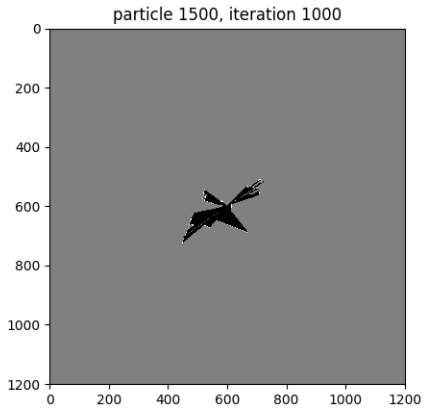

Fig. 3: Map example

IV. RESULTS

In this section, we will shown and discuss results generated from data set. We will show map in some iteration and final map with robot trajectory. The **green line** represents robot trajectory.

A. Data_0

The final map with robot trajectory is shown in Fig. 4.

B. Data_1

For this data set, we try two different number of particle. The final map with robot trajectory is shown in Fig. 5 and Fig. 6. We can see that by adding more particle, our performance has significant improvement. The noise in these two experiment is about $3e-2$.

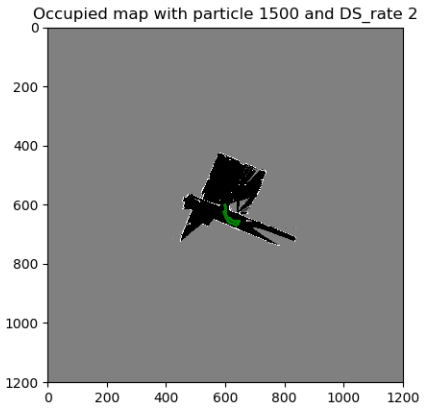

Fig. 4: Data_0 Final Map and Trajectory

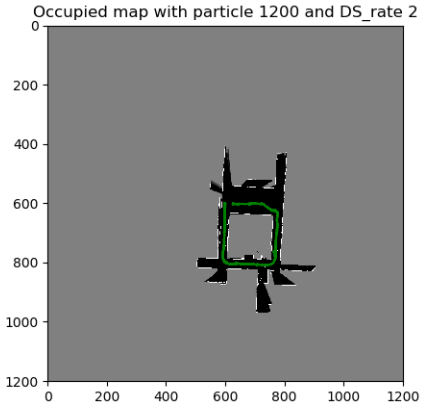

Fig. 5: Data_1 Final Map and Trajectory with 1200 particle

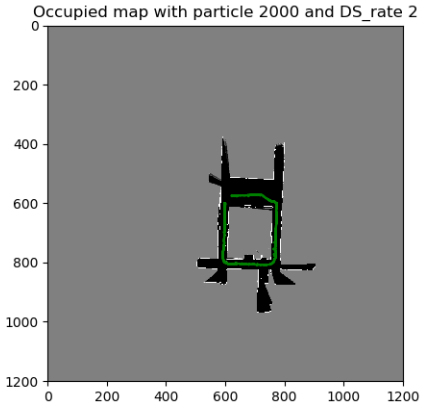

Fig. 6: Data_1 Final Map and Trajectory with 2000 particle

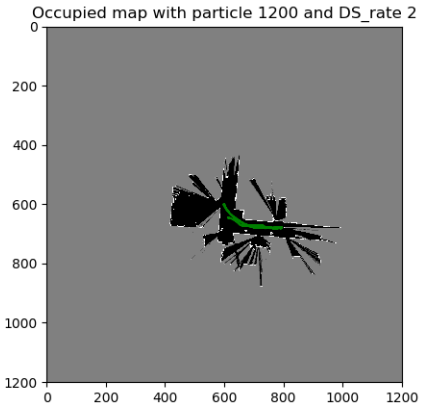

Fig. 7: Data_2 Final Map and Trajectory with 1200 particle

C. Data_2

The final map with robot trajectory is shown in Fig. 7.

D. Data_3

The final map with robot trajectory is shown in Fig. 8.

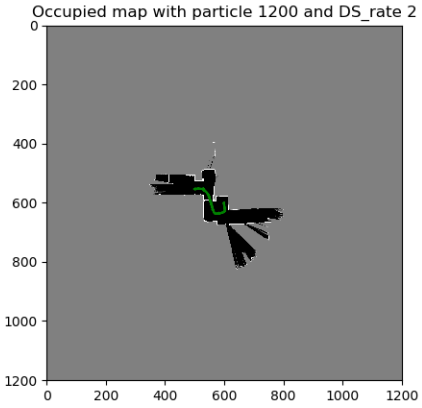

Fig. 8: Data_2 Final Map and Trajectory with 1200 particle

E. Data_4

For this data set, we try two different number of particle. The final map with robot trajectory is shown in Fig. 9 and Fig. 10. The same as data_1, we can see that by adding more particle, our performance has significant improvement. The noise in these two experiment is about $3e-2$. But the final map is still not perfect, because some walls are not parallel. The reason is that, the robot walks for a longer trajectory and perform Yaw rotation more times than the other data set. Hence the deviation of odometry accumulate to relatively large scale.

Fig. 9: Data_4 Final Map and Trajectory with 1200 particle

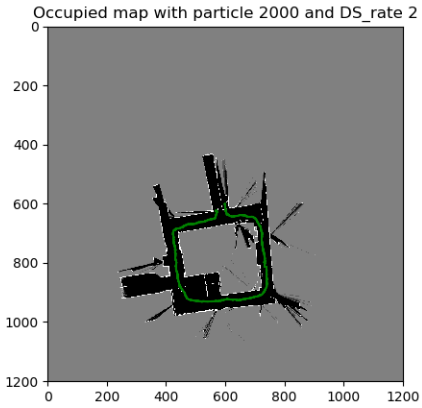

Fig. 10: Data_4 Final Map and Trajectory with 2000 particle

V. DISCUSSION

Our method work pretty well in data_0, data_1, data_2 and data_3. However, in data_4, our method work not well.

As we discuss above, increasing number of particle would significantly improve our final performance. Once we increase the number of particle, we could also increase noise level in particle filter predict step. But note that, increasing particle would lead to more computing time consume. So this is a trade-off we should consider carefully. In some cases, for example situation in data_4, the robot walk in a long trajectory and perform Yaw rotation many times, our method may fail in the end. The deviation would accumulate to a large scale. It is intuitive because our model is simplified.

VI. CITATIONS

The method of updating map using **cv2.drawContours** is inspired by Jiangeng Dong.