Using Particle Filter for SLAM

Yushen Bai
University of California, San Diego
yub025@eng.ucsd.edu

Abstract: This report presents using an occupancy grid for mapping and a particle filter with a laser-grid correlation model for localization. In this work, we first tried mapping from the first scan. Then we implemented a prediction-only particle filter. Once the prediction-only filter works, include an update step that uses scan matching to correct the robot pose. Finally, we projected colored points from the RGBD sensor onto the occupancy grid in order to color the floor.

Index terms: mapping, particle filter, texture map

I. Introduction

In robotic mapping and navigation, simultaneous localization and mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. It asks if it is possible for a mobile robot to be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map. Popular approximate solution methods include the particle filter, extended Kalman filter, and GraphSLAM.

In this report, the method of particle filter was used. The goal is using an occupancy grid for mapping and a particle filter with a laser-grid correlation model for localization. Then using the data in RGBD sensor to get the texture map. The mathematical method was introduced in Section II. The process of mapping and localization was described in Section III. Section IV shows the results and discussion.

II. Problem Formulations

A. Rigid Body Pose

Through an angle θ can be described by a rotation matrix R(θ):

$$s_w = \mathrm{R}(\theta) * s_B = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\gamma & 0 & \sin\gamma \\ 0 & 1 & 0 \\ -\sin\gamma & 0 & \cos\gamma \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{bmatrix} * s_B$$

Where s_w is the position in the world frame, and s_B is the position in the body frame. θ, γ, φ are yaw, pitch, roll angels respectively.

Let B be a body frame whose position and orientation with respect to the world frame W are $\mathrm{p} \in R^3$ and $\mathrm{R} \in \mathrm{SO}(3)$, respectively. The rigid-body transformation is not linear but affine. It can be converted to linear by appending 1 to the coordinates of a points:

$$\begin{bmatrix} s_w \\ 1 \end{bmatrix} = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s_B \\ 1 \end{bmatrix}$$

B. Particle Filter

Used a mixture of delta functions (particles), x is the state:

$$\delta(\mathrm{x};\mu^{(k)}) = \begin{cases} 1 & x = \mu^{(k)} \\ 0 & else \end{cases} \quad \text{for k=1,……,N}$$

with weight $\alpha^{(k)}$ to represent the pdfs $p_{t\,/t}$ and $p_{t+1\ /t}$.

Prior distribution, N is the number of particles in the approximation:

$$x_t\ /z_{0:t}, u_{0:t-1} \sim p_{t\,/t}(x_t) := \sum_{k=1}^{N_{t\,/t}} \alpha_{t\,/t}^{(k)} \delta(\mathrm{x};\mu^{(k)})$$

Prediction: approximate the mixture by sampling:

$$p_{t+1\ /t}(\mathrm{x}) = \sum_{k=1}^{N_{t\,/t}} \alpha_{t\,/t}^{(k)} p_f(x\ /\mu_{t\,/t}^{(k)}, u_t) \approx \sum_{k=1}^{N_{t+1\ /t}} \alpha_{t+1\ /t}^{(k)} \delta\left(\mathrm{x};\mu_{t+1\ /t}^{(k)}\right)$$

Update: rescale the particles based on the observation likelihood, z is the observation:

$$p_{t+1\ /t+1}(\mathrm{x}) = \sum_{k=1}^{N_{t+1\ /t}} \left[\frac{\alpha_{t+1\ /t}^{(k)} p_h(z_{t+1}\ /\mu_{t+1\ /t}^{(k)})}{\sum_{j=1}^{N_{t+1\ /t}} \alpha_{t+1\ /t}^{(j)} p_h(z_{t+1}\ /\mu_{t+1\ /t}^{(j)})}\right] \delta\left(\mathrm{x};\mu_{t+1\ /t}^{(k)}\right)$$

If $N_{eff} = \frac{1}{\sum_{k=1}^{N_{t\,/t}} (\alpha_{t\,/t}^{(k)})^{2}} \le N_{\text{threshold}}$, then resample the particle set via stratified resampling. Stratified resampling guarantees that samples with large weights appear at least once and those with small weights at most once. First, add the weights along the circumference of a circle. Next, divide the circle into N equal pieces and sample a uniform on each piece. Then, Samples with large weights are chosen at least once and those with small weights at most once.

C. Texture Map

From world frame to optical frame:

$$\begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{bmatrix} = \begin{bmatrix} R_{oc}R_{wc}^T & -R_{oc}R_{wc}^T p_{wc} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Where R_{oc} is the rotation from a regular to an optical frame, p_{wc} and R_{wc} are the camera position and orientation in the world frame, respectively.

From optical frame to pixel:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fs_u & fs_\theta & c_u \\ 0 & fs_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \frac{1}{Z_0} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{bmatrix}$$

Where u, v are the position of pixel. The first matrix in the right side is the calibration.

III. Technical Approaches

The algorithms used in this project are discussed in this section.

A. Motion Model

I got data from encoders, IMU, Hokuyo and Kinect. However, the times of them were not corresponded. I built the motion model based on the times of encoder, because they are nearly the same as Hokuyo's. Between two nearby times of encoder, I tried to find if there is any data of IMU. If there exist the data, I calculated the average yaw speed by add them together and divided by the number of them. Then, the yaw speed timed by the time interval to get the yaw angle. I accumulated them by time to get the yaw angle for the whole motion. For the displacement, I set the average of 4 encoders is the displacement in the time interval. The velocity is the displacement divided by time. After that, I can use the formulas below to get the position at each time:

$$\mathrm{x(t)} = \mathrm{x}(t_0) + \mathrm{v} * (\mathrm{t} - t_0) * \frac{\sin\left(\omega * (\mathrm{t} - t_0)/2\right)}{\left(\omega * (\mathrm{t} - t_0)/2\right)} * \cos\left(\theta(t_0) + \omega * (\mathrm{t} - t_0)/2\right)$$

$$\mathrm{y(t)} = \mathrm{y}(t_0) + \mathrm{v} * (\mathrm{t} - t_0) * \frac{\sin\left(\omega * (\mathrm{t} - t_0)/2\right)}{\left(\omega * (\mathrm{t} - t_0)/2\right)} * \sin\left(\theta(t_0) + \omega * (\mathrm{t} - t_0)/2\right)$$

B. Mapping

According to the transformation of rigid body pose, the position of lidar scan can be transformed to the world frame. Then, using the function bresenham2D to figure out the free grids and the edge, and label them by positive and negative numbers, respectively. I labeled the free grids 1, while the edge is -1. The value of labels would be added the new labels by time.

C. Prediction

Created some particles. Based on the motion model, additive gaussian noise to compute an updated pose for each particle.

D. Update

For each particle, transformed the laser scan to world frame, and the transformation matrix was depended on the particle's position. Computed the correlation by function mapCorrelation. Then, updated the particle weights according to the particle filter equations, and chose the best particle for the next update. If $N_{eff} = \frac{1}{\sum_{k=1}^{N_{t\,/\,t}} (\alpha_{t\,/\,t}^{(k)})^{2}} \leq N_{\mathrm{threshold}}$, then resample the particle set via stratified resampling.

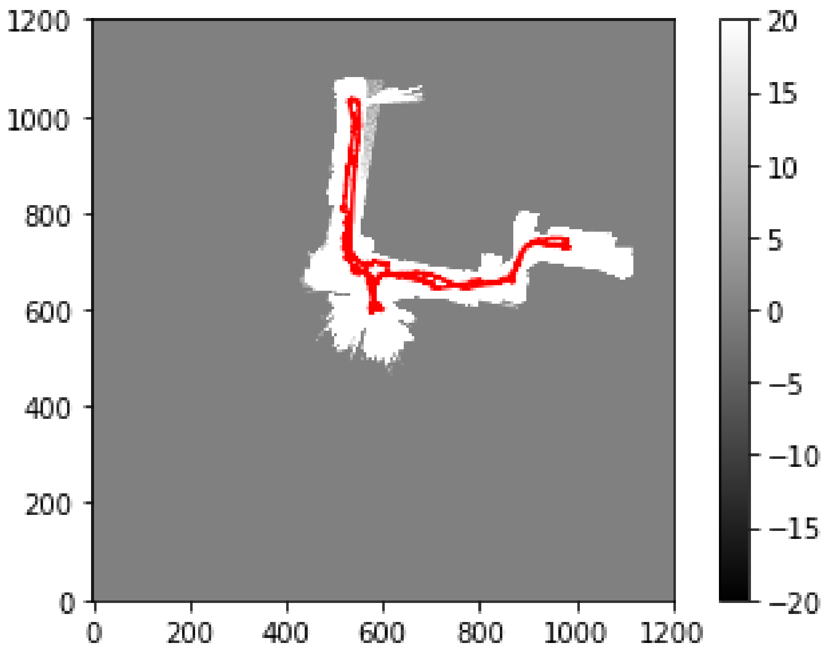
E. Texture Mapping

Transformed the data of RGB image and depth image to the world frame. By formulas in Section II, found the ground plane via thresholding on the height (Zw=0). Then colored the cells that belong to the ground plane. To increase the running speed of the code, I built a matrix included all the pixel positions of an image. Then did the matrix multiplication. It was faster than the for loop.

IV. Result and Discussion

A. Set 20

Figure 1 shows the image of the SLAM system over time and the textured map of set 20. I change the direction of the vertical axis in the SLAM image, so it is opposite to the textured map. As shown in the figure, it works well.



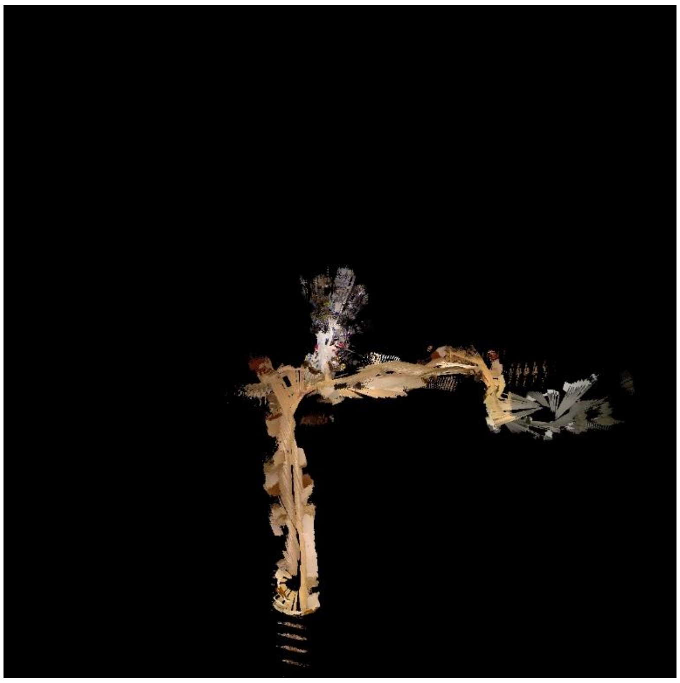


Figure 1. the image of the SLAM system and the textured map of set 20

B. Set 21

Figure 2 shows the image of the SLAM system over time and the textured map of set 21. I change the direction of the vertical axis in the SLAM image, so it is opposite to the textured map. As shown in the figure, there is a losing part in the textured compared with the SLAM image. It might because the motion angle of the robot changed so sharp at some times, and the position data did not correspond to the right RGB data.

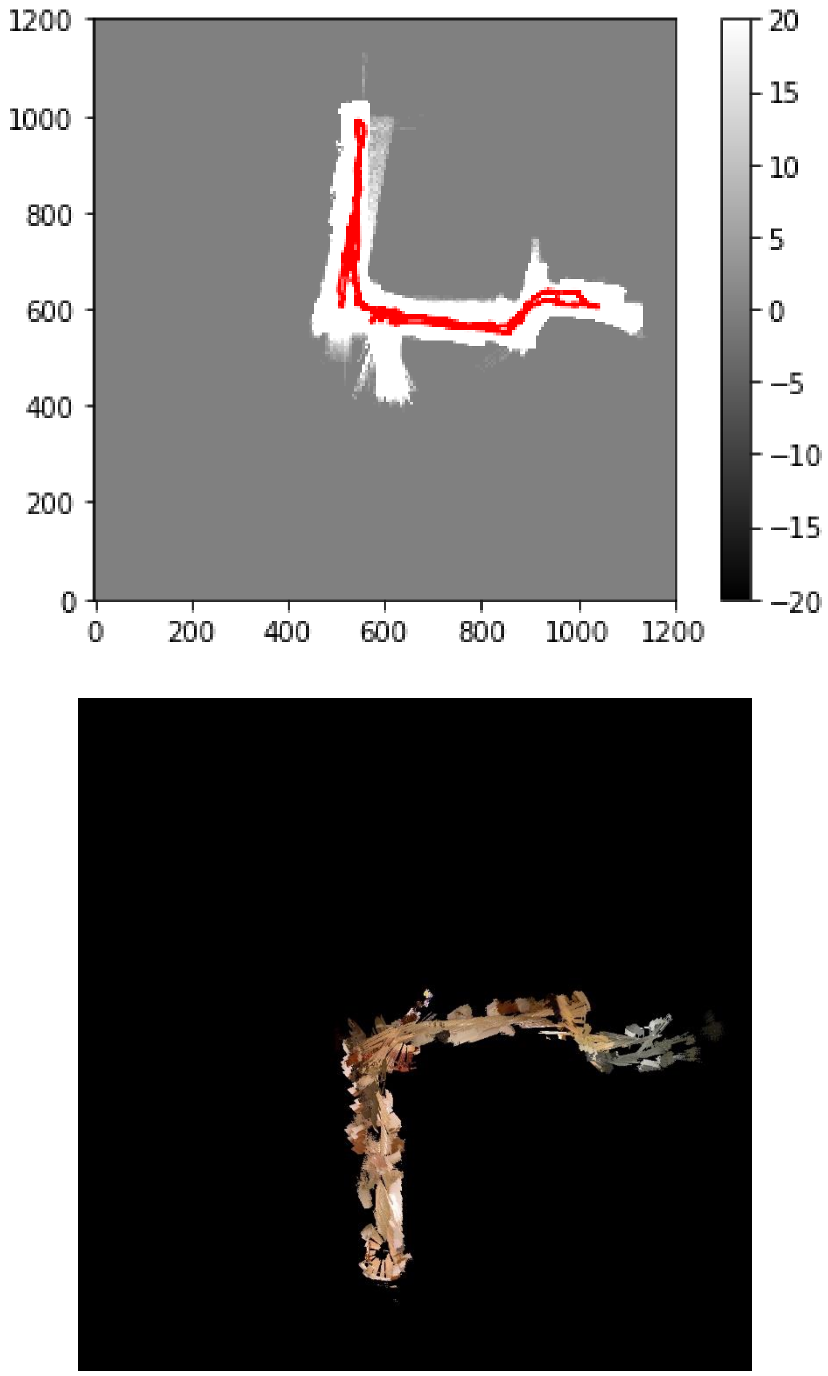


Figure 2. the image of the SLAM system and the textured map of set 22

C. Test Set

Figure 3 shows the image of the SLAM system over time and the textured map of test set. As shown in the figure, it works well.

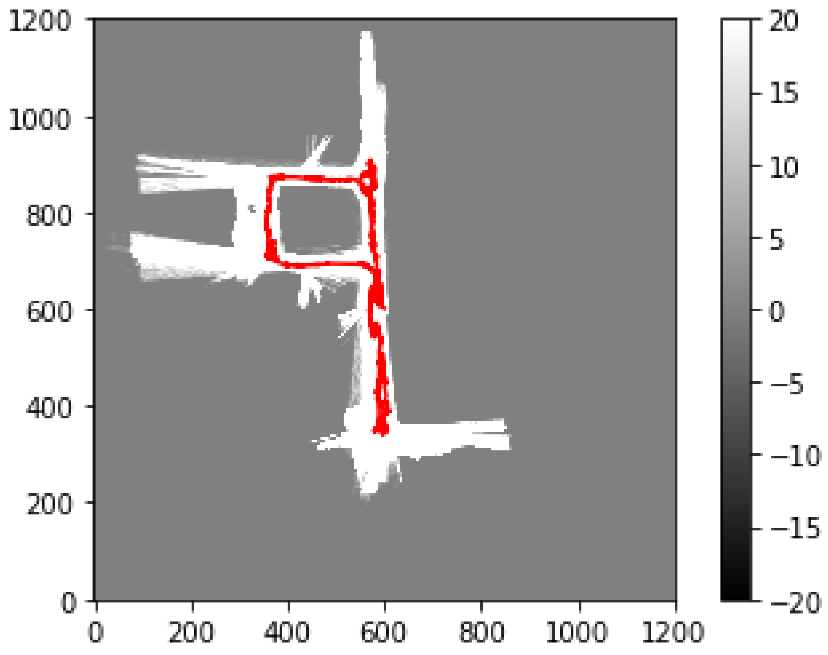


Figure 3. the image of the SLAM system of test set