Simultaneous Localization and Mapping (SLAM) Using Particle Filter

Jingpei Lu
Jacobs School of Engineering
University of California, San Diego
jil360@ucsd.edu

***Abstract*—This paper presented my work of using particle filter to implement SLAM Algorithm. Our goal is to use the given LIDAR, IMU, and encoder data to localize our robot and create the map simultaneously. In our implementation, we will use the laser observation model and differential-drive motion model and use particle filter to predict and update. Then we will use RGBD data to create the texture map.**

Keywords—SLAM, Particle Filter, Localization, differential-drive motion model, Laser observation model, Texture mapping

I. INTRODUCTION

Simultaneous localization and mapping (SLAM) is a fundamental and important problem in robotic mapping and navigation. It 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. Its applications including self-driving cars, unmanned aerial vehicles, and autonomous underwater vehicles.

Nowadays, there are several popular algorithms known for solving or providing approximate solution for the SLAM problem. For example, Particle filter, Kalman filter, and GraphSLAM. Particle filters comprise a broad family of sequential Monte Carlo algorithms for approximate inference in partially observable Markov chain. Using the particle filter, we are able to solve the localization problem and estimate the robot's pose. In this paper, we are going to focus on the math of the particle filter and its implementation on predicting the pose and localizing the robot using the given data. Specifically, we are going to use the IMU, odometry, and laser measurements to localize the robot and build a 2-D occupancy grid map of the environment.

Our approach of solving the SLAM problem can be divided into two major parts, prediction and update. We treat each particle as one of our potential robots. In prediction step, we are going to use the differential-drive model to predict the motion of each particle. In the update step, we are going to use the laser scan from each particle and use the laser observation model to compute map correlation. Then we choose the particle has the best map correlation, project the laser scan, and update the map. Finally we can use the RBGD data to texturize the map.

II. PROBLEM FORMULATION

A. *Structure of Robotics Problem*

In robotics problem, we are generally interested in these parameters: time t, robot state x_t, control input u_t, observation z_t, environment state m_t. Usually, the sequence of control inputs and observations are assumed known, and we are interested in estimating the robot states and environment states. Under the Markov Assumptions, the state x_{t+1} only depends on the previous input u_t and state x_t, and the observation z_t only depends on the robot state x_t and the environment state m_t.

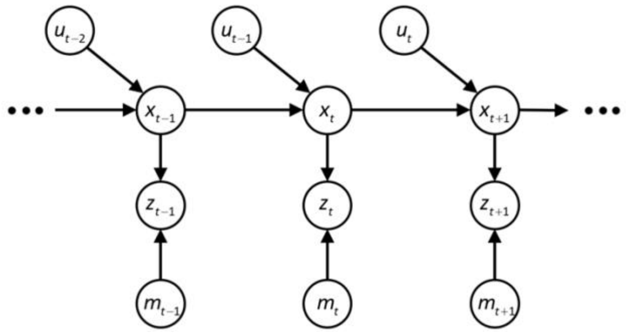


Figure 1

Figure 1is the Markov Chain which representing the relationship between the parameters.

Under Markov Assumption, we can decompose the joint probability density function of state $x_{0:t}$, observation $z_{0:t}$, and controls $u_{0:t-1}$ as below

$$p(x_{0:T}, z_{0:T}, u_{0:T-1}) = p_{0|0}(x_0)\prod_{t=0}^{T} p_h(z_t|x_t)\prod_{t=0}^{T} p_f(x_t|x_{t-1}, u_{t-1}) \quad (1)$$

where $p_{0|0}(x_0)$ is prior, $p_h(z_t|x_t)$ is observation model, and $p_f(x_t|x_{t-1}, u_{t-1})$ is motion model. We will discuss these models and how to use them to predict and update the probability of robot states for our particle filter later.

B. *Mapping*

In our case, we don't have the ground truth map beforehand. So, one way to approximate the map of the environment is to do the lidar-based mapping. Given the robot trajectory $x_{0:t}$ and a sequence of lidar scans $z_{0:t}$, we are going to build an occupancy grid map m of the environment.

$$m_i|z_{0:t} = \begin{cases} 1\ (occupied), & with\, prob.\, \gamma_{i,t} \\ 0\ (free), & with\, prob.\, 1-\gamma_{i,t} \end{cases} \quad (2)$$

Where m_i is the i th cell of our map m and $\gamma_{i,t}$ is the occupancy probability of ith cell in time t.

$$\gamma_{i,t} := p\big(m_i = 1\big|z_{0,t}, x_{0,t}\big) \quad (3)$$

For a binary random variable m_i , maintaining the probability density function is equivalent to maintain the log-odds $\lambda_{i,t}$ of each cell, since

$$\gamma_{i,t} = 1 - \frac{1}{1+\exp\big(\lambda_{i,t}\big)} \quad (4)$$

where $\lambda_{i,t}$ can be obtained by

$$\lambda_{i,t} = \lambda_{i,t-1} + logg_h(z_t|m_i, x_t) \quad (5)$$

Here, $g_h(z_t|m_i, x_t)$ is our believes relating to the observation model. To simply the model, we choose it to be 4 if it is occupied and ¼ if it is free.

C. Prediction

For particle filter, the prior $p_{t|t}(x)$ at time t is defined by

$$p_{t|t}(x) = \sum_{k=1}^{N} \alpha_{t|t}^{(k)} \delta\left(x_t; \mu_{t|t}^{(k)}\right) \tag{6}$$

where N is the number of particles, $\alpha_{t|t}^{(k)}$ is the weight of kth particle at time t and $\mu_{t|t}^{(k)} \in SE(2)$ is the pose (state) of the kth particle at time t.

Given a new control input u_t and current state x_t, the predicted probability density function of particle filter can be represented as

$$p_{t+1|t}(x) \approx \sum_{k=1}^{N} \alpha_{t+1|t}^{(k)} \delta\left(x; \mu_{t+1|t}^{(k)}\right) \tag{7}$$

Here, $\alpha_{t+1|t}^{(k)}$ is the weight of the kth particle at time t+1, and in our case, this should be the same as the weight in time t. $\mu_{t+1|t}^{(k)}$ is the predicted pose of the kth particle, and it can be obtained using the motion model. In our case, we are using the differential-drive motion model. Then for each particle, its estimated pose can be described as

$$\begin{aligned} \mu_{t+1|t} &= f\left(\mu_{t|t}, u_t\right) \\ &= \mu_{t|t} + \tau \begin{pmatrix} v_t sinc\left(\frac{w_t \tau}{2}\right) \cos\left(\theta_t + \frac{w_t \tau}{2}\right) \\ v_t sinc\left(\frac{w_t \tau}{2}\right) \sin\left(\theta_t + \frac{w_t \tau}{2}\right) \\ w_t \end{pmatrix} \end{aligned} \tag{8}$$

Here, τ is the time difference, $\theta_t \in (-\pi, \pi]$ is the yaw angle at time t. $w_t \in \mathbb{R}$ is the rotational velocity (yaw rate), which can be obtained by IMU. $v_t \in \mathbb{R}$ is the linear velocity at time t, which can be obtained by encoder.

D. Update

For particle filter, given a new observation z_{t+1}, we can update the probability density function of particle filter at time t+1 based on the observation model as

$$p_{t+1|t+1}(x) = \sum_{k=1}^{N} \left[\frac{\alpha_{t+1|t}^{(k)} p_h\left(z_{t+1}|\mu_{t+1|t}^{(k)}\right)}{\sum_{j=1}^{N} \alpha_{t+1|t}^{(j)} p_h\left(z_{t+1}|\mu_{t+1|t}^{(j)}\right)}\right] \delta\left(x; \mu_{t+1|t}^{(k)}\right) \tag{9}$$

Here, p_h is our observation model. In our case, we are going to use Laser Correlation model as our observation model. The Laser Correlation model is defined as

$$p_h(\mathrm{z}|x, m) = \frac{\exp\left(corr(y, m)\right)}{\sum_v \exp\left(corr(v, m)\right)} \tag{10}$$

where y is our observation (laser data) transformed into grid map, and m is our grid map. The correlation function $corr$ is defined as

$$corr(y, m) = \sum_i 1\left\{m_i = y_i\right\} \tag{11}$$

So, our observation model is proportional to the correlation of the laser scan and grid map at time t.

E. Projection Function

Given a point $x \in \mathbb{R}^3$ in world frame, we can find the corresponding point $y \in \mathbb{R}^2$ using the following projection function.

$$y = K\pi\left(R_{oc} R_{cw}^T (x - p)\right) \tag{12}$$

Here, $K \in \mathbb{R}^{3*3}$ is an intrinsic parameter matrix, R_{oc} is the rotation from camera frame to optical frame, $R_{cw} \in SO(3)$ is the rotation from world frame to camera frame, p is the transition from world origin to camera position, and π is defined as $\pi(a) = \frac{1}{a_3} a$ for a vector $a \in \mathbb{R}^3$.

III. Technical Approach

A. Initialization and Data Synchronization

Our data set consists laser scans, IMU, and encoder data, but they are from different sensors, so they have different timestamps. In order to use these data, we need to find associate data for each timestamp. To simplify, we are going to use the timestamp of the encoder data as our reference. To find the IMU data and laser scans associate to our reference timestamp, we first find the closest timestamp in IMU data and laser scans. Then, we obtain the data at that timestamp and associate the data with our reference timestamp.

To begin the particle filter, we need to initialize our particle set. Let's say, we have N particles, each particle has its weight and its pose. We initialize the weight of each particle by $\alpha_{0|0} = 1/N$ and pose of each particle $\mu_{0|0} = (0,0)^T$. Here, we arbitrarily set each particle to the origin of our grid map because we know nothing about our environment. Therefore, we can choose whatever place as our initial state. And we set the weight to 1/N uniformly because we don't know anything of the environment.

To initialize the map, we will use the first laser scan to update our occupancy grid map. To transfer the laser scans to our world coordinate, we use this equation

$$R^T(m - p) = \begin{bmatrix} \mathrm{r}\cos(\alpha)\cos(\epsilon) \\ \mathrm{r}\sin(\alpha)\cos(\epsilon) \\ r\sin(\epsilon) \end{bmatrix} \tag{13}$$

where r is range, α is the azimuth, ϵ is the elevation, R is the orientation from lidar frame to world frame, p is the transition from the lidar frame to the world frame, and m is the points in the world frame. Then we transfer m to our grid map and use the bresenham2D algorithm [1] to obtain the occupied grids and free grids. Finally, we can update the log-odds of our occupancy grid map using the equation (5).

B. Mapping and Localizaion

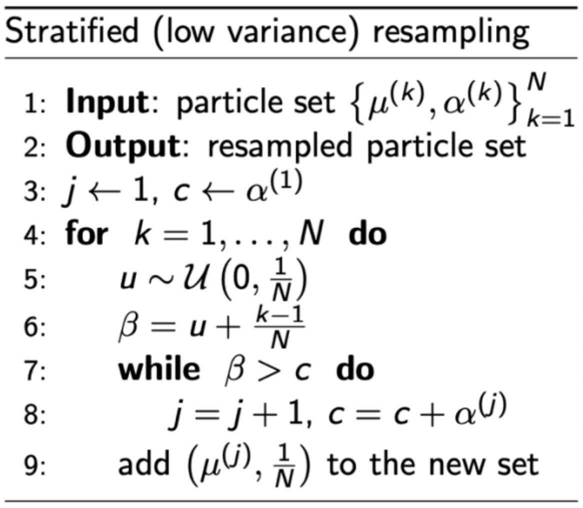
As we said before, we divide the mapping and localization problem into two parts: prediction step and update step. We iteratively apply these two steps to optimize our estimation of the robot pose and the environment map.

During the prediction step, we predict the state/pose $\mu_{t+1|t}^{(k)}(k = 1, \ldots, N)$ for every particle using the differential-drive motion model with the reading of IMU and encoder as we stated in equation (8). In order to obtain a better estimation, we can add 2D Gaussian noise to our prediction.

During the update step, we transform the scan z_{t+1} to the world frame using the equation (12) with the current particle pose $\mu_{t+1|t}^{(k)}$ for k = 1,..,N. Then, we update the particle weights using the laser correlation model we described in (9), (10), and (11). Because the observation model is proportional to the correlation of the laser scans, so we can simplify the observation model by

$$p_h\left(z_{t+1}\middle|\mu_{t+1|t}^{(j)}, m\right) \propto \exp(corr(y, m))$$

To obtain the better performance of particle filter, we can resample our particle set when $N_{eff} \le N_{threshold}$. $N_{threshold}$ is some threshold value and $N_{eff} = 1/\sum_{j=1}^{N} {\alpha_{t|t}^{(j)}}^2$. In our case, we are going to use the stratified resampling method which is described in Figure 2.



Stratified (low variance) resampling

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

Figure 2

After the update step, we should update the map. We choose the best particle (the particle with the largest weight) and transform the laser scans to the grid map using the pose of the best particle as we described in equation (5).

Therefore, the whole process is described as below.

Particle SLAM Algorithm

```
for t in timestamps:
    for all p in particle set:
        **predict**:
        move the particle according to the motion
        model and input $u$
        **update**:
        given the laser scan, compute the map
        correlation and update the weight
    **choose** the best particle and update the map log-odds
    if $N_{eff} \le N_{threshold}$:
        **resample** the particle
```

Finally, we will use the RBG data I_t and depth D_t to texturize the map. First, we have to transform I_t and depth D_t to the world frame. To do that, we need first obtain the coordinates in the camera frame corresponding to each pixel in image and then, transfer the coordinates from the camera frame to world frame using the robot's pose at time t. In this case, we have the image coordinates and we want the coordinates in world frame, which is the inverse transform using the equation we described in (12). Then, we can find the ground plane in the transformed coordinates by thresholding on the height. After this, we can color the map grids using the RGB data of the pixels that belong to the ground.

IV. Result

For my implementation, I add Gaussian noise $\epsilon \sim \mathcal{N}(0, 0.001)$ in prediction step, and I have tried using particle filter with 20 particles and 100 particles. The runtime of our particle SLAM algorithm increases significantly when we add more particles. For 20 particles, the runtime for an episode is about 15 minutes. For 100 particles, it takes about an hour to complete an episode. However, performance wise, the improvement is nor significant. Therefore, when running on the test dataset, I only use 20 particles.

One other thing to mention is the noise. If we want to have better localization performance, we should add the noise with large deviation, because we want to spread the particles more to explore more possibility. However, when it comes to mapping, large noise would significantly affects our update step. We might have bad mapping because the output of the motion model becomes insignificant and the pose of the particle largely depends on the additive noise. Therefore, when we want to do the mapping and localization simultaneously, we have to carefully choose the deviation of the additive noise.

When doing the texture mapping, it is important to have a good deterministic method to find the pixels that belong to the ground. In my implementation, I just finetuned the thresholding value for heights of the world coordinates. However, it is time consuming and not very deterministic. It might be significantly affected by the noise. For example, if there is a hill or if the robot is trembling, the results would be unreliable.

Below are the results of my particle filter SLAM on 3 different datasets.

Images of dataset 20

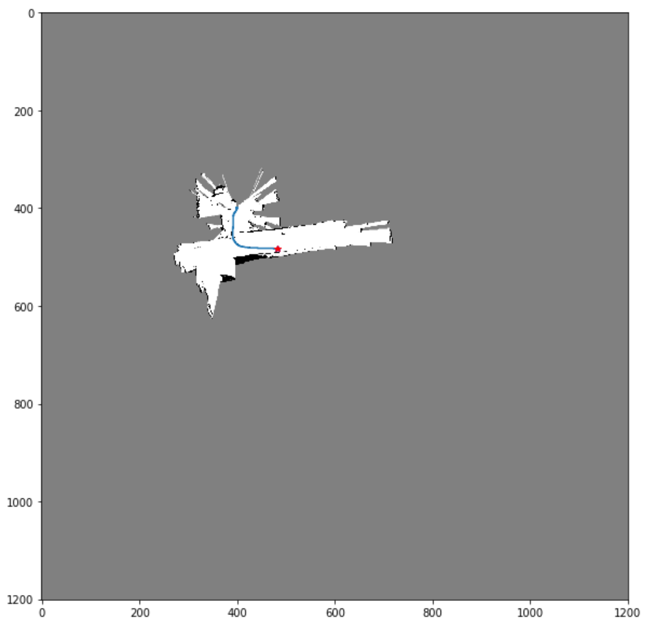


Figure 3

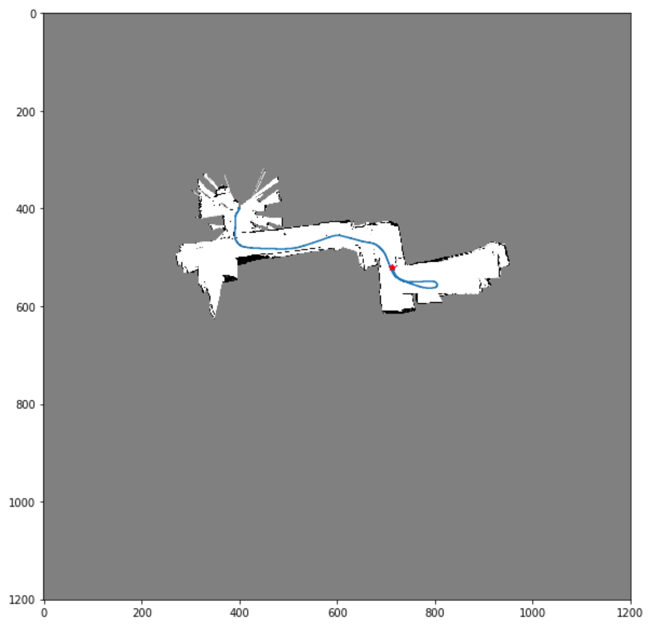


Figure 4

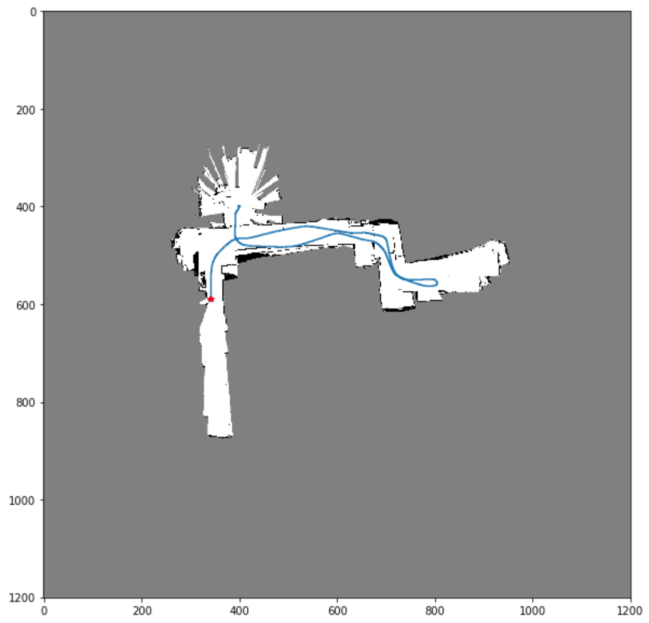


Figure 5

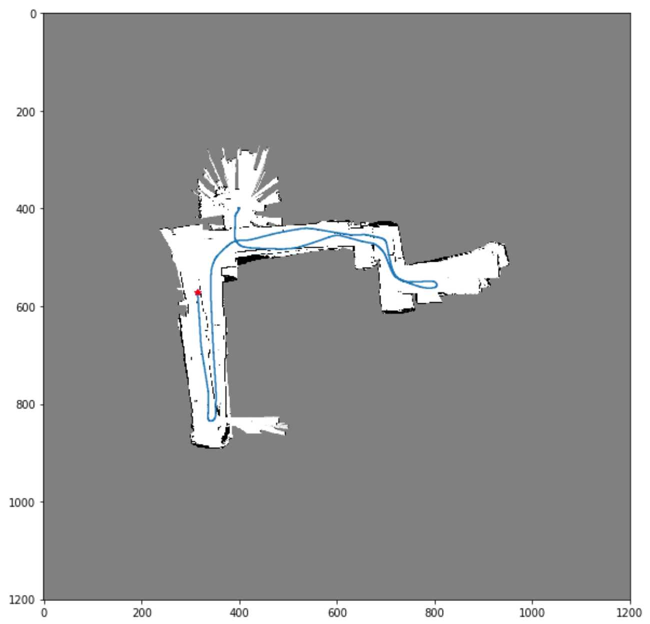


Figure 6

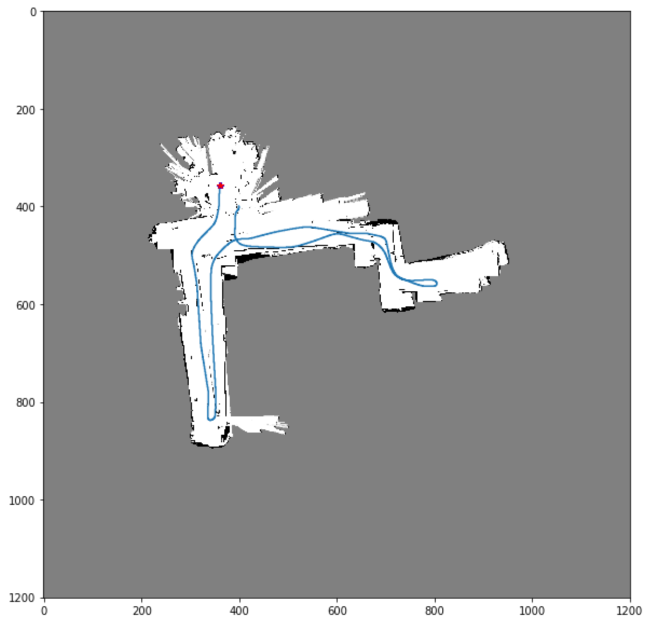


Figure 7

Figure 3, 4, 5, 6, 7 represents the result of particle filter SLAM of dataset 20. Each picture shows how the occupancy grid map looks like over after timestamps. White grids in map represent the free grids, black grids represent the occupied grids, and gray grids indicates the grids are unexplored. The blue line indicates the robot trajectory.

Images of dataset 21

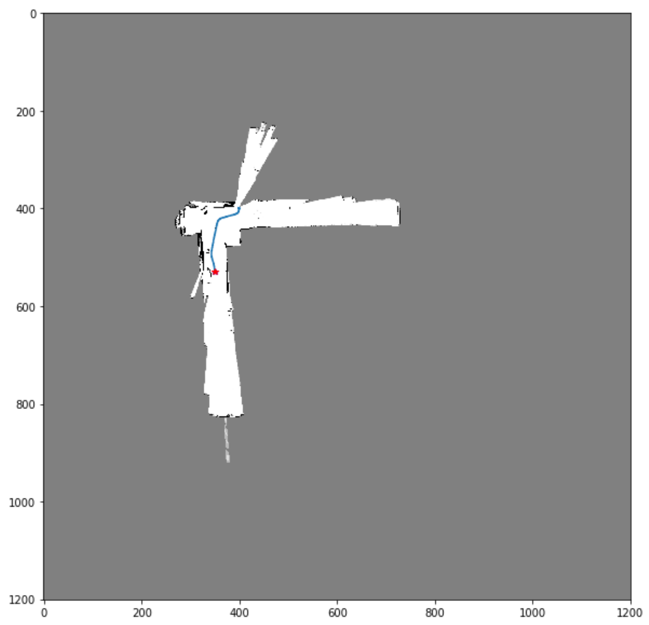


Figure 8

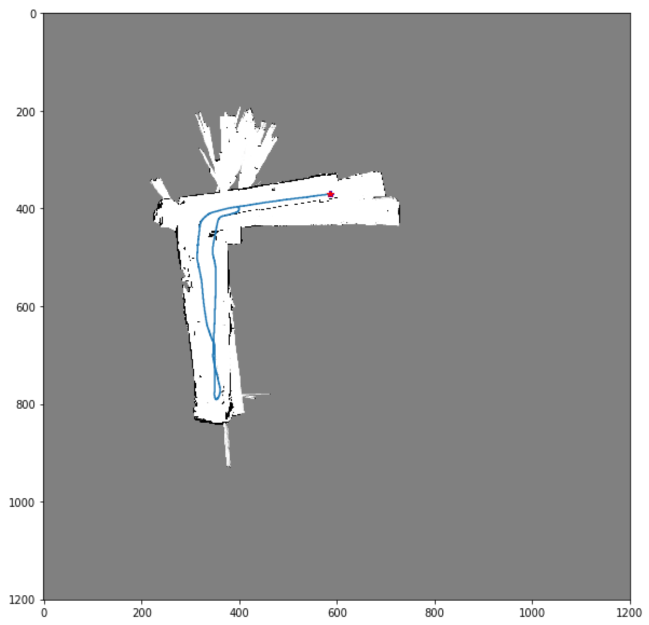


Figure 10

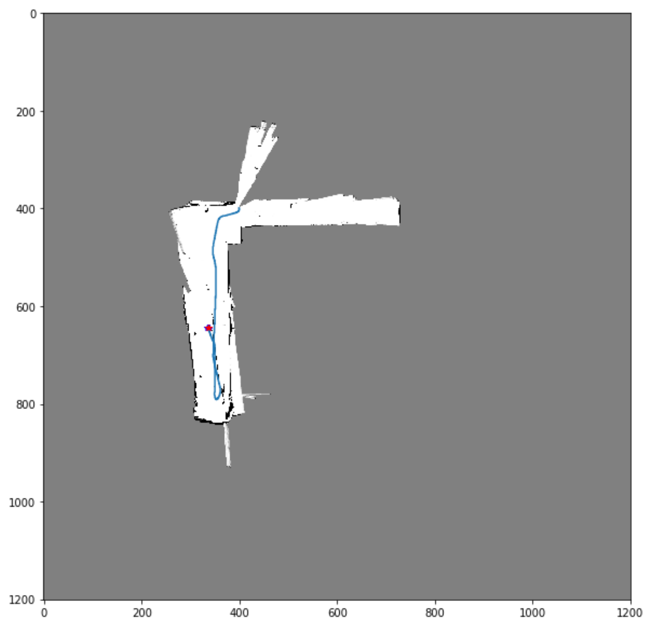


Figure 9

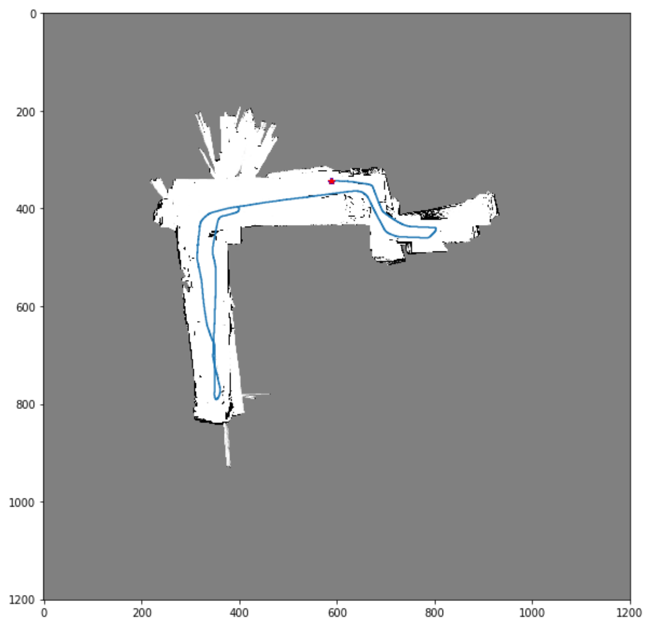


Figure 11

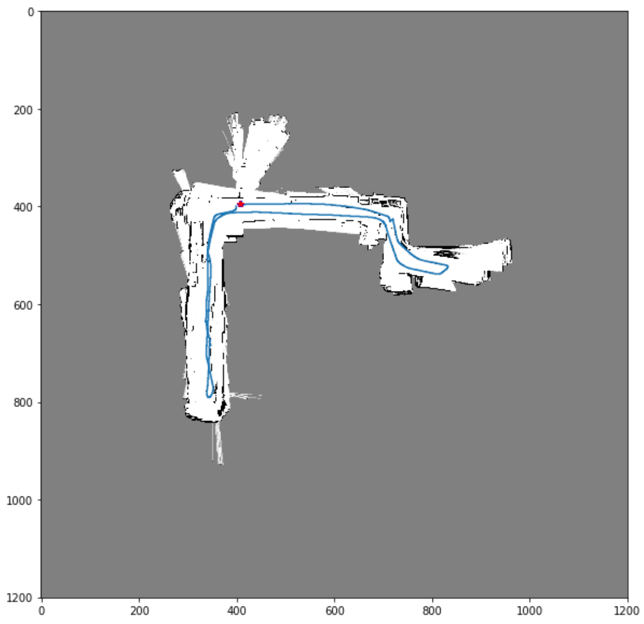


Figure 12

Figure 8, 9, 10, 11, 12 represents the result of particle filter SLAM of dataset 21. Each picture shows how the occupancy grid map looks like after 1000 timestamps. White grids in map represent the free grids, black grids represent the occupied grids, and gray grids indicates the grids are unexplored. The blue line indicates the robot trajectory.

Images of dataset 23

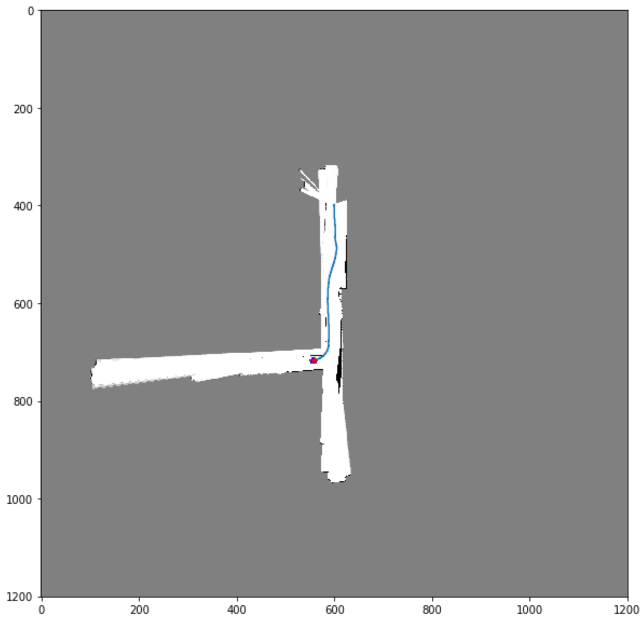


Figure 13

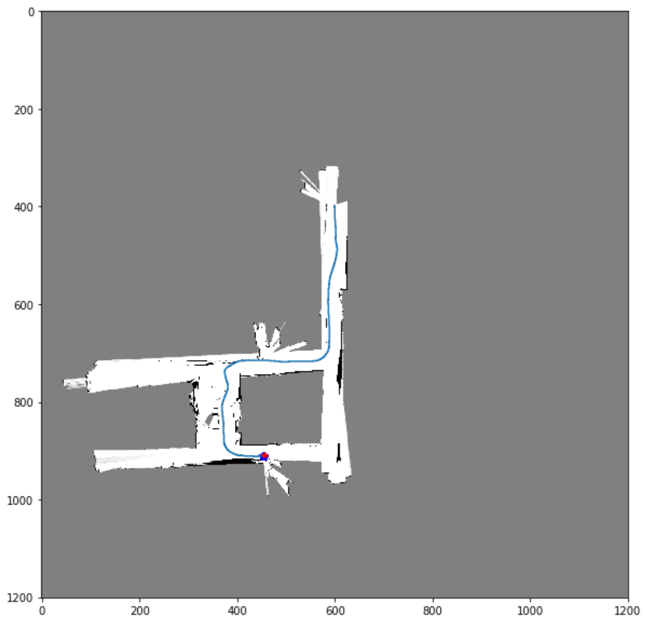


Figure 14

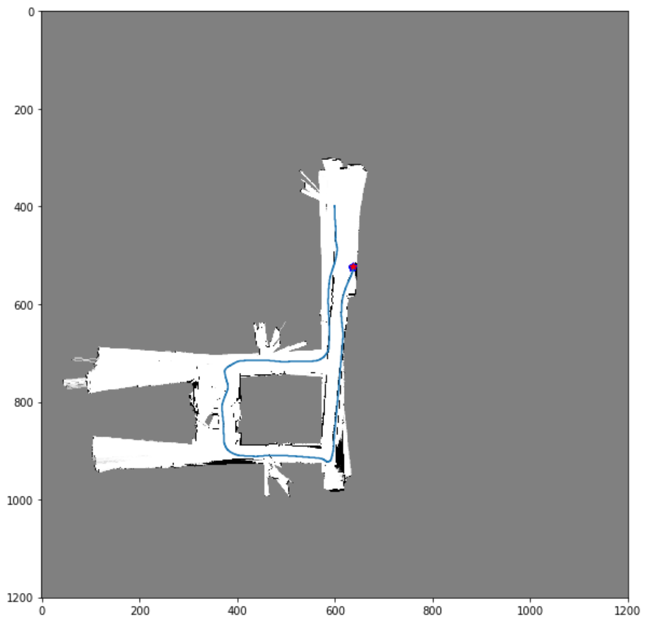


Figure 15

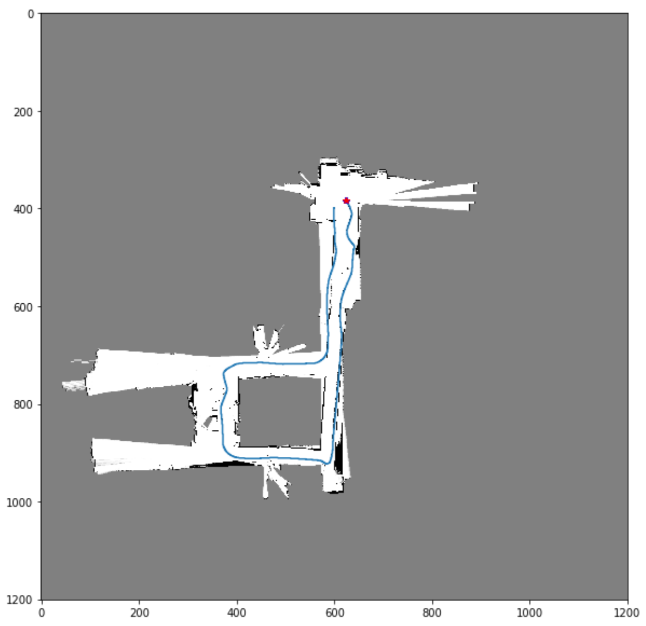


Figure 16

Figure 13, 14, 15, 16 represents the result of particle filter SLAM of dataset 21. Each picture shows how the occupancy grid map looks like after 1000 timestamps. White grids in map represent the free grids, black grids represent the occupied grids, and gray grids indicates the grids are unexplored. The blue line indicates the robot trajectory.

Images of texture map

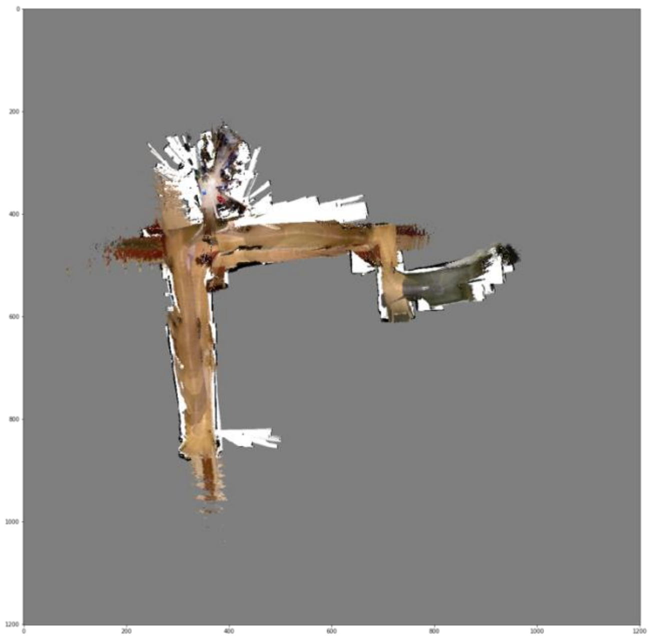


Figure 17

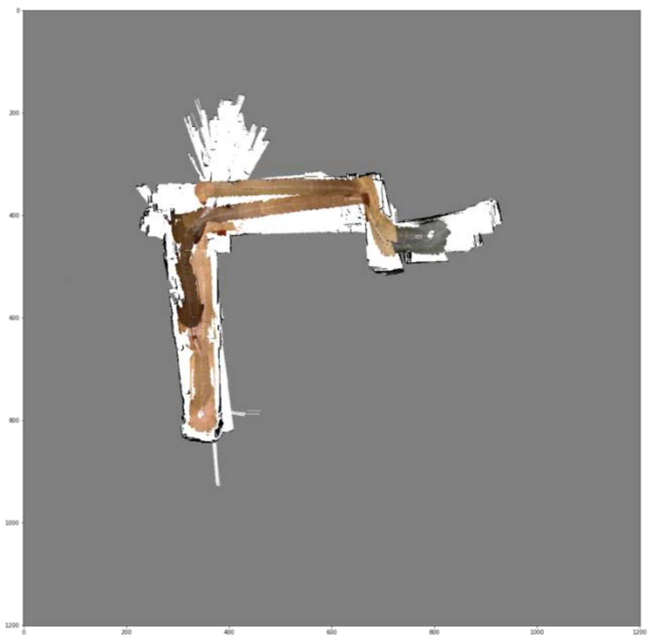


Figure 18

Figure 17 represents the resulting texture map of dataset 20. Figure 18 represents the resulting texture map of dataset 21. The result looks not quite good and I am guessing that is because of the way we determine the ground points is not robust. Although this need some finetune process, it is extremely difficult to get a good threshold value because each episode takes more than an hour to complete.

Conclusion

In this project, we are trying to localize the robot and mapping the environment using the particle filter. After obtaining the poses of robot, with the provided RBGD data, we can color/texturize the map using the RGB value belong to the ground. However, our texture map is not smooth. To create better texture map, we might want to explore more robust and advanced technique to texturize the map.

References

[1] Paul E. Black. Dictionary of Algorithms and Data Structures, NIST. https://xlinux.nist.gov/dads/HTML/bresenham.html