Particle Filter SLAM and Texture Mapping

1st Zhirui Dai
dept. Electrical and Computer Engineering

UC San Diego
San Diego, United States
zhdai@ucsd.edu

***Abstract*—The ability of simultaneously localization and mapping (SALM) is very important for an autonomous robot to sense the environment before it makes decision about how to interact with the world. In this report, based on particle filter, I presented and implemented an SLAM with some useful techniques for improving the robustness and an algorithm for texturing the map with RGBD camera image data when the robot gets a high confidence in its location and the environment. The effect of those techniques I used is detailedly discussed in the following sections.**

I. Introduction

To enable a robot's autonomous ability, SLAM is one of the most important puzzels to the blueprint. Beginning with completely unknown environment, a robot can use some sensors, e.g. Lidar, Camera, etc. to measure the environment and try to move to explore its surroundings. With probability inference theory, the robot can estimate its position and orientation, and the obstacles in the environment by evaluating data from its sensors, which is exactly what SLAM does.

There are different kinds of estimators available for SLAM, such as particle filter, Kalman filter, etc. In this report, I implemented an SLAM algorithm based on particle filter. To improve the robustness of this particle-filter-based SLAM, some techniques, like using dynamic offsets when fine tuning a particle pose in the map correlation step, are applied.

With a well-perfomed SLAM, the robot is able to map its RGBD camera data to the world frame and paint the map, which is called texture mapping. In this report, an algorithm for texture mapping is also presented and tested.

II. Problem Formulation

A. Motion Model

A motion model is a function f that describes how the robot's state x_t transit to another state x_{t+1} with a given control input u_t at state x_t. If we consider the noise to the motion model, this function can also be expressed as a probability density function:

$$\mathbf{x}_{t+1} = f\left(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t\right) \sim p_f\left(\cdot|\mathbf{x}_t, \mathbf{u}_t\right) \tag{1}$$

where $\mathbf{w}_t$ is the noise of the motion model.

B. Observation Model

To quantify the sensor measurement, we need an observation model, which is a function h or equivalently a probability density function p_h if we consider the noise of observation. This model describes the observation z_t of the robot depending on x_t and m_t:

$$\mathbf{z}_t = h\left(\mathbf{x}_t, \mathbf{m}_t, \mathbf{v}_t\right) \sim p_h\left(\cdot|\mathbf{x}_t, \mathbf{m}_t\right) \tag{2}$$

where $\mathbf{v}_t$ is the noise of the observation model.

C. Markov Assumptions

Markov assumption is a method to simplify the structure of a dynamic system. The main assumptions for SLAM are:

1) The robot state $\mathbf{x}_{t+1}$ only depends on the previous input $\mathbf{u}_t$ and state $\mathbf{x}_t$
2) The environment state m_{t+1} only depends on the previous environment state $\mathbf{m}_t$
3) The environment state $\mathbf{m}_t$ and robot state $\mathbf{x}_t$ may affect each other's motion, but this may be ignored to further simplify the system presentation
4) The observation $\mathbf{z}_t$ only depends on the robot state $\mathbf{x}_t$ and the environment state $\mathbf{m}_t$

D. Bayesian Filter

With the motion model and the observation model, we can use Bayes filter, which is a probabilistic inference technique for estimating the state of dynamical systems, e.g. the robot or the environment, based on evidences from control inputs and observations using the Markov assumptions and Bayes rule:

$$p(x) = \int p(x, y)dy \tag{3}$$

$$p(x, y) = p(y|x)p(x) \tag{4}$$

$$p(x|y, z) = \frac{p(y|x, z)p(x|z)}{\int p(y, s|z)ds} = \frac{p(y|x, z)p(z|x)p(x)}{p(y|z)p(z)} \tag{5}$$

The foundation of SLAM is Bayesian filter because some famous filters, like particle filer, Kalman filter, etc., helpful in SLAM are actually special cases of the Bayes filter.

1) Joint Distribution: To combine the motion model and the observation model, Bayesian filter uses a joint distribution:

$$p\left(\mathbf{x}_{0:T}, \mathbf{z}_{0:T}, \mathbf{u}_{0:T-1}\right)$$

$$= \underbrace{p_{0|0}\left(\mathbf{x}_0\right)}_{\text{prior}} \prod_{t=0}^{T} \underbrace{p_h\left(\mathbf{z}_t|\mathbf{x}_t\right)}_{\text{observation model}} \prod_{t=1}^{T} \underbrace{p_f\left(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_{t-1}\right)}_{\text{motion model}} \quad (6)$$

2) Prediction Step and Update Step: A Bayesian filter performs mainly two steps in every inference cycle: prediction step and update step, to track $p_{t|t}(\mathbf{x}_t)$ and $p_{t+1|t}(x_{t+1})$

Prediction Step: given a prior density $p_{t|t}$ over $\mathbf{x}_t$ and the control input $\mathbf{u}_t$, uses the motion model p_f to compute the predicted density $p_{t+1|t}$ over $\mathbf{x}_{t+1}$:

$$p_{t+1|t}(\mathbf{x}) = \int p_f\left(\mathbf{x}|\mathbf{s}, \mathbf{u}_t\right) p_{t|t}(\mathbf{s}) d\mathbf{s} \quad (7)$$

Update Step: given the predicted density $p_{t+1|t}$ over $\mathbf{x}_{t+1}$ and the measurement $\mathbf{z}_{t+1}$, uses the observation model p_h to incorporate the measurement information and obtain the posterior $p_{t+1|t+1}$ over $\mathbf{x}_{t+1}$

$$p_{t+1|t+1}(\mathbf{x}) = \frac{p_h\left(\mathbf{z}_{t+1}|\mathbf{x}\right) p_{t+1|t}(\mathbf{x})}{\int p_h\left(\mathbf{z}_{t+1}|\mathbf{s}\right) p_{t+1|t}(\mathbf{s}) d\mathbf{s}} =$$

$$\frac{p_h\left(\mathbf{z}_{t+1}|\mathbf{x}_{t+1}\right)}{p\left(\mathbf{z}_{t+1}|\mathbf{z}_{0:t}, \mathbf{u}_{0:t}\right)} \int p_f\left(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t\right) p_{t|t}\left(\mathbf{x}_t\right) d\mathbf{x}_t \quad (8)$$

These two steps respectively correspond to the motion model and the observation model, which describes the dynamics of the robot and the environment.

E. Particle Filter

In practice, it is sometimes difficult to well-model the motion and the observation because of some unpredictable disturbances. Therefore, we can use sampling method to approximate these two models. The particle filter uses particles to sample the environment state space and the robot state space. To model particles, particle filter uses a mixture of delta functions:

$$\delta\left(\mathbf{x}; \boldsymbol{\mu}^{(k)}\right) := \begin{cases} 1 & \mathbf{x} = \boldsymbol{\mu}^{(k)} \\ 0 & \text{else} \end{cases} \quad \text{for } k = 1, \ldots, N \quad (9)$$

with weights $\alpha^{(k)}$ to represent $p_{t|t}$ and $p_{t+1|t}$, i.e.,
prediction step:

$$p\left(\mathbf{x}_{t+1}|\mathbf{z}_{0:t}, \mathbf{u}_{0:t}\right) = p_{t+1|t}\left(\mathbf{x}_{t+1}\right)$$

$$= \sum_{k=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(k)} \delta\left(\mathbf{x}_{t+1}; \boldsymbol{\mu}_{t+1|t}^{(k)}\right) \quad (10)$$

update step:

$$p\left(\mathbf{x}_t|\mathbf{z}_{0:t}, \mathbf{u}_{0:t-1}\right) = p_{t|t}\left(\mathbf{x}_t\right) = \sum_{k=1}^{N_{t|t}} \alpha_{t|t}^{(k)} \delta\left(\mathbf{x}_t; \boldsymbol{\mu}_{t|t}^{(k)}\right) \quad (11)$$

However, if we plugin the update step result (11) into the prediction step of (7), we get

$$p_{t+1|t}(\mathbf{x}) = \sum_{k=1}^{N_{t|t}} \alpha_{t|t}^{(k)} p_f\left(\mathbf{x}|\boldsymbol{\mu}_{t|t}^{(k)}, \mathbf{u}_t\right) \quad (12)$$

which looks different from (10). This means that (10) is an approximation to the prediction step.

By plugin (10) into (8), we can derive an approximation of the update step for particle filter:

$$p_{t+1|t+1}(x)$$

$$= \sum_{k=1}^{N_{t+1|t}} \left[\frac{\alpha_{t+1|t}^{(k)} p_h\left(\mathbf{z}_{t+1}|\boldsymbol{\mu}_{t+1|t}^{(k)}\right)}{\sum_{j=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(j)} p_h\left(\mathbf{z}_{t+1}|\boldsymbol{\mu}_{t+1|t}^{(j)}\right)} \right] \delta\left(\mathbf{x}; \boldsymbol{\mu}_{t+1|t}^{(k)}\right) \quad (13)$$

1) Particle Resampling: As the particle filter updates the weights of particles over and over again, some particles' weights get higher and higher while the others' lower and lower. The particles with very lower weights are trivial to the update process, which is similar to the depletion of particles. Therefore, we need to resample the particles if necessary.

The threshold used to resample is:

$$N_{eff} := \frac{1}{\sum_{k=1}^{N} \left(\alpha_{t|t}^{(k)}\right)^2} \quad (14)$$

There are many resampling algorithms, such as inverse transform sampling, rejection sampling, sampling importance resampling (SIR), stratified and systematic resampling (SSR), etc.

In this report, I used SSR in the implementation of particle filter SLAM.

2) Laser Correlation Model and Map Correlation: To calculate the updated weights in (13) with Lidar data, it may be difficult to model the lidar observation well. However, we can utilize the dependence of lidar data in the time sequence, i.e. use the estimated map, to correlate the data of two neighboring lidar scan to approximate the following result:

$$\alpha_{t+1|t+1} = \frac{\alpha_{t+1|t}^{(k)} p_h\left(\mathbf{z}_{t+1}|\boldsymbol{\mu}_{t+1|t}^{(k)}\right)}{\sum_{j=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(j)} p_h\left(\mathbf{z}_{t+1}|\boldsymbol{\mu}_{t+1|t}^{(j)}\right)} \quad (15)$$

by replacing with the correlation score:

$$\text{corr}(y, m) = \sum_i \mathbb{1}\{m_i = y_i\} \quad (16)$$

$$p_h(\mathbf{z}|\mathbf{x}, \mathbf{m}) = \frac{e^{\text{corr}(\mathbf{y}, \mathbf{m})}}{\sum_{\mathbf{v}} e^{\text{corr}(\mathbf{v}, \mathbf{m})}} \propto e^{\text{corr}(\mathbf{y}, \mathbf{m})} \quad (17)$$

We can see that (17) is actually a combination of the correlation score and the softmax.

And the strategy I used to determine when to resample is given by:

$$N_{eff} = \frac{1}{\sum_{k=1}^{N} \left(\alpha_{t|t}^{(k)}\right)^2} \leq N_{\text{threshold}} \tag{18}$$

F. SLAM

1) Occupancy Grid Mapping: If we model the map cells m_i as independent Bernoulli random variables

$$m_i = \begin{cases} \text{Occupied } (1) & \text{with prob. } \gamma_{i,t} = p\left(m_i = 1 | \mathbf{z}_{0:t}, \mathbf{x}_{0:t}\right) \\ \text{Free } (-1) & \text{with prob. } 1 - \gamma_{i,t} \end{cases} \tag{19}$$

with Bayes rule, we can derive the probability that a map cell is occupied as

$$\begin{aligned} \gamma_{i,t} &= p\left(m_i = 1 | \mathbf{z}_{0:t}, \mathbf{x}_{0:t}\right) && (20) \\ &= \frac{1}{\eta_t} p_h\left(\mathbf{z}_t | m_i = 1, \mathbf{x}_t\right) p\left(m_i = 1 | \mathbf{z}_{0:t-1}, \mathbf{x}_{0:t-1}\right) && (21) \\ &= \frac{1}{\eta_t} p_h\left(\mathbf{z}_t | m_i = 1, \mathbf{x}_t\right) \gamma_{i,t-1} && (22) \end{aligned}$$

$$\begin{aligned} (1 - \gamma_{i,t}) &= p\left(m_i = -1 | \mathbf{z}_{0:t}, \mathbf{x}_{0:t}\right) && (23) \\ &= \frac{1}{\eta_t} p_h\left(\mathbf{z}_t | m_i = -1, \mathbf{x}_t\right) (1 - \gamma_{i,t-1}) && (24) \end{aligned}$$

2) Steps in A Cycle: After initialization with the first lidar frame, the steps in a single cycle of the SLAM is

- 1. prediction: update particles' pose with lidar IMU data
- 2. map correlation to evaluate particles' pose
- 3. update particles' weights
- 4. choose the best particle and update the map with this particle's weight as the trust
- 5. resample if N_{eff} is lower than the threshold

G. Camera Model and Texture Mapping

With a good estimation of the robot pose and the map structure, we can projection the RGBD camera data back to the world frame to paint the map.

1) Extrinsic Matrix: The extrinsic matrix of the camera is used to transform points between the world frame and the camera optical frame:

$$\begin{pmatrix} X_o \\ Y_o \\ Z_o \\ 1 \end{pmatrix} = \underbrace{\begin{bmatrix} {}_oR_r R^\top & -{}_oR_r R^\top \mathbf{p} \\ \mathbf{0}^\top & 1 \end{bmatrix}}_{\text{Extrinsic Matrix}} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \tag{25}$$

2) Projection and Intrinsics: The projection matrix and the intrinsics matrix are used to transform points between the camera optical frame and the camera image frame:

$$\underbrace{\begin{pmatrix} u \\ v \\ 1 \end{pmatrix}}_{\text{pixels}} = \underbrace{\begin{bmatrix} fs_u & fs_\theta & c_u \\ 0 & fs_v & c_v \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Intrinsics: } K} \underbrace{\frac{1}{Z_o} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{canonical projection: } \pi} \begin{pmatrix} X_0 \\ Y_o \\ Z_o \\ 1 \end{pmatrix} \tag{26}$$

H. Rotation and Translation

Given the pose (orientation R and position $\mathbf{p}$) of $\{\mathrm{B}\}$ in frame $\{\mathrm{A}\}$, to transform a point $\mathbf{p}$ in the $\{\mathrm{B}\}$ frame to the $\{\mathrm{A}\}$ frame, we use the transformation matrix:

$${}_AT_B = \begin{bmatrix} R & \mathbf{p} \\ \mathbf{0}^\top & 1 \end{bmatrix} \tag{27}$$

III. Technical Approach

A. Data Synchronization and Joint Interpolation

Because in the dataset, joint data is more than lidar data and the lidar data is more than the RGBD camera data, we need to synchronize these data according to their timestamps. In order to make the influence of timestamp mismatching lighter, I used linear interpolation to calculate the joint data (head angle and neck angle) given a lidar data frame's timestamp t_l:

1) Find two joint data frames that $t_{j_1} < t_l < t_{j_2}$
2) Do linear interpolation with these two joint data frame for the joint result $J = [\theta_{head}, \theta_{neck}]$

B. Dynamic Offsets Scan in Map Correlation

Because the robot's motion magnitude can vary in a wide range, I used a strategy to adjust the number of offsets of x, y, and yaw to calculate the map correlation result around a particle to fine-tune the particle's pose.

C. Update Map with Trust

Since it takes time to reach the next resampling (a resampling means a high confidence in the location and the map), I multiplied the log odds with the best particles' weight as a trust about the map update.

D. Log Odds Clip and Occupied/Free Threshold

To avoid over confident in the cell occupation, I used log odds clips $\lambda_{min} = -200$ and $\lambda_{max} = 1000$, threshold of occupied cell $H_{occupied} = 2$ and $H_{free} = -15$ with a delta log odds for occupied cells $\Delta_{occupied} = 2\log(4)$ and a delta log odds for free cells $\Delta_{free} = -0.6\log 4$

E. Votes Cells With Lidar Scan Data

To better discriminate the free cells and the occupied cells, I used the number of lidar points dropped in a cell to evaluate the confidence of cell occupancy instead of simply assigning those cells containing lidar points to 1.

F. Softmax Temperature

Softmax is also useful to boost the speed of convergence. However, it makes the particle weights too sharp too quickly, which makes the particle depleting too fast. Therefore, I used a temperature $\tau = 10$ to make the softmax not too sharp:

$$\text{softmax}(\mathbf{x}/\tau) \tag{28}$$

G. Keep the Best Particle From Noise

We need to add noise to particles to do sampling. However, the best particle is the estimation with the highest confidence, which should not be destroyed by adding noise. Instead, I added noise to other particles to make them compete with the best particles. This is a very important technique to make the SLAM robust.

H. Parallel Computation and Visualization

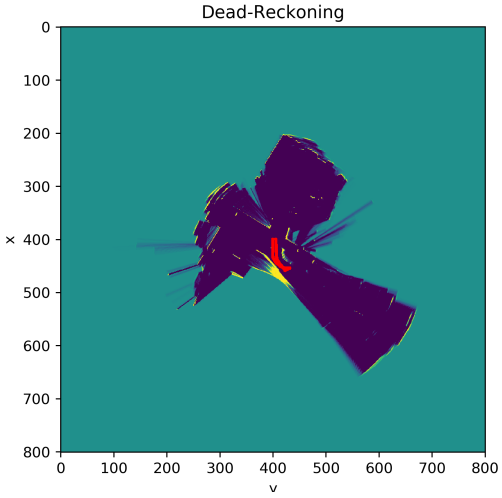
To improve the performance, I implemented a parallel computation module that computes free cell searching. I also vectorized the computation of map correlation to use the numpy auto parallel computation.

The realtime visualization of the result is also done on a separate process so that it won't influence the SLAM's speed.

IV. Results and Discussion

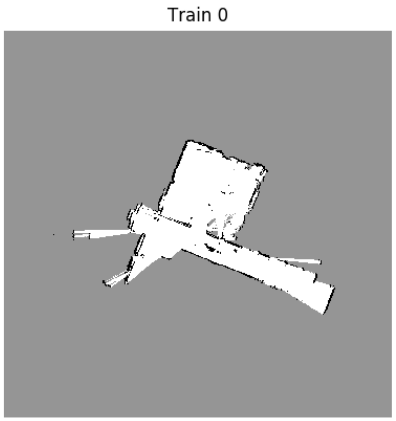
A. Dead Reckoning

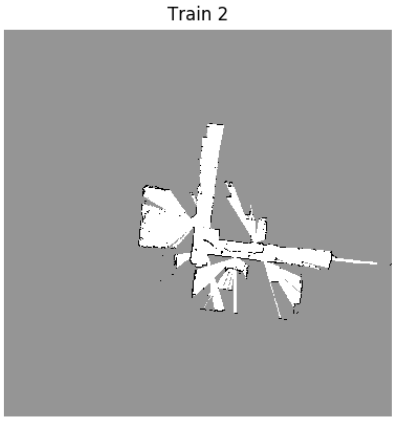
With only the prediction step, the result shows that the robot is very sensitive to turning.

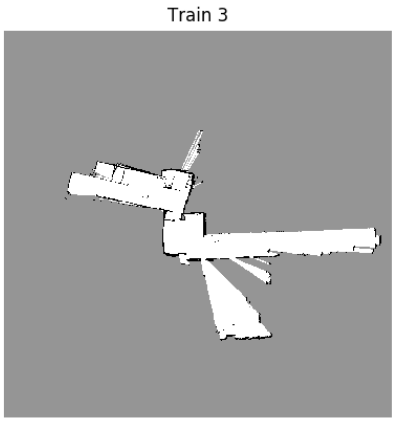


B. SLAM

Since dataset 1 and 4 are much larger, their final results are still unavailable before the deadline. But the results of dataset 0, 2 and 3 show that the SLAM I implemented is very robust even when the robot makes a 90 degree turn. A SLAM solves the localization during turning very well.







C. Texture Mapping