Particle Filter SLAM

Jiayang Zhou
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, U.S.A.
jiz141@ucsd.edu

Nikolay Atanasov
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, U.S.A.
natanasov@eng.ucsd.edu

***Abstract*—This report focuses on an approach to simultaneous localization and mapping problems (also known as SLAM). In this project, I use the obtained sensor data, odometer, 2d laser scanning, and stereo measurements from a vehicle. However, there is some noise inside those data and they are also not perfectly synchronized. For this reason, raw data needs to be modified and filtered to synchronize and filter noise. In this project, I used a particle filter to generate some particles at different locations and determine which particles are best using the map correlation method and use the best one to update the map and using the stereo images to do texture mapping.**

Keywords— Simultaneous Localization and Mapping(SLAM), Particle filter, Texture mapping, Sensor noise

I. Introduction

With the advent of the era of automation and intelligence，people are trying to make everything in their life easier and faster with the help of machines and robotics.

Robotics is a science of sensing and manipulating the physical world through computer-controlled devices. Successful robotic systems include mobile robots for planetary exploration, industrial robot arms on assembly lines, self-driving cars and robotic hands to assist surgeons. The robotic system is in the physical world and responds to information from the local environment through on-board sensors. While all these examples sound fantastic, there are still some really challenging issues that make these applications unusable right now. First, the robot environment is inherently unpredictable and dynamically changing.

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. While this initially appears to be a chicken-and-egg problem there are several algorithms known for solving it, at least approximately, in tractable time for certain environments. Popular approximate solution methods include the Particle Filter, extended Kalman Filter, Covariance Intersection, and GraphicSLAM algorithms are based on concepts in geometry and computer vision, and are used in robot navigation, robotic mapping and odometry for VR and AR.

In this article, the main approach is to use a particle filter to retrieve the path of the vehicle and the structure of the space, while also using the stereo image to draw the map texture.

II. Problem Formulation

Given a set of sensor data, including odometers, lidar scans, and stereo images, our goal is to reconstruct the map and trace the robot's trajectory on the map.

However, there is random noise in these data and those data are also not synchronized. Therefore, we need to chromize the data first and then use a particle filter to update and predict the trajectory of the vehicle to reduce the effect of the noise brought by sensors.

A. Synchronize all the data

First, we have three kinds of data from encoder of the vehicle, the lidar scanner from the vehicle, the FOG data and also the stereo image data from the camera. But all of them are not synchronized.

In order to retrieve the path and map information, I first need to synchronize all of them.

B. Transform the lidar Scan Data to the World Frame

After synchronizing all the data, we can draw the path of the vehicle (without adding noise) as follows:

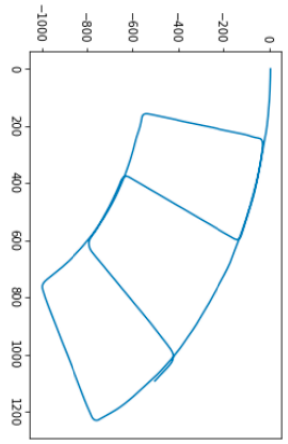


Fig. 1. Vehicle Path(Without adding noise)

In order to use the data from the lidar to do the motion and weight update, we also need to transform the lidar data from the lidar to the vehicle frame to the lidar to the world frame.

After the world frame transformation, we can use the lidar scanning data to do the following steps.

C. Using the Transformed Data to Detect the Map

Use the provided bresenham2D method with the lidar scan data to update the Probabilistic Occupancy Grid Map.

For points along the scan line, the Probabilistic Occupancy Grid Map minus log(4), for a point at the end of the scan line add log(4).

D. Update the Weight of Each Particle

For each particle, we created, compute its correlation with the former position and use the computed correlation to update the weight of the particles.

E. Update the Position of Each Particle

After updating the Probabilistic Occupancy Grid Map at a fixed position, we can construct our occupancy map for one time.

we need to update the delta_x and delta_y(position move) to the particles to inform all the particles about the new prediction, according to the particles, which had been weight updated.

F. Resample the particles when necessary

Compute the Neff factor after updating the weight of the particles. When Neff is smaller than the threshold, which means the particles are the particles is lacking sparsity.

Therefore, we need to perform a resample according to their probability to get N new particles with equal probability to maintain the function of the particle filter.

G. Creat the Textured Map

Using the RGBD images from the largest-weight particle's pose to assign colors to the occupancy grid cells.

III. Technical approach

In the Problem Formulation section, I briefly describe the basic idea of how to finish this project. In this section, I am going to make a more thorough explanation of the theory and the technical approaches.

A. Synchronize all the data

The current timestamp is 18 bytes (nanoseconds), which is not possible for all the encoders, FOG, and lidar to synchronize. Therefore, I used the former 12 bytes as the timestamp and stored them in the dictionary data structure in python, which enables me to directly access the data at a specific timestamp.

After the synchronization, there is 11 thousand data record in total, which is enough for me to do the later steps.

B. Transform the lidar Scan Data to the World Frame

- First, the lidar data is gathered through scan from -5 degree to 185 degree at a resolution of 0.666 degree. The lidar detect the distance from the lidar to the detected spot. In order to make the data more reliable, I only use data from 0.1 to 40. The use the following equation to compute the position in the world to lidar frame.

$$x_i = l_i * \cos\ \alpha_i$$
$$y_i = l_i * \sin\ \alpha_i$$

- Second, compute the rotation matrix and shift from lidar to vehicle and vehicle to world frame to get the world to world frame lidar scan data. The computation of the rotation matrix goes as follows:

$$R = R_z(\psi)R_y(\theta)R_x(\phi)$$
$$= \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

- Third, use the drive motion model to compute the moving distance of the vehicle. The total moving distance is the mathematical mean of the left wheel moving distance and the right wheel moving distance. The distance computation equation is as follows:

$$\text{meters/pertick} = \frac{\pi \times (\text{ wheel diameter })}{\text{ticks/perrevolution}}$$

C. Using the Transformed Data to Detect the Map

After transferring all the data to the world frame, I can using the provided bresenham2D algorithm to update the probabilistic occupancy map with the starting and ending position of the lidar scan ray.

For each observed cell, decrease the log-odds if it was observed free or increase the log-odds if the cell was observed occupied:

$$\lambda_{i,t+1} = \lambda_{i,t} + \log\ g_h(z_{t+1} \mid m_i, x_{t+1})$$

After having the probabilistic occupancy map, we can also update the occupancy map using the following equation:

$$m = \begin{cases} \text{Occupied(1)} & \text{with prob. } \gamma_{i,t} = p(m_i = 1 \mid z_{0:t}, x_{0:t}) \\ \text{Free(0)} & \text{with prob.1- } \gamma_{i,t} \end{cases}$$

D. Using Particle Filter to Update Weights of Particles

Particle Filter uses a certain number of particles to approximate the probability distribution and update the motion and weight of each particle after each observation.

To be specific, the particle filter uses bunches of delta functions and weights alpha to represent the filter as follows:

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

$$p(\mathbf{x}_{t+1} \mid \mathbf{z}_{0:t}, \mathbf{u}_{0:t}) = p_{t+1|t}(\mathbf{x}_{t+1}) = \sum^{N_{t+1|t}} \alpha_{t+1|t}^{(k)} \delta\left(\mathbf{x}_{t+1}; \boldsymbol{\mu}_{t+1|t}^{(k)}\right)$$

The weight of each particle is initialized to be 1/N.

In order to update the weight of each particle, I used the MapCorralation function in the utils using Laser Correlation

Model to compute the likelihood of a laser scan z proportional to the correlation between the scan's world-frame projection y = r (z, x) via the robot pose x and the occupancy grid m.

The mathematic function goes as follows:

$$p_{t+1|t+1}(\mathbf{x}) = \frac{p_h(\mathbf{z}_{t+1} \mid \mathbf{x}) \sum_{k=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(k)} \delta(\mathbf{x}; \boldsymbol{\mu}_{t+1|t}^{(k)})}{\int p_h(\mathbf{z}_{t+1} \mid \mathbf{s}) \sum_{j=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(j)} \delta(\mathbf{s}; \boldsymbol{\mu}_{t+1|t}^{(j)}) d\mathbf{s}}$$

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

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

$$e^{\text{corr}(\mathbf{z},\mathbf{m})} = \text{MapCorralation}(\, \text{z}, \text{m}\,)$$

The MapCorralation is computed as follows:

$$\text{MapCorralation}(\, \text{z}, \text{m}\,) = \sum_i \mathbb{1}\{Z_i = m_i\}$$

E. Update the Position of Each Particle

The position update is relatively each, just need to update the delta_x, dealt_y and delta_yaw to all the particles, while we also need to add some Gaussian Noise to three dimensions to make particle filter work to improve the accuracy.

F. Resample the particles when necessary

It is also necessary to compute Neff after the weight update of particles and if it is smaller than N, resample all the particles according to their existing probability.

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

In this project, I set it to be 1.6 and during the experiment, it has been proved to be working.

G. Color The Map

For this part, I use the constructed map, RGB, and depth cameras to color the map.

First, compute the disparity between the left and the right image and then use the disparity, focal length, and the baseline to compute the depth of each pixel in the image according to the following equation:

$$d = u_L - u_R$$
$$z = \frac{f * b}{d}$$

Second, after getting the depth information, I can get the position of these points in the space with the following equation:

$$[u_L, v_L, d]^T = \begin{bmatrix} fs_u & s_\theta & c_u & 0 \\ 0 & fs_v & c_v & 0 \\ 0 & 0 & 0 & fs_u b \end{bmatrix} \frac{1}{Z_o} [X_o Y_o Z_o 1]^T$$

After that we get the camera to vehicle frame data and use the same way mentioned before to transfer the position into the camera to world frame.

Finally, project the 3d position onto the 2d floor and color the position with RGB value to create the texture map as follows:

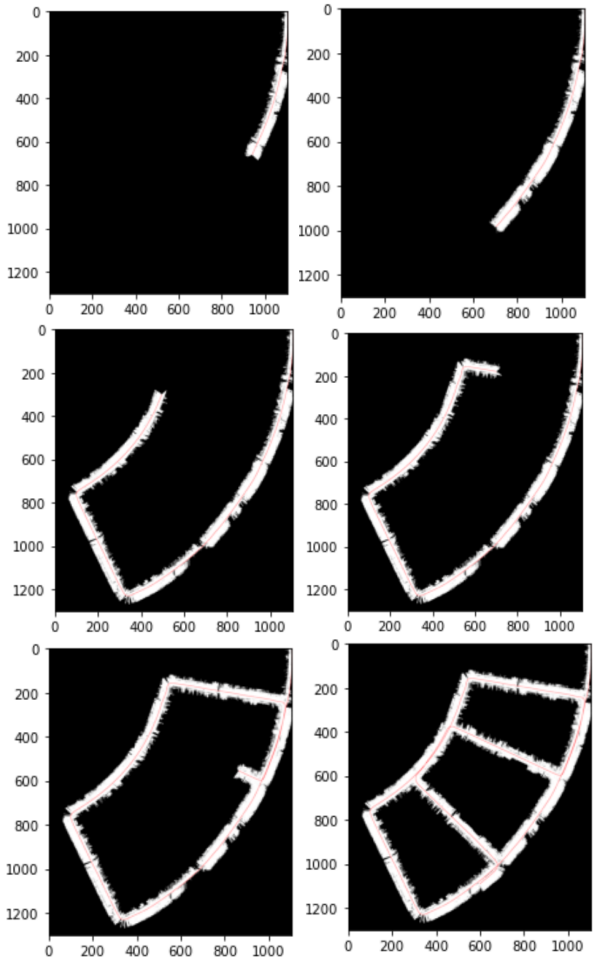
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z} \begin{bmatrix} f_x & 0 & s_x \\ 0 & f_y & s_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

IV. Results

After implementing all the models above, the final results go as follows.

A. Dead-reckoning

When there is no noise and only one particle, the trajectory and occupancy grid map over time goes as follows:



B. Particel Filter

When having noise with 10 particle, the trajectory and occupancy grid map over time goes as follows:

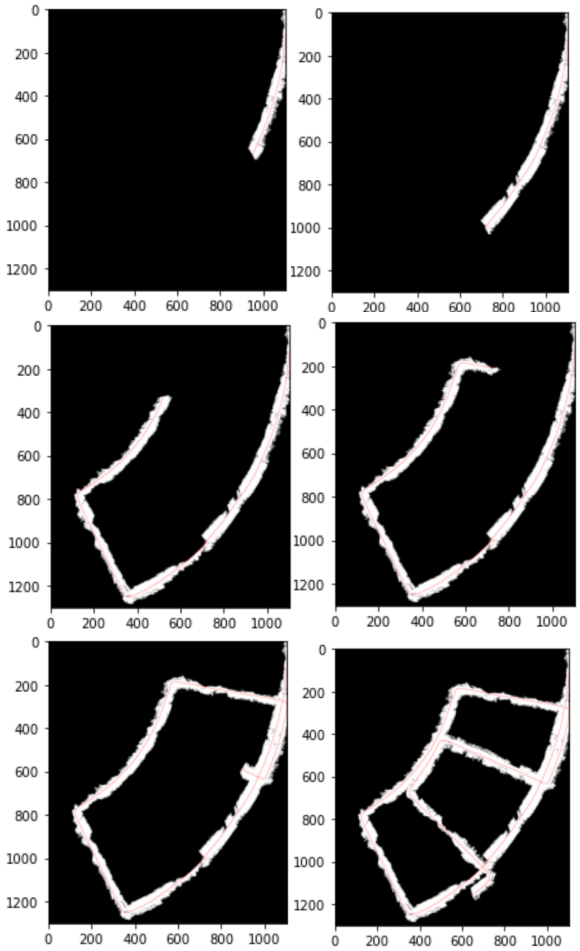


Fig. 2. Images of the trajectory and occupancy grid map over time

From the result above, we could see that, after adding noise to the sensor data and using 10 particles to filter the map, the reconstructed map and trajectory have a better result and more clear boundary.

But there is also some small shift of the path of the vehicle brought by the noise.

REFERENCES

[1] Ng, A.Y. & Jordan, M. I. (2002). On Discriminative vs. Generative Classifiers: A comparison of Logistic Regression and Naive Bayes, *Neural Information Processing Systems*, Ng, A.Y., and Jordan, M. (2002).

[2] SebastianThrun,WolframBurgardandDieterFox,Probabilisticrobotic- s, MIT Press.