Particle Filter SLAM

Benjamin Chang
Department of Electrical and Computer Engineering
University of California, San Diego
bmc011@ucsd.edu

***Abstract*—In recent years, large efforts have been made toward enabling autonomous vehicles to better understand their surroundings. This usually involves a variety of sensors such as lidar, stereo cameras, and radar. One such common method which uses lidar is SLAM (Simultaneous Localization and Mapping). This project aims to use data captured from a variety of sensors attached to a vehicle to perform SLAM, thus localizing the vehicle and mapping its surrounding environment.**

I. Introduction

The ability for autonomous vehicles to locate themselves in their environment is extremely important for safe navigation. Mapping a vehicles surrounding environment is also highly valuable as that enables it to better understand of its environment and to traverse more safely. Environment mapping also has applications not related to autonomous vehicles as seen in the Google Earth project. Without a way to do environment mapping, the task of mapping large areas would otherwise be tedious and require cars with human drivers.

In this project, several steps are implemented to localize a vehicle and map its surroundings using four types of captured sensor data: 2D lidar, FOG, encoder, and stereo RGB. These enable the vehicle to understand the distance of objects in its surrounding, its change in angle, the distance which it travels, and the color of objects in its surroundings.

The project is split into four sections. The first is ”Particle Filter Prediction” which predicts where the vehicle will be at a future time. The second is ”Particle Filter Update” which finds where the corresponding lidar data should plotted on a map. The third is ”Map_Update” which plots the vehicle trajectory and lidar readings onto the map. The fourth is ”Texture_Mapping” which adds color to the lidar readings on the map.

II. Problem Formulation

We set the initial position of the vehicle at $[x, y, theta] = [0, 0, 0]$ where x and y are the axes of the vehicle in the 2D world frame and theta is the angle of the robot relative to the world frame x-axis. An occupancy-grid map of log-odds $\lambda_{i,t}$ is initialized to record the vehicle's motion and environment. Given a lidar scan,

new lidar scan z_{t+1}, the scan is transformed into the world frame using the robot pose x_{t+1}. From this, we determine the cells that the lidar beams pass through using Bresenham's line rasterization algorithm. For each observed cell i, the map log-odds are updated as below.

$$\Delta\lambda_{i,t} = \log \frac{p(m_i = 1 \mid \mathbf{z}_t, \mathbf{x}_t)}{p(m_i = -1 \mid \mathbf{z}_t, \mathbf{x}_t)} = \begin{cases} +\log 4 & \text{if } \mathbf{z}_t \text{ indicates } m_i \text{ is occupied} \\ -\log 4 & \text{if } \mathbf{z}_t \text{ indicates } m_i \text{ is free} \end{cases}$$

We use the constraint $\lambda_{MIN} \leq \lambda_{i,t} \leq \lambda_{MAX}$ to avoid over-confident estimation and the below logistic sigmoid function to recover the map pmf.

$$\gamma_{i,t} = p(m_i = 1 \mid \mathbf{z}_{0:t}, \mathbf{x}_{0:t}) = \sigma(\lambda_{i,t}) = \frac{\exp(\lambda_{i,t})}{1 + \exp(\lambda_{i,t})}$$

We use a particle filter to maintain the pdf of the robot state over time. Each particle $\mu^{(k)}_{(t|t)}$ is a hypothesis on the state x_t with confidence $\alpha^{(k)}_{(t|t)}$. The particles specify the pdf of the vehicle state at time t as below.

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

To predict the pdf of each particle's predicted vehicle state at a new timestamp, the following equations are used where $\mu^{(k)}_{(t+1|t)}$ is the hypothesis of the next vehicle state and $\alpha^{k}_{t|t}$ is the weight given for how likely a particle is to the vehicle's actual position at time $t + 1$.

$$\boldsymbol{\mu}^{(k)}_{t+1|t} = f\left(\boldsymbol{\mu}^{(k)}_{t|t}, \mathbf{u}_t + \boldsymbol{\epsilon}_t\right) \qquad \alpha^{(k)}_{t+1|t} = \alpha^{(k)}_{t|t}$$

More specifically, to predict the next vehicle pose, we assume a differential-drive model for our vehicle and predict a future vehicle pose at time $t + 1$ as below.

$$\mathbf{x}_{t+1} = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \mathbf{x}_t + \begin{bmatrix} v_t \cos(\theta_t) \\ v_t \sin(\theta_t) \\ \omega_t \end{bmatrix}$$

In the above equation, v_τ is the distance travelled by the vehicle in time τ and is calculated by $v_\tau = (\pi * encoder\_left\_wheel\_diameter * (encoder\_left\_count\_in_use[t] - encoder\_left\_count[t-1]))encoder\_resolution$ where t is a variable which iterates through FOG timestamps. For $t = 0$, the subtraction part of the v_tau equation is left out. Though the equation uses

ω above, we put use it to represent $\Delta\theta$ which is the angle change of the vehicle. We add to each predicted pose element Gaussian noise with mean 0 and standard deviation equivalent to the maximum change each respective pose element.

We use the predicted particle poses to update the weight (likelihood) of each particle using the equations below.

$$\boldsymbol{\mu}^{(k)}_{t+1|t+1} = \boldsymbol{\mu}^{(k)}_{t+1|t} \qquad \alpha^{(k)}_{t+1|t+1} \propto p_h(\mathbf{z}_{t+1} \mid \boldsymbol{\mu}^{(k)}_{t+1|t}, \mathbf{m})\alpha^{(k)}_{t+1|t}$$

$p_h(z_{t+1}|\mu_{t+1}|t^k, m)$ is calculated using the laser correlation model given below.

$$p_h(\mathbf{z}_{t+1} \mid \boldsymbol{\mu}^{(k)}_{t+1|t}, \mathbf{m}) \propto \exp\left(\text{corr}\left(\mathbf{y}^{(k)}_{t+1}, \mathbf{m}\right)\right) \qquad \text{corr}(\mathbf{y}, \mathbf{m}) = \sum_i \mathbb{1}\{y_i = m_i\}$$

The particle with the highest value weight is updated as the current vehicle pose x_t and the particle filter prediction/update steps iterated till the map is complete. If the number of particles gets too low, particles are resampled and the existing particle weights redistributed uniformly.

To update the particle pose/weights and the occupancy-grid map, transforming the lidar scan coordinates from lidar frame to vehicle frame and then frame vehicle frame to world frame is required. To obtain the lidar to vehicle frame transform, we use the following equation where R is a $3x3$ rotation matrix and p is a $3x1$ translation matrix.

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

To apply a translation matrix to a set of coordinates, we create a $4xN$ matrix out of particle coordinates where N is the total number of particles. To go from lidar to world coordinates, we use the following equation.

$$_W T_L = {_W T_V}\ {_V T_L}$$

In the map update step, converting from world frame to map frame is also necessary to plot world frame lidar coordinates on the occupancy-grid map. This involves taking a set of x,y lidar coordinates, subtracting from its respective minimum map index, dividing the result by the map resolution, and then subtracting the result by 1.

Adding color to the map lidar scans, also known as texture mapping, involves using stereo RGB images to find disparity between the images using the equation below where d is disparity, z is depth, f is focal length, and b is baseline.

$$d = u_L - u_R = \frac{1}{z} f s_u b$$

Depth can be computed by taking the inverse of disparity. Following this, pixels of certain colors can then be mapped to and drawn on map lidar scan areas depending on which pixel depths they correspond to at a given FOG timestamp. The stereo to vehicle matrix can be computed similarly to the other transform matrices mentioned above. It simply requires a rotation matrix R and translation vector t.

III. Technical Approach

We first import all of our data which include encoder, FOG, and lidar data as well as parameters such as encoder resolution, encoder wheel diameters, lidar to vehicle frame rotation matrices, stereo camera to vehicle frame rotation matrices, etc. to use in our calculations.

The encoder data includes counts of the number of ticks of the vehicle's left and right wheels. The FOG data includes the vehicle's roll, pitch, and yaw of which we only use yaw. The lidar data includes distance values of laser scans at 286 different angles. Each data file also includes timestamps for when each row of data was recorded. The encoder data has 116,048 timestamps data, the lidar data 115,865 timestamps, and the FOG data 1,160,508 timestamps. The data posed the problem that the timestamps of the 3 data types need to be synced to be useful. This is done by creating encoder and lidar variables with the same number of rows as the FOG data. For row i of the FOG data, the corresponding row i of the new encoder/lidar variables is filled with encoder/lidar data collected at the timestamp closest to the the timestamp of the FOG data at row i.

With the data synced, we initialize a map of size $[-100, 1300]x[-1200, 1300]$ which is used to represent the vehicle's environment. This map size is chosen to fit the complete trajectory of our vehicle.

Next, we perform the particle filter prediction and update steps which are defined in the "Problem Formulation" section. To run the particle filter, we iterate through the total number of FOG timestamps, predicting a new vehicle pose every timestamp. Every 10 timestamps, we update the particle weights. This is done by using map correlation which determines how closely a new lidar scan end points allign with previous scans. Every 10 timestamps, we also update the occupancy-grid map with new lidar scans with the pose of the highest weight particle set as the vehicle pose. When each lidar scan is plotted on the map, Bresenham's 2D rasterization algorithm is used to calculate which cells are passed through by the lidar scan beams so to those cells can be given higher weights as well and plotted. This results in a mapping of the surrounding vehicle environment in the occupancy-grid map. Each update step, the highest weight particle vehicle pose is stored as part of the vehicle's past trajectory. If the number of particles becomes less than 5, we resample the particles, redistributing the particle weights uniformly. Every 500 iterations, we also plot the occupancy-grid map cells which lie within the map boundaries as well as the vehicle trajectory. Cells with pmf $\geq$ 0.9 are plotted with a higher value. The particle filter prediction and update steps are run until all the Fog timestamps have been iterated through. The specific equations used to calculate all the above mentioned steps are discussed in detail in the ”Problem Formulation” Section.

We computed and saved disparity images from the stereo RGB images. These can be converted to depth values by taking the inverse of the disparity values. We then transform the depth values from the stereo camera frame to the vehicle frame and then to the world frame using tranformation matrices. The pixels with given depth that corresponds to the location of a lidar scan in the world frame are then added to color the lidar scans on the occupancy-grid map.

IV. Results

We start by plotting the first lidar scan in its own frame and in the occupancy-grid map to test for correctness. The results are shown below.

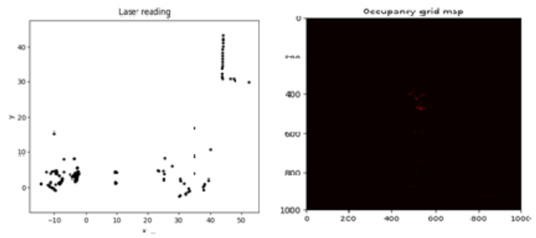

Fig. 1. First lidar Scan in lidar frame and map frame

Running the particle filter on our encoder, lidar, and Fog data produces the following occupancy-grid map. We test our particle filter using 10 particles. The map at various iterations are shown in the figures below.

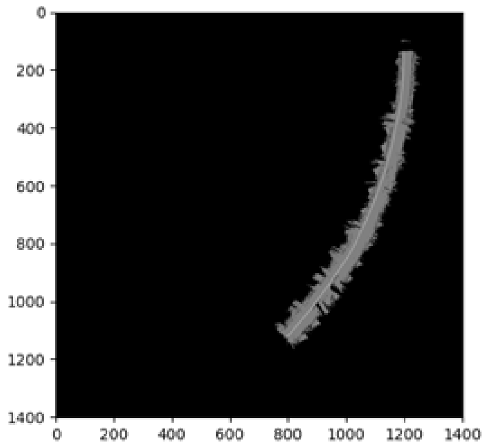

Fig. 2. 315,000th FOG timestamp

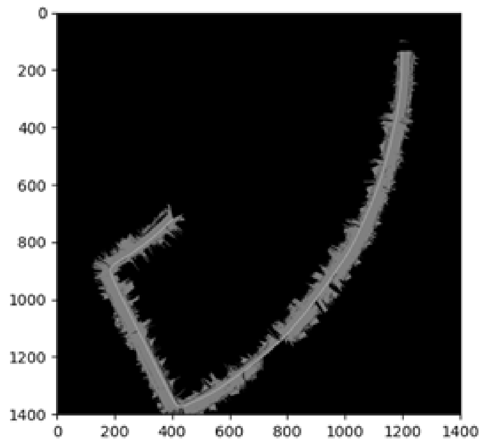

Fig. 3. 475,000th FOG timestamp

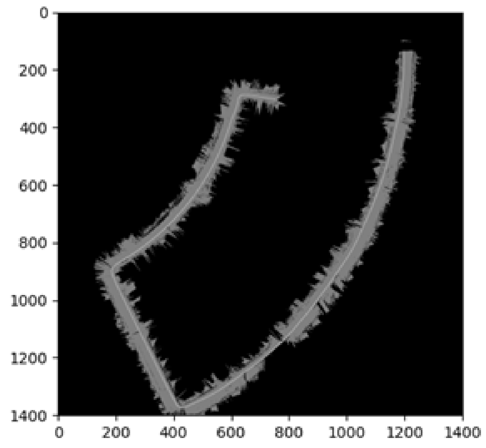

Fig. 4. 579,000th FOG timestamp

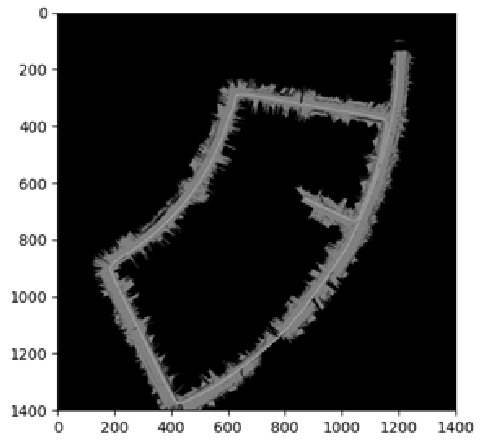

Fig. 5. 719,500th FOG timestamp

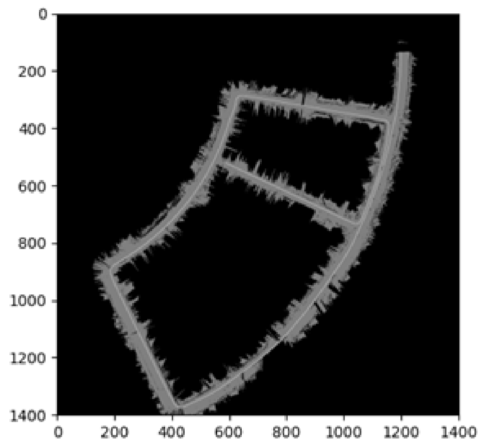

Fig. 6. 812,500th FOG timestamp

The vehicle trajectory in the result looks smooth and the plotted lidar scans seem reasonable. However, it is to be noted that there is a good amount of noise on the edges of the lidar data that the vehicle maps out. This makes sense since looking at the stereo images, the boundaries of the areas the vehicle is mapping aren't flat walls, but outdoor areas which have a large variety of distances and occluding objects such as trees, poles, and pedestrians. This could also come from reflections in the vehicle's environment or sensor noise.

The plotted vehicle trajectory makes sense from taking a look at the stereo camera RGB data.

V. Discussion

From the above results, the particle filter seems to do a good job of plotting the vehicle trajectory smoothly, but runs much slower than would be needed for real-time vehicle localization and environment mapping.

Problems that were run into while doing this project included timesyncing, frame transformations, understanding how to use the encoder data, figuring out what map size to use, etc.

VI. Conclusion

This paper explores the Particle Filter localization and Occupancy-Grid mapping methods, providing results which show they achieve good performance. There are many other more complex methods to do vehicle localization, but the simple effectiveness of Particle Filter is certainly something to be behold. In the future, the particle filter can be timesynced to the lidar data's timstamps instead do decrease run-time and texture mapping can be added to increase the completeness of the mapping results.