Particle Filter SLAM

Zhexu Li
Department of Electrical Computer Engineering
University of California, San Diego
San Diego, California, United States
zhl411@ucsd.edu

***Abstract*—In this project I Implement simultaneous localization and mapping (SLAM) using odometry, 2-D LiDAR scans, and stereo camera measurements from an autonomous car. Use the odometry and LiDAR measurementsto localize the robot and build a 2-D occupancy grid map of the environment.**

***Index Terms*—Particle filter, SLAM, Map Texturing.**

I. Introduction

Simultaneous Localization and Mapping (SLAM) is one of the fundamental problems in robotic navigation. An accurate, fast, and reliable SLAM algorithm has great potential in many industries including robotics, security, and transportation. Despite the current research focus has shifted to deep learning based approaches because of the rapid advance in neural network architectures, traditional particle filter based SLAM algorithms are still valuable because they require less computational resources than deep learning models to train, not to mention understanding the intuitions behind these models will provide a solid foundation for learning more advanced models. In this project, I implement a particle filter with a differential-drive motion model and scan-grid correlation observation model for simultaneous localization and occupancy-grid mapping. Then I implement a map texturing algorithm to color the cells in the occupancy grid with RGB values according to the projected points that belong to the its plane. I deploy this algorithm on a autonomous vehicle dataset to create occupancy grid mapping and textured map. The resulting algorithm could potentially be used in many scenarios including robotic navigation, and autonomous mapping.

II. Problem Formulation

A. Mapping

Given the robot trajectory $x_{0:t}$ and a sequence of observations $z_{0:t}$, and the unknown occupancy grid m, the Occupancy Grid Mapping problem can be described as:

$$p(m|z_{0:t}, x_{0:t}) = \prod_{i=1}^{n} p(m_i|z_{0:t}, x_{0:t}) \tag{1}$$

where m_i = 1 or -1, with p(m_i = 1) = $p(m_i = 1|z_{0:t}, x_{0:t})$.

Given a matrix of pixels $X \in \Re^{n\times3}$, we are interested in classifying every pixel $x \in \Re^{1\times3}$ into a color category $y_i \in \{1, 2, 3\}$, where 1 = red, 2 = green, 3 = blue, and return a vector of predicted color labels y $\in \Re^{n\times1}$.

B. Particle Filter

A particle is a hypothesis that the value of x is μ with probability a. Given a map m, a sequence of control inputs $u_{0:T-1}$, and a sequence of measurements $z_{0:T}$, the localization can be described as infering the robot state trajectory $x_{0:T}$, and the particle filter estimates

$$p(x_t|z_{0:t}, u_{0:t}, m) \tag{2}$$

of the robot state over a period of times.

C. Prediction

The purpose of prediction step is to obtain the prediction pdf

$$p_{t+1|t}(x_{t+1}) \tag{3}$$

using the motion model p_f, and here I use the differential drive motion model.

D. Update

The purpose of the update step is to obtain the updated pdf

$$p_{t+1|t+1}(x_{t+1}) \tag{4}$$

using the observation model p_h, and here I use the Laser Correlation model.

E. Resampling

Resampling is required in case of Particle depletion, which is a situation in which most of the updated particle weights become close to zero because the finite number of particles is not enough, which means the

$$p_h(z_{t+1}|u^k_{t+1|t}) \tag{5}$$

are too small at all k = 1, ..., N.

F. Map Texturing

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

III. Technical Approach

I first implemented the Mapping algorithm based on the formulas mentioned above. The implemented algorithm, mapping(wx, wy, theta, T, par, lir, ang) intakes the vehicle position (wx = x, wy = y, theta = theta), transformation matrix T (from lidar to body frame), map parameters par, and valid Lidar scans points lir and angle, it returns the updated map parameters. The algorithm basically transforms the input Lidar points from the Lidar frame to the world frame, then use the bresenham2D algorithm to obtain the occupied cells and free cells that correspond to the lidar scan, and updates the map log-odds according to these observations. Notice The input lidar scans should be in desired range, I chose to filter lidar points outside of the function because it's easier to adjust.

Then, I implemented the particle filter prediction step. The prediction(state, ti, vt, wt) intakes particle state (x, y, theta) at time t, time interval ti, linear velocity vt and change of yaw wt, it returns the new particle state (x, y, theta) at time t + 1. Basically it uses the encoders and the FOG data to estimate the robot trajectory via the differential drive motion models f(x, y) mentioned in lecture 10 slide 20. The noise I use for prediction step are: mean 0 and variance 0.0005 for theta, mean 0 and variance 0.005 for x and y, because based on my experiment these noise performed the best.

Then I implemented the particle filter updates step. The update(state, weights, mp, lir, ang) intakes particle state (x, y, theta) at time t + 1, particle weights at time t, map parameters mp at time t, and new lidar points and angles lir, ang, it returns updated particle weights at time t + 1 based on the lidar input. The algorithm utilizes Laser correlation model (mentioned above and in lecture 10 slide 20, 21, 22, 23) to correct the robot pose. The particle poses remain unchanged but the weights are scaled by the model. The dimension of grid used for scan-grid corrlation is 9 x 9.

Once I have the updates step working, I implemented a resampling step to prevent Particle Depletion problem mentioned above. The resampling(particles, weights) intakes particle state and particle weights, and returns resampled particle state and weights. I used the Stratified resampling discussed in lecture 9 slide 34.

The map texturing algorithm can be approached by first, extract the pose of the particle with the highest weight, then calculate disparity and estimate depth using stereo images, and project the RGB colors using the the left camera onto the occupancy grid in order to color it.

There are some helper functions (plotting, etc) defined in the helper function section in the Combined SLAM notebook, see their discrptions for more information. All algorithms are well documented in the notebook.

Once I had all these algorithms defined, I deployed it on the autnomous vehicle dataset which contains 115865 Lidar scans and 116048 encoder readings, I used 20 particles, resolution = 2, map dimension x = (-100, 1700), y = (-1700, 100), which gives 900 x 900 grids map, max Lidar range = 80 based on the Lidar specifications, and minimum Lidar range = 1. I ran the algorithm until all the Lidar readings were used, and stopped there because there is no point to continut mapping given there is no more Lidar scans. The results will be discussed in the results section.

IV. Results

A. First Lidar Scan

The map generated by the first Lidar scan is displayed in figure 1.

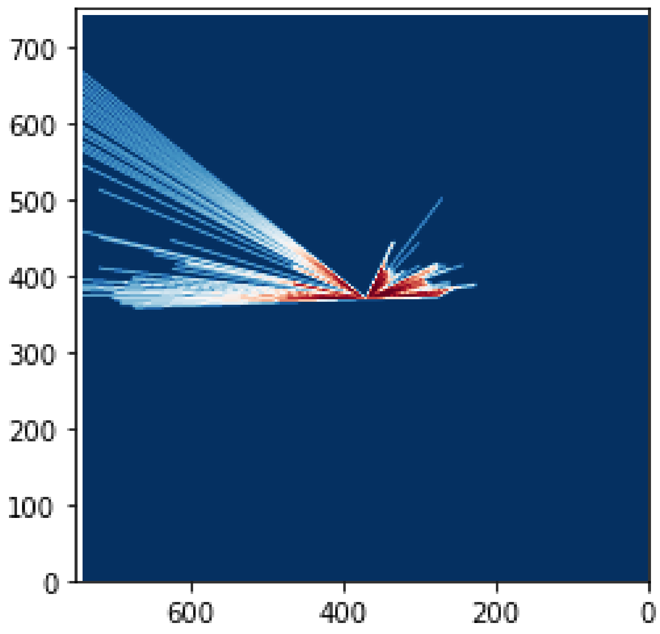

Fig. 1. The resulting map of the first lidar scans.

Notice the forward trajectory of the car is upwards in the map. Valid Lidar range was set to 1 to 80 meters. The result is hard to interpret but the lidar ray seems reasonable.

B. Prediction

The prediction result using all encoder and fog readings, with one particle and no noise, is shown in figure 2.

The result makes sense and the trajectory basically matches the movement of the autonomous car shown by stereo images. The car kept going right before making several sharp right turns and made several other turns.

C. Particle Filter SLAM

Using the parameters mentioned above, the resulting map are shown in figure 3 and 4. Notice I stopped mapping when the Lidar scans ran out, because there is no point to continue mapping without Lidar readings. Again, the results make sense and match the movement of the car shown in the stereo images. The car first kept going right for a while, and made a sharp right turn, and went straight for a while before turing right again, and it kept going left for a while, and made a sharp right turn again. I only stored the initial map and the last map, because the intermediate plots were overwritten by

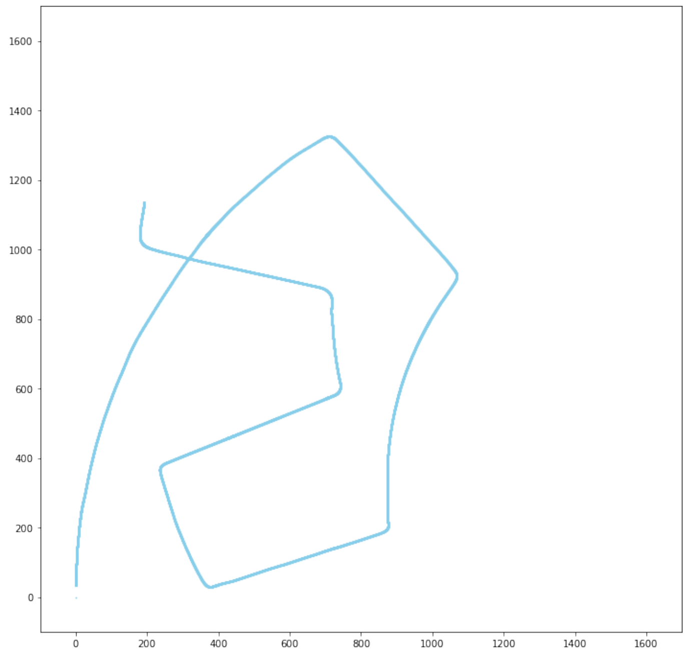

Fig. 2. Prediction result with one particle and no noise.

other plots in the jupyter notebook. But I did noticed the car was stationary for about 5 minutes during encoder index 10000 to 31000, which might because of a long red light in a busy intersection. Overall, the mapping results were quite impressive and captured the movement of the car pretty well.

I also tried texturing the map and spent a lot of time trying to make it work, but it was not working and the process was too slow. I described the texture mapping problem and my approach in some details above but the result was not ideal.

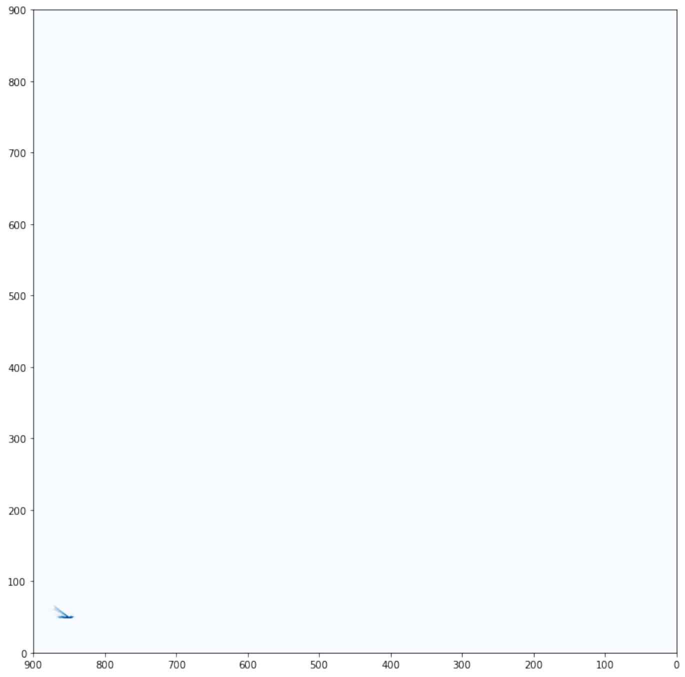

Fig. 3. The initial mapping, the car is on the bottom left of the plots.

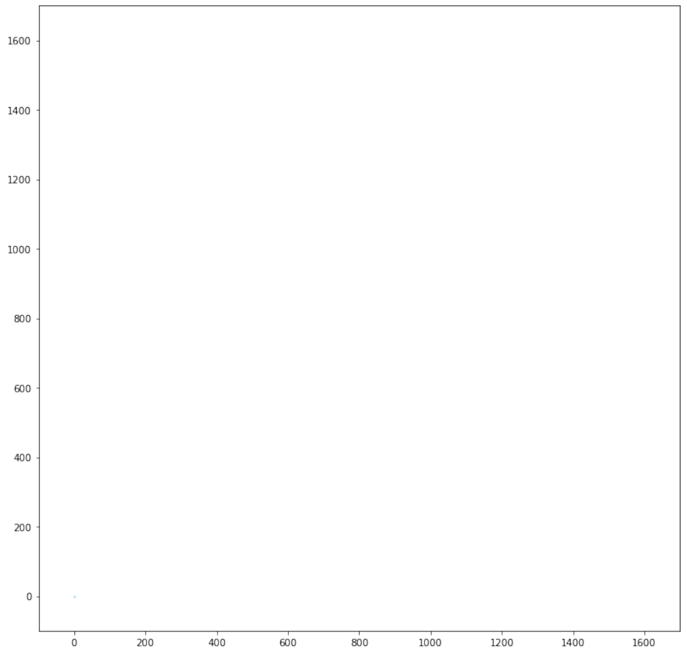

Fig. 4. The initial trajectory, the car is on the bottom left of the plot, which is a blue dot that's hard to see

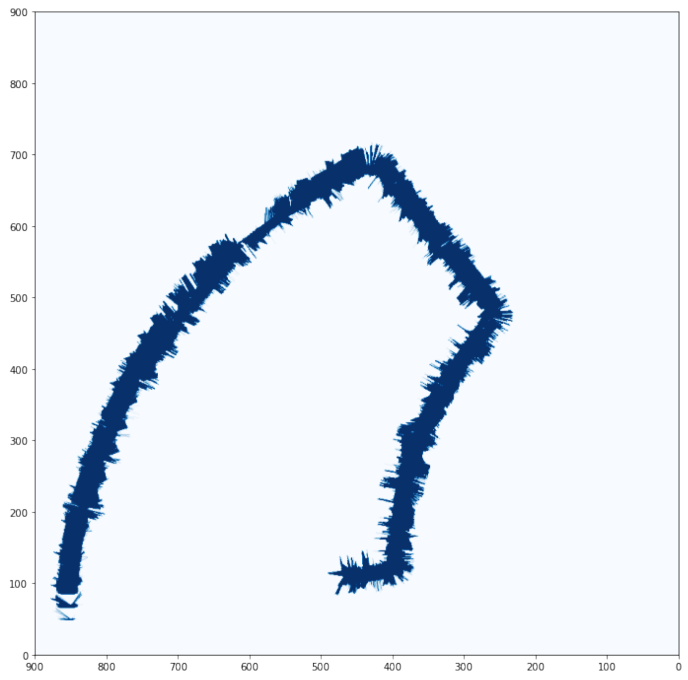

Fig. 5. The resulting map at the last Lidar reading.

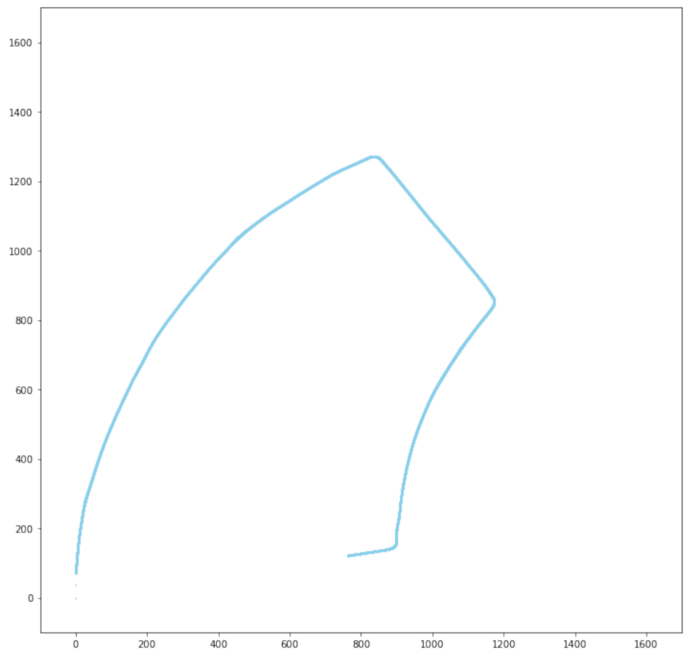

Fig. 6. The resulting trajectory at the last Lidar reading, the later trajectory are not plotted.