PARTICLE FILTER SLAM

Joseph Chang

Department of Electrical and Computer Engineering, University of California, San Diego

ABSTRACT

There are many scenarios in the real-world that require calculating the position of an object relative to its environment. This is used in cleaning robots autonomous vehicles. It's also crucial for the robot to build a map of the environment so it can avoid crashing into objects or walls it has already detected. This is solved by simultaneous localization and mapping aided by many sensor data collected from a moving vehicle.

Index Terms— computer vision, SLAM, particle filter

1. INTRODUCTION

Simultaneous Localization and Mapping (SLAM) is an important topic in computer vision. It is used to gain a 2D understanding of the world from 2D lidar scans, wheel enoder counts, and fiber optic gyro (FOG) angle data. The algorithm consists of two primary steps in mapping and localization. For mapping, we build a map of the environment given a robot's state trajectory and sensor measurements. In localization, we estimate the robot's trajectory given a map of the environment and so the cycle continues. There are other popular sparse SLAM algorithms including the Kalman Filter and Factor Graphs SLAM. However, we choose to use the particle filter method which uses particles and Gaussian distributions for landmark positions. We perform mapping with a log-odds occupancy grid map which is updated when a new lidar scan arrives. Localization contains a prediction step to obtain a predicted pmf and update step to obtain the updated pmf. In this paper, we implement particle filter SLAM to create a map of a vehicle's environment as it drives and track it's position.

2. PROBLEM FORMULATION

Our problem is to localize the car and build an occupancy map as well as a texture map.

Dataset: We are given encoder data from a car's left and right wheel. The resolution is 2096 and the wheels have diameters of 0.62m. The FOG data contains the change in roll, pitch, and yaw of the vehicle which is angle data. Lidar scans are given with a start angle of -5°, end angle of 185°, angular resolution of 0.666, and maximum range of 80m. We are also given stereo images for texture mapping. As data from these sensors start at different times and vary in frame-rate, we synchronize FOG to encoder timestamps so there is one FOG corresponding to each encoder reading.

Occupancy Mapping: Occupancy grid mapping is used to represent the environment by dividing it into cells. It is used to keep track of the state of each map cell as occupied or free denoted as 1 and -1 respectively. Each cell value is a pmf conditioned on the robot's state $x_{0:t}$ and sensor observations $z_{0:t}$. The cell values are assumed independent so each cell can be tracked individually.

$$p(\mathbf{m} \mid \mathbf{z}_{0:t}, \mathbf{x}_{0:t}) = \prod_{i=1}^{n} p(m_i \mid \mathbf{z}_{0:t}, \mathbf{x}_{0:t})$$

Localization: Given a grid map m, sequence of control inputs $u_{0:t-1}$, and sequence of sensor measurements $z_{0:t}$, the robot state trajectory $x_{0:t}$ can be inferred. We maintain a pmf of the robot state over time. A prediction step is performed using the motion model p_f to obtain the predicted pmf.

$$p_{t+1|t}(\mathbf{x}_{t+1})$$

An update step is performed using the observation model p_h to obtain the updated pmf. The particle poses remain unchanged, but the weights are scaled by the observation model.

$$p_{t+1|t+1}(\mathbf{x}_{t+1})$$

Motion Model: In the prediction step, the differential-drive motion model p_f is used to compute the vehicle's trajectory. For a given linear velocity v_t and angular velocity w_t, the vehicle's location in the world can be updated. Adding small 2D Gaussian noise $_t$ to the vehicle state can increase the accuracy of prediction. u_t is a vector containing linear and angular velocity.

$$\mu^{(k)}_{t+1|t} = f(\mu^{(k)}_{t|t}, \mu_t + \epsilon_t)$$

Texture Mapping: Given stereo images and disparity, the RGB values can be projected onto an occupancy grid map as 3D vector.

3. TECHNICAL APPROACH

The goal of this project is to use a particle filter with a differential-drive motion model and scan-grid correlation observation model for simultaneous localization and occupancy-grid mapping. At every encoder reading, we we predict the robot's next position. At every lidar reading, we update the robot pose and grid map. We ensure the timestamps remain synchronized between FOG, lidar, and encoder so none get ahead of the others.

Mapping: We create a 1400x1400 occupancy grid map with 1m resolution and initialize a set of N=40 particles. Each particle has its own 3D state $\mu = [x, y, \theta]$ representing a possible vehicle 2D position (x, y) and orientation θ. Each particle also has a weight α initialized as 1/N. Given a new lidar scan z_{t+1}, we remove points that are too close (<2m) or too far (>75m). Using the vehicle state μ at the largest weighted particle α, we update the map.

The end points of the lidar scan are converted to cartesian xyz-coordinates where z=0. Then, it is transformed from the lidar frame $\{L\}$ to the body to the world frame. We convert them to grid cell units and find the cells between the lidar start and end points using bresenham2D. The detected cells are potentially occupied and the rest are free. We update the log-odds map cells as occupied or free by increasing or decreasing cells by log(4) respectively. To avoid overconfident estimation, we clip the cell values as shown below.

$$\lambda_{i,t+1} = \lambda_{i,t} \pm \log 4$$

$$\lambda_{MIN} = -10log(4) \leq \lambda_{i,t} \geq \lambda_{MAX} = 10log(4)$$

Plotting the first lidar scan to check that our transforms are correct, we get the following reading and grid map output.

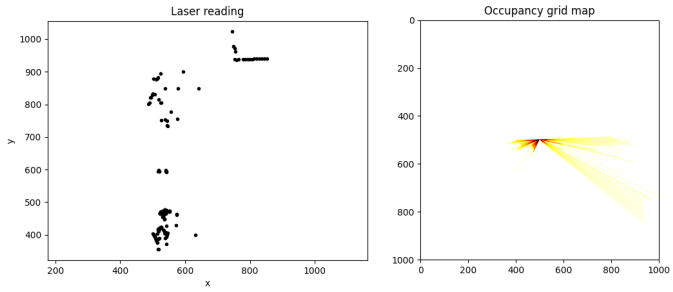


Fig. 1: First Lidar Scan

When we want to recover the map pmf from the log-odds map for plotting, we use the following logistic sigmoid function. We generate two plots every 100 iterations. One shows the detected walls and the other shows the lidar map.

$$\gamma_{i,t} = exp(\lambda_{i,t})/1 + x_(\lambda_{i,t})$$

Prediction: Given an encoder reading z_t with a timestamp in nanoseconds, we calculate the time between readings.

$$\tau = z_t - z_{t-1}/10^9$$

Then, we find the linear velocity for each wheel linear velocity v_t in seconds. The equation below shows how to obtain linear velocity given time traveled τ_t, encoder count z_t, wheel diameters d, and an encoder resolution of 4096. Since we have two wheels, we average their velocities. We find the vehicle begins traveling at 4.6m/s

$$\tau_t v_t \approx \frac{\pi d z_t}{4096}$$

We are given $\Delta\theta$ from the FOG sensor's change in yaw detection. We use the differential-drive motion model to predict the updated pose of each particle $\mu_t = [x_t, y_t, theta_t]$.

$$\mu_{t+1} = \mu_t + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} + _t$$

For Gaussian noise $_t$, we choose mean to be 0. Because linear and angular velocity has high noise values, we replace the variance parameters with max($\Delta x/10$), max($\Delta y/10$), and max($\Delta theta/10$).

$$\epsilon_t \sim \mathcal{N}\left(0, \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\omega^2 \end{bmatrix}\right)$$

Update:

Once the prediction works, we update the robot pose using scan-grid correlation. Lidar scans that are too close (<2m) or too far (>75m) are removed. We update the particle weights using the laser correlation model. First, the lidar scan z_{t+1} is converted to xy-coordinates and transformed from lidar to body to world frame.

We create a 9x9 grid around each particle's pose. Then, we find all map cells y_{t+1} that correspond to the lidar scan when projected to the world frame using the pose of particles $\mu_{t+1|t}$ and a constant d. The role of d is to stabilize the exponential function so there's no numerical overflow.

$$d = \max_k corr(y_{t+1}^{(k)}, m)$$

This gives β. Then, we use the softmax function to normalize and get particle weights α. We replace each particle's weight by α.

$$\beta^{(k)} = exp(corr(y_{t+1}^{(k)}, m) - d)\alpha_{t+1|t}^{(k)}$$

$$\alpha_{t+1|t+1}^{(k)} = \frac{\beta^{(k)}}{\sum_j \beta^{(j)}}$$

The state of the particle with greatest weight becomes the vehicle's current position and is appended to the overall trajectory. Now, we can update the log-odds map again. We start

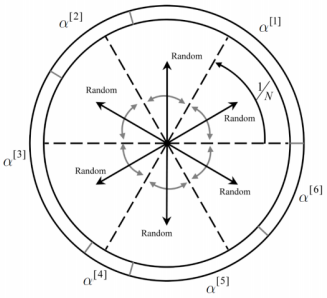
at the particle with the largest weight and project the lidar scan z_t from that location.

Resampling:

To combat particle depletion, we resample when N_{eff} is below a threshold. We set the threshold to 8 which is 20% of our particle count $N = 40$. Resampling focuses particles towards likely regions and leaves unlikely regions with only a few particles. Particularly, we perform stratified resampling which guarantees samples with large weights appear at leaset once and those with small weights at most once. The algorithm is shown below.

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



Texture Mapping: Using RGB images and calculated disparity from the largest-weight particle's pose, we can project colors to the occupancy grid cells. Because our stereo images are in Bayer format, we convert them into BGR.

4. RESULTS

We run our SLAM algorithm on FOG, Lidar, and encoder data after synchronization with 40 particles. We plot our occupancy grid map every 100 iterations. The total number of iterations is the sum of the number of lidar and encoder data. Every 10,000 iterations, the map and wall plot outputs are saved as images.

The resulting plots show good results. The car starts from the top right of the map and begins driving down while curving to its right. The car trajectory is the white line on the road. The map is a smooth curve and looks like a road. It matches the stereo data from the front cameras of the car. The car stops from iteration 10,000 to 30,000 and 50,000 to 60,000 during the downward path where linear velocity becomes zero.

This is probably due to a red light or stop sign. We notice noisy lidar readings sticking out of each side of the road. This makes sense since there is some noise in the projection scans while objects like leaves, trees, people, or poles can cause reflections. In addition, the resolution is not detailed enough to show the shape and we can't see height on a 2D lidar scan.

5. CONCLUSION

In this paper, we implemented a particle filter SLAM algorithm to localize a driving car and its location and surrounding environment with lidar scans onto an occupancy grid map. We tracked the car's pose with a differential-drive motion model by computing its linear velocity with encoder data readings from its wheels. The angle of the car's direction was tracked using a FOG sensor which detects change in yaw. The lidar scans were successfully transformed between the lidar and world frames. The car pose was also successfully transformed between the world frame and grid coorinates. Further research would be increasing the efficiency of the algorithm by using findContours instead of bresenham2D, improving the mapCorrelation algorithm, and vectorizing some loops in the code to increase computational efficiency.

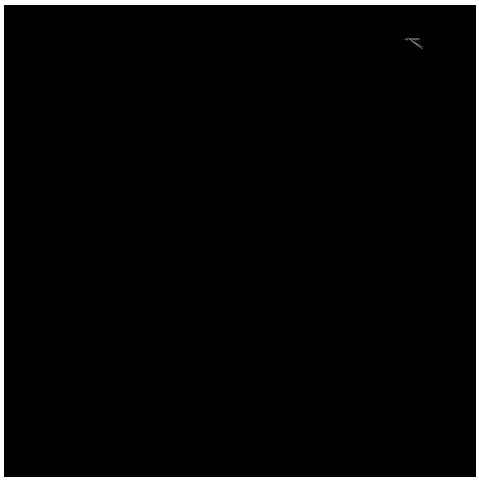


Fig. 2: Occupancy Grid Map, Iteration 0

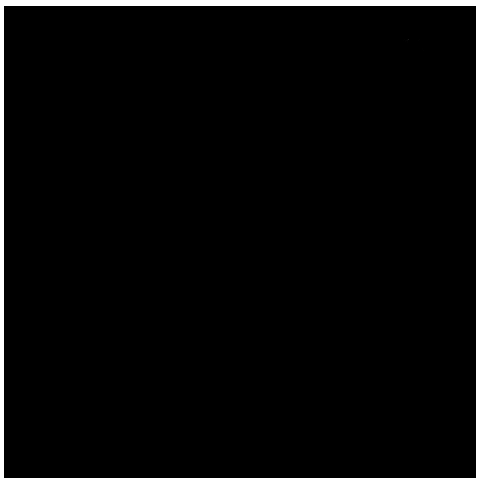


Fig. 3: Occupancy Grid Map Walls, Iteration 0

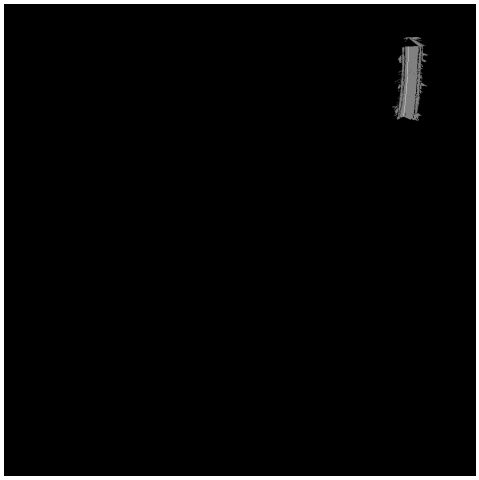


Fig. 4: Occupancy Grid Map, Iteration 10000

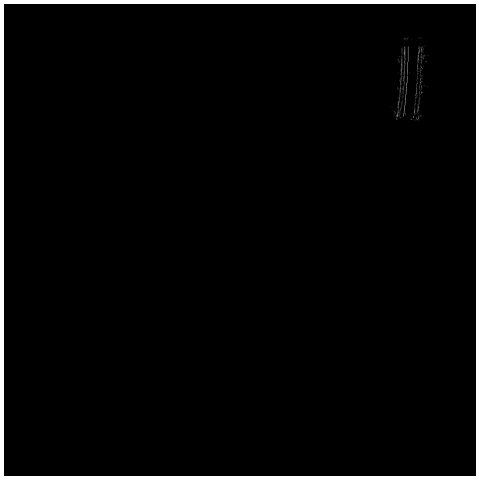


Fig. 7: Occupancy Grid Map Walls, Iteration 20000

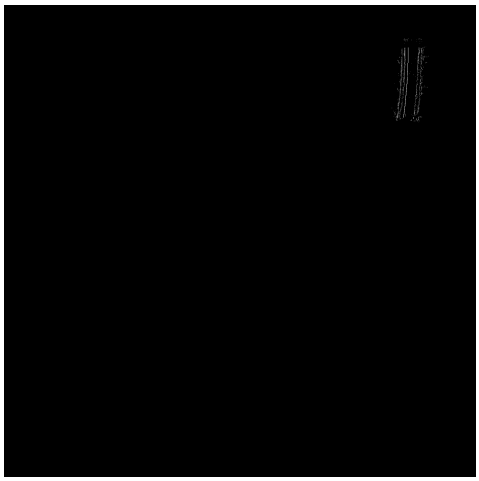


Fig. 5: Occupancy Grid Map Walls, Iteration 10000

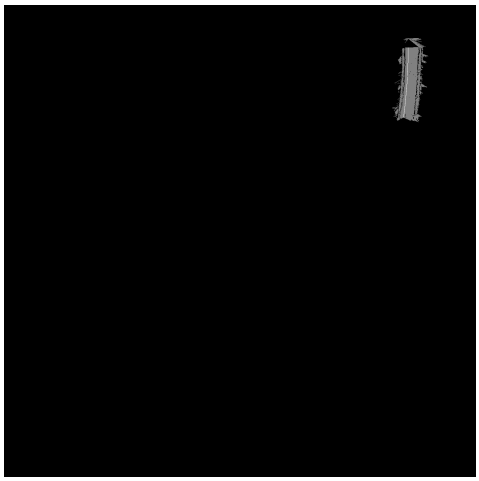


Fig. 8: Occupancy Grid Map, Iteration 30000

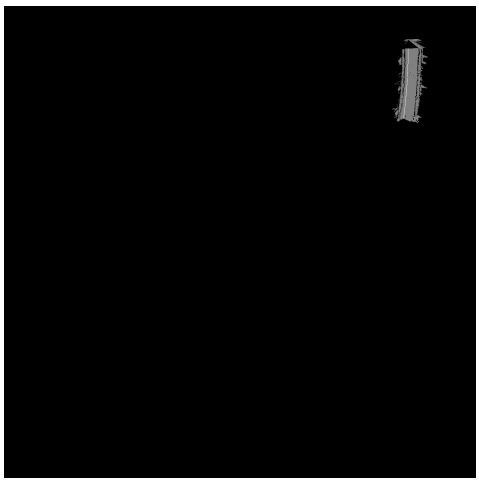


Fig. 6: Occupancy Grid Map, Iteration 20000

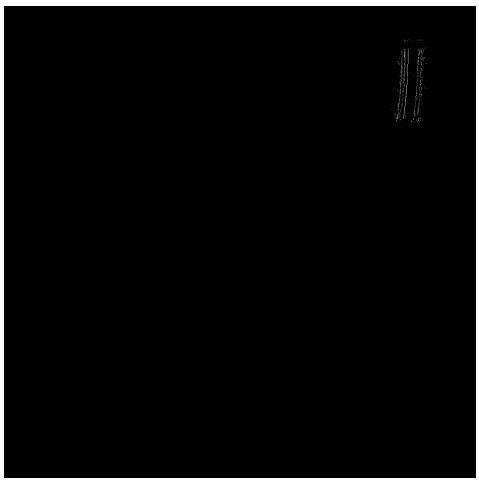


Fig. 9: Occupancy Grid Map Walls, Iteration 30000

Fig. 10: Occupancy Grid Map, Iteration 40000

Fig. 13: Occupancy Grid Map Walls, Iteration 50000

Fig. 11: Occupancy Grid Map Walls, Iteration 40000

Fig. 14: Occupancy Grid Map, Iteration 60000

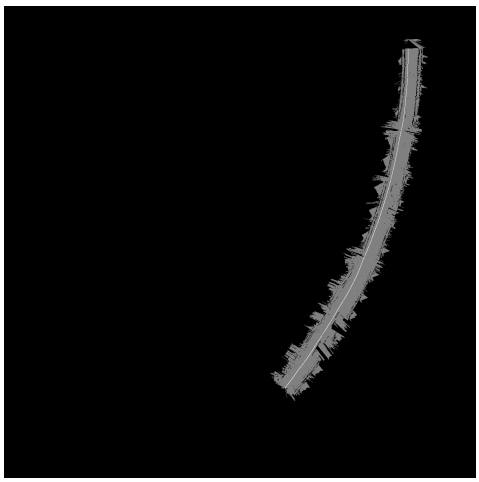


Fig. 12: Occupancy Grid Map, Iteration 50000

Fig. 15: Occupancy Grid Map Walls, Iteration 60000

Fig. 16: Occupancy Grid Map, Iteration 70000

Fig. 19: Occupancy Grid Map Walls, Iteration 80000

Fig. 17: Occupancy Grid Map Walls, Iteration 70000

Fig. 20: Occupancy Grid Map, Iteration 90000

Fig. 18: Occupancy Grid Map, Iteration 80000

Fig. 21: Occupancy Grid Map Walls, Iteration 90000