ECE 276A: Particle Filter SLAM

or: How I Learned to Live Without Sleep

Roumen Guha
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, United States
roumen.guha@gmail.com

***Abstract*—This paper presents a particular solution to tackle the problem of simultaneous localization and maximization (SLAM) via the particle filter algorithm (PF) with the THOR robot configuration of sensors. Very briefly, we implemented a particle filter to localize a robot based on odometry readings and lidar scans, then used this filter to compose a map of the robot's surroundings. The results section contains pictures and animations concerning the effectiveness of this algorithm.**

***Index Terms*—lidar, odometry, particle filter, localization, mapping**

I. Introduction

The goal of this project was to simultaneously localize a robot in its surroundings using the given lidar and odometry readings. Further, using these readings, we are to generate a map this previously unknown indoor environment. A particle filter approach was used for this task.

Particle filters (a form of a Bayesian filter) can effectively model a robot's location in the world as a discrete probability distribution based on past control inputs and past and current observations. In the Problem Formulation section, we will succinctly model the filter in mathematical terms. In the Technical Approach section, we will detail the implementation of the particle filter, discussing how the prediction, update, and resampling steps were chose, as well as some implementation details that might be interesting for further study. Finally, in the Results section, we will discuss the performance of the algorithm with the chosen hyperparamaters on the given lidar and odometry datasets.

II. Problem Formulation

In this section, we precisely define the quantities we are interested in. We also cover some necessary knowledge necessary to understand how this algorithm was implemented.

A. Occupancy Grid (Mapping)

At any time t, we are given lidar scan observations $z_{0:t}$. Using these readings, and our current estimate of the robot's pose (i.e. trajectory) in the world $x_{0:t}$, we generate and maintain a log-odds grid m_t that is used to record the robot's current surroundings.

This project was worked on with Maria Harris, Duke Lin, Shubha Bhaskaran, Stephen West, and Will Argus.

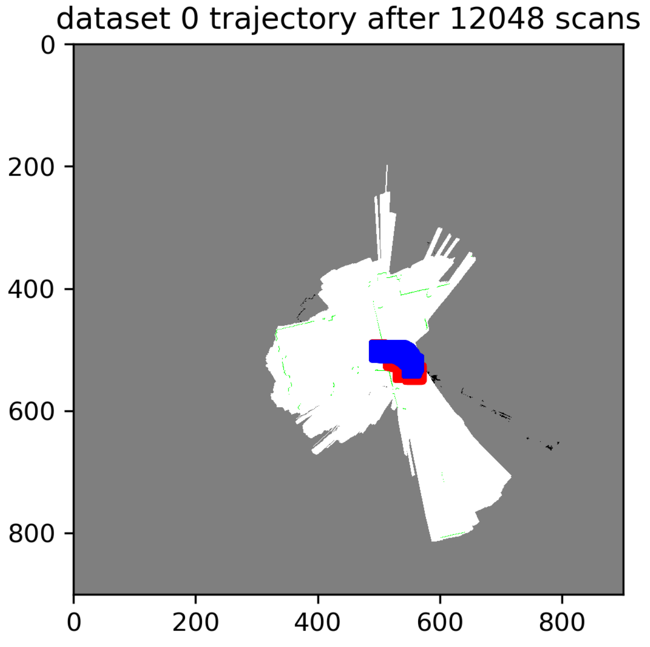

Fig. 1: Example of particle filter SLAM. Blue is raw cumulative odometry reading, red is pose of particle with greatest weight.

Each cell i in the grid m_t is modeled as occupied $(+1)$ or free (-1) independently of other cells. If we define $\gamma_{i,t} := p(m_i = 1|z_{0:t}, x_{0:t})$, then:

$$m_{i,t} = \begin{cases} +1(\text{occupied}), & \text{with prob. } \gamma_{i,t} \\ -1(\text{free}), & \text{with prob. } 1-\gamma_{i,t} \end{cases} \tag{1}$$

And the odds ratio $o_{i,t} := o(m_i|z_{0:t}, x_{0:t})$ of a cell, updated over time by the measurements is given by:

$$\begin{aligned} o_{i,t} &= \frac{p(m_i = 1|z_{0:t}, x_{0:t})}{p(m_i = -1|z_{0:t}, x_{0:t})} = \frac{\gamma_{i,t}}{1-\gamma_{i,t}} \\ &= \frac{p(z_t|m_i = 1, x_t)}{p(z_t|m_i = -1, x_t)} \times \frac{\gamma_{i,t-1}}{1-\gamma_{i,t-1}} \end{aligned} \tag{2}$$

Here, we can take the log of this expression to obtain $\lambda_{i,t} := \lambda(m_i|z_{0:t}, x_{0:t}) := log(o(m_i|z_{0:t}, x_{0:t})) = log(o_{i,t})$ to update our log-odds map:

$$\lambda_{i,t+1} = \lambda_{i,t} + \Delta\lambda_{i,t+1} \tag{3}$$

where $\Delta\lambda_{i,t+1}$ is a hyperparameter that specifies how much we trust our observation z_{t+1}.

In this way, our update at time $t+1$ for any cell of our log-odds grid is simply to add $\Delta\lambda_{i,t+1}$ to the current value of the cell.

For the initial cell value, before we have any observations to process, it is fine to initialize the cells to zero.

To generate the occupancy grid (a binary image of cells considered either occupied or not occupied), we threshold the log-odds grid based on thresholds chosen via empirical observations. Again, these thresholds are therefore considered hyperparameters.

B. Particle Filter (Localization)

The probability distribution $p(x)$ of the robot's next location x_{t+1} after the **prediction step** can be represented as a mixture of delta functions (representing the motion model p_f in a general Bayesian filter) weighted by the N particle weights:

$$p_{t+1|t}(x) = \sum_{n=1}^{N} \alpha_{t|t}^{(n)} \delta(x; \mu_{t+1|t}^{(n)}) \tag{4}$$

where $\mu_{t+1|t}^{(n)}$ is the prediction for the n^{th} particle given the motion model (in our case, this is just the relative odometry reading at this time with some added zero-mean Gaussian noise to model the uncertainty of our system).

To complete the **update step**, we simply compute the correlation between each particle's next observation and our current knowledge of the world around us (i.e. compare with the occupancy grid) and weight the particle weights accordingly (remembering to normalize, so that the weights resemble a probability distribution):

$$p_{t+1|t+1}(x) = \sum_{n=1}^{N} \frac{\alpha_{t+1|t}^{(n)} p_h(z_{t+1}|\mu_{t+1|t})}{\sum_{n=1}^{N} \alpha_{t+1|t}^{(n)} p_h(z_{t+1}|\mu_{t+1|t})} \tag{5}$$

(where p_h represents the observation model in a general Bayesian filter).

III. Technical Approach

In this section, we discuss the algorithms implemented in this project.

Fundamental to our operations in this project were coordinate-system transformations. We will not go into much detail here, but transformation methods were formed for converting lidar scan coordinates to world-frame coordinates, and converting between world-frame coordinates and map coordinates, accounting for rotations in the robot's head and neck with respect to its body, and the robot's yaw with respect to the world.

We did initially face synchronization issues; the head angles of the robot come from a different sensor, and so the timestamps between the lidar scans and the head angles don't match up perfectly.

Additionally, we faced some trouble with implementing the particle filter; the particles would simply disperse over the map throughout the scans. The order of events in the particle filter algorithm is important, and we've laid it out in the appropriate section.

A. Occupancy Grid (Mapping)

We updated our log-odds grid according to if a lidar beam passed through the cell (free) or if a beam was stopped at the cell (occupied). We used 'cv2.drawContours()' to do this, as it was much faster than using provided Bresenham's line algorithm. Our log-odds update $\Delta\lambda_{i,t+1}$ was set to be the sigmoid of how much we trust our lidar scans (0.8), and this is an important hyperparameter.

When forming the occupancy grid, we thresholded the log-odds grid with another hyperparamater to obtain the binary map.

We also included a hyperparamater called 'memory' for decaying the map without repeated confirmation about previously seen information. We left it set to one for this scenario.

And finally, we inlcuded a variable for how confident we can possibly be about a cell, and we disallow the log-odds grid from being able to exceed this value. This allows us to recover a cell if the state m_i of that space changes.

B. Particle Filter (Localization)

The basic setup of the robot's measurements in this scenario is as follows:

- We have lidar scans available at certain times.
- We have noisy odometry readings that give us some information about our relative movement between timesteps.

We are then trying to localize our robot in an unknown indoor environment using these sensor readings.

The idea of the particle filter is then as follows:

- Generate/maintain N hypotheses (particles) of our current robot pose based off of the relative odometry and some noise (to model the uncertainty in our measurement of the environment). This is the **prediction step** of the particle filter.
- To see which hypothesis (particle, $\mu_{t+1|t}^{(n)}$) is most likely to be correct, estimate the correlation between the information obtained from the lidar scans at $t+1$ to the current occupancy grid, and give each hypothesis a weight $\alpha^{(n)}$ proportional to this correlation so we know which ones are most trustworthy. This is the **update step** of the particle filter.
- At any time we need to update the map given the next lidar scan z_{t+1}, we can use the particle with the greatest weight as the current robot's pose to do so.
- If we find that most of our particles have weights that make them effectively useless, generate new particles based upon the current distribution of weights of the particles, and give them uniform weights. This is the **resampling step** of the particle filter.

- Iterate to the next lidar scan.

For the noise, we generated independent zero-mean Gaussians for the robot's relative odometry in x, y, and θ (torso yaw). We also scaled this noise's variance by the relative odometry, ensuring that if the robot did not move, the particles would not shift unnecessarily. For the variances, we used 10% in the x and y relative odometry readings, and 0.5% in the torso yaw relative odometry reading

For computing the correlation for a given particle and a given scan, we simply computed the sum of cells that agreed between the latest scan and the current binary occupancy grid. We used the computed the softmax of the correlations to obtain the new set of weights, but we subtracted the maximum of the current particles' correlations from all the current particles' correlations. This seemed to result in less resampling overall, meaning we would track the particle with the greatest weight for longer periods of time.

We performed the **resampling step** when $N_{effective} := \frac{1}{\sum_{n=1}^{N}(\alpha^{(n)})^2} < N_{threshold}$, where $N_{threshold}$ is a hyperparameter to be found from experimentation. We implemented low-variance resampling, given in Fig. 2.

The more particles we used, the better the results got. But, over time, we noticed that the datasets got progressively more and more difficult. So we used 5 particles for the first dataset, 10 for the second, 20 for the third, 40 for the fourth, and 80 for the fifth.

```
Low_variance_resampling($\mathcal{X}_t, \mathcal{W}_t$):
1:  $\bar{\mathcal{X}}_t = \emptyset$
2:  $r = \text{rand}(0; M^{-1})$
3:  $c = w_t^{[1]}$
4:  $i = 1$
5:  for $m = 1$ to $M$ do
6:      $U = r + (m-1) \cdot M^{-1}$
7:      while $U > c$
8:          $i = i + 1$
9:          $c = c + w_t^{[i]}$
10:     endwhile
11:     add $x_t^{[i]}$ to $\bar{\mathcal{X}}_t$
12: endfor
13: return $\bar{\mathcal{X}}_t$
```

Fig. 2: Low variance resampling algorithm. Obtained from The University of Freiburg.

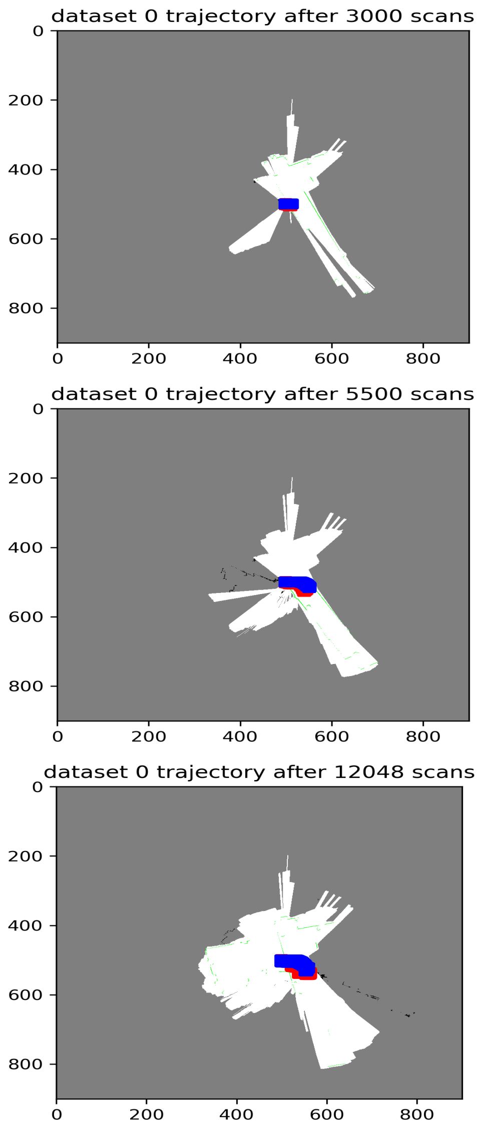

Fig. 3: Particle filter in dataset 0. Red is highest-weighted particle, blue is raw cumulative odometry.

IV. Results

A. Discussion

As we can see from the results above, our algorithm didn't produce the cleanest results, especially in datasets 3 and 4. In the first three datasets, we see that the particles follow the odometry readings fairly closely. In dataset 0, the particles even rotate appropriately, following the raw cumulative odometry. We can see from the results of the other datasets that rotations seem to consistently be a problem; perhaps the torso yaw reading is especially noisy.

We suspect that with more time spent trying different hyperparamaters (as there were several in this project), a far better solution could have been found with more optimal parameters. Perhaps if we had access to the THOR robot, we could record the odometry readings with identical input to give us an idea of the noise distribution in its sensors, allowing us to model the noise more accurately in the particle filter's prediction step. We additionally think it is likely that with different thresholds and values of the trust parameter we would see the most improvement, and suggest this for further work.

More extensions could include pre-filtering the relative odometry readings to clean them up before using them in the particle filtering algorithm, or perhaps modelling lidar noise as a Gaussian random variable.

Finally, we'd like to make a suggestion to our future selves: don't take more than one project class per quarter.

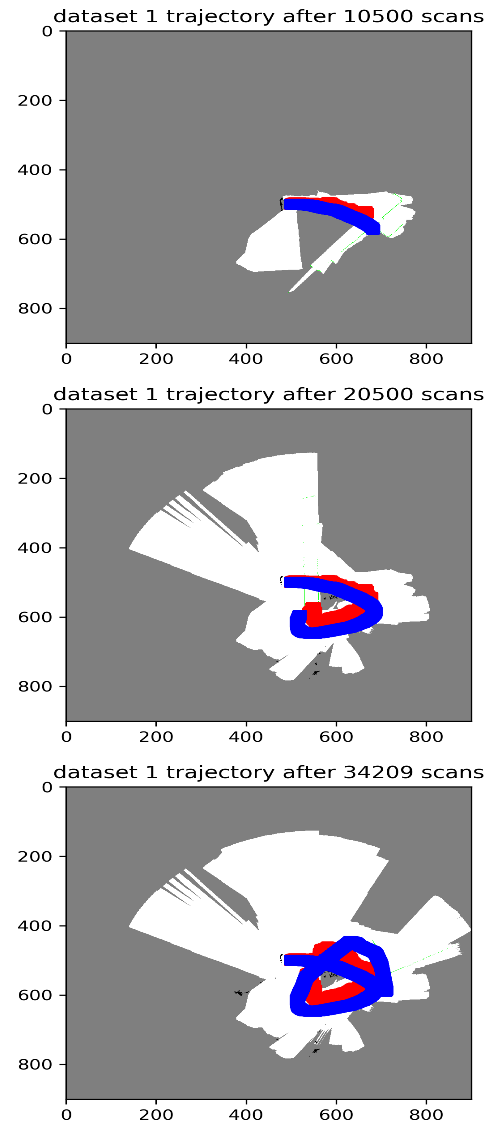

Fig. 4: Particle filter in dataset 1. Red is highest-weighted particle, blue is raw cumulative odometry.

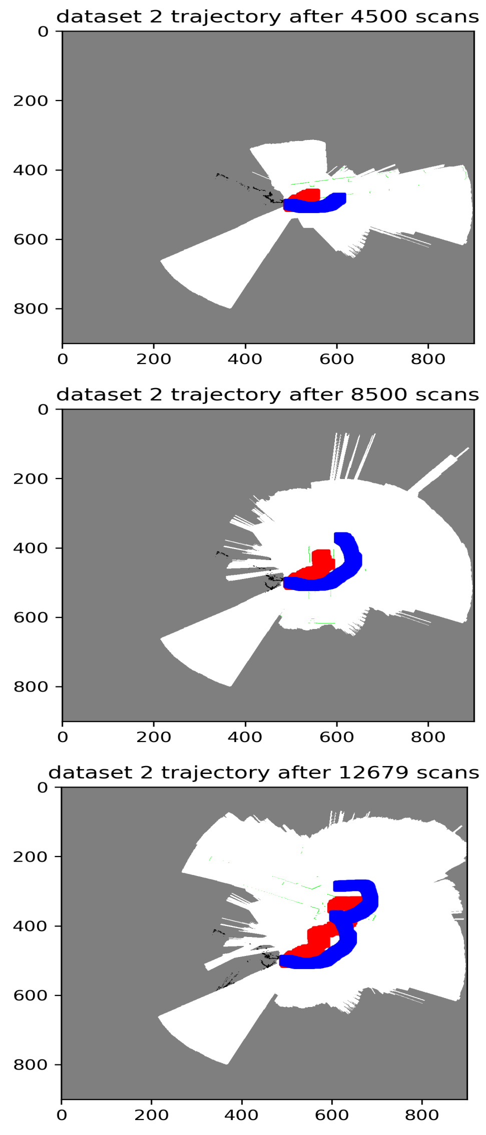

Fig. 5: Particle filter in dataset 2. Red is highest-weighted particle, blue is raw cumulative odometry.

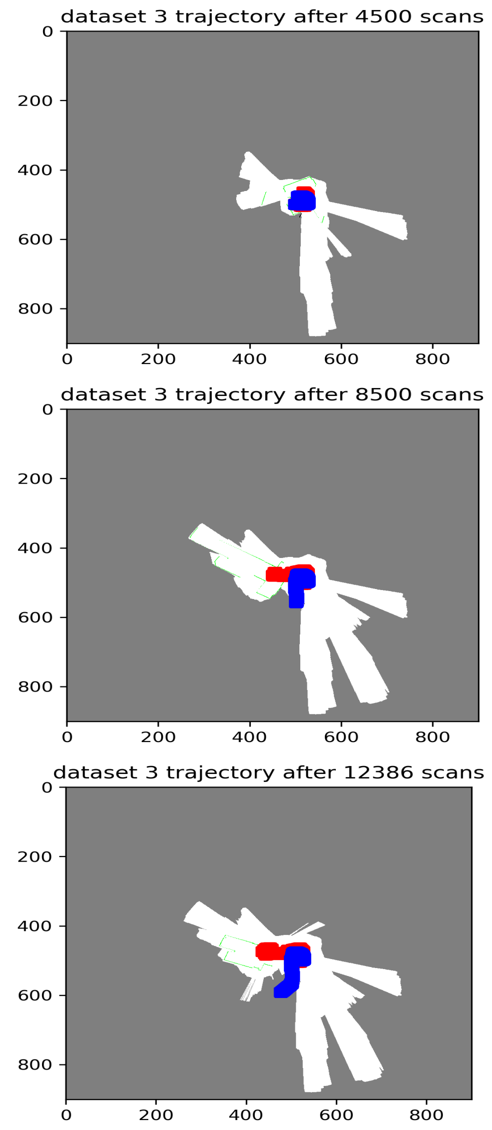

Fig. 6: Particle filter in dataset 3. Red is highest-weighted particle, blue is raw cumulative odometry.

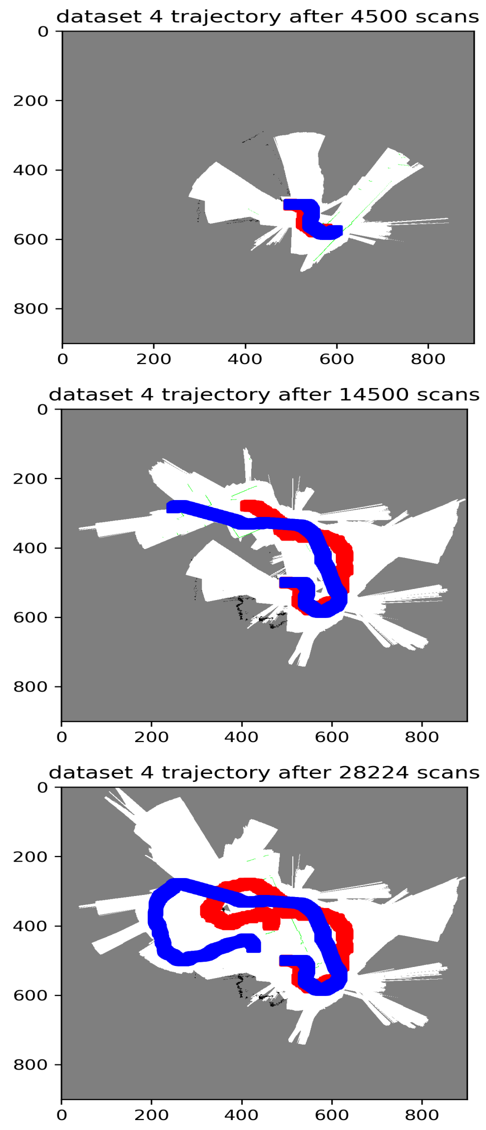

Fig. 7: Particle filter in dataset 4. Red is highest-weighted particle, blue is raw cumulative odometry.