Lab 1 Report - Particle Filter
16831 - Statistical Techniques in Robotics
Chunyang Sun, Eric Westman, Samuel Yim

Approach

To obtain global convergence on a mobile robot given odometry data and laser data we implemented a particle filter as described in class lectures this semester as well as in *Probabilistic Robotics* [1]. The steps of the filter are summarized below.

(1) Read map and log files
(2) Draw initial particles
(3) Apply motion model
(4) Update particle weights
(5) Resample particles (importance sampling)
(6) Visualize particles + map
(7) Go to step (3)

Results

The particles converged in each of the two log files we used. For the first log file, we found that the particle filter could robustly handle unexpected obstacles in the line of sight of the laser scanner (e.g. people). For the second log file we tested, results were inconsistent.

Implementation

We implemented our particle filter in C++, with visualization in OpenCV, utilizing only C++ 11 standard libraries.

Sampling distribution

20000 initial particles are uniformly randomly sampled from the empty hallways and rooms, which are the only reasonable locations for the robot to start. In each iteration, we resample the same number of particles.

Motion Model

We used a simple motion model that applied the changes in x, y, and theta from odometry to each of the particles, with random Gaussian error that varied for x,y motion and for theta rotation. In other words, for all particles $\Delta x_{particle} = \Delta x_{odometry} + \text{randNormal}(0, \sigma)$, and similarly for the $\Delta y_{particle}$ and $\Delta theta_{particle}$.The exact values used can be found in the Parameters section.

Weight the Particles

After drawing particles in the map, we used a Ray-casting method to simulate lasers to 800 grid cells, starting from the right of the robot and going 180 degrees to the left and processed the weight of each particles in the following steps:

1. If the ray goes outside of the image before finding a wall, give low weight.
2. If the ray sees something outside the map that we don't know about (hits -1 probability), give it a lower weight.
3. If the ray hits a wall, then use the observation model.

Each of the weights are added together in log space and converted back to a probability before being assigned to the appropriate particle.

Observation Model

The sensor model is comprised of four separate functions as illustrated by the lecture notes. These functions are

(1) a gaussian which corresponds to the likelihood of the laser measurement of a known obstacle in the map
(2) an exponential which corresponds to the likelihood of measuring unknown obstacles
(3) a uniform distribution which corresponds to the likelihood of measuring unknown obstacles and general uncertainty in measurements and
(4) a constant piecewise distribution which corresponds to max range measurements.

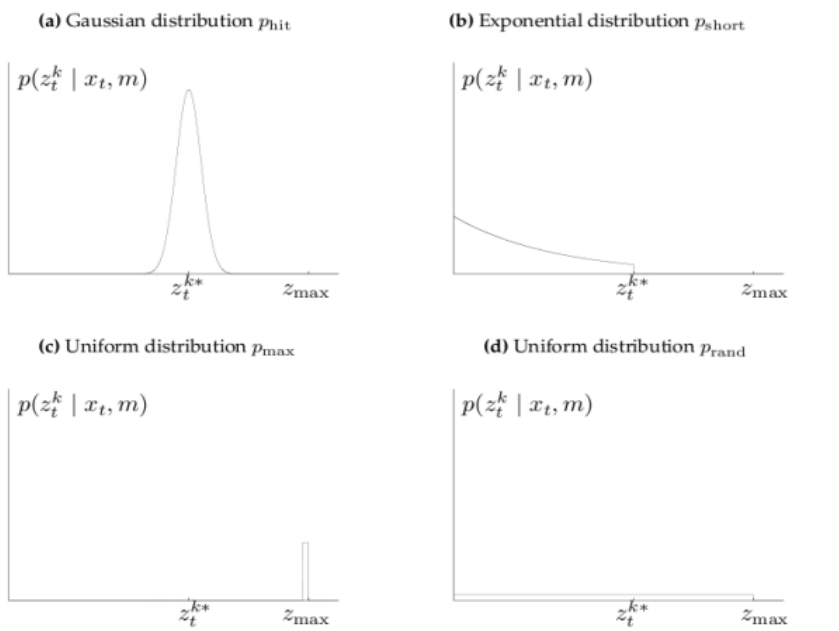

Figure 6.3 Components of the range finder sensor model. In each diagram the horizontal axis corresponds to the measurement z_t^k, the vertical to the likelihood.

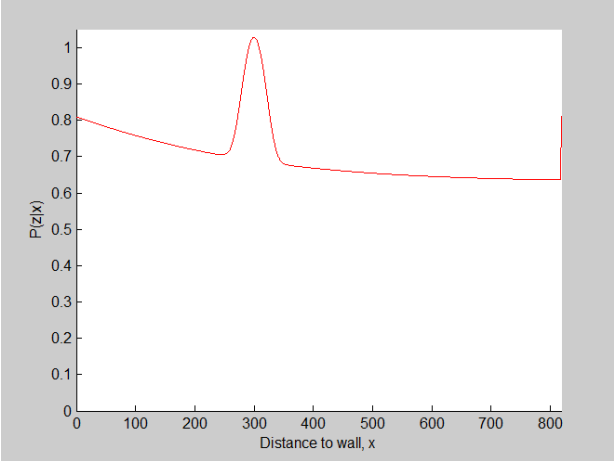

Figure 1. The observation model used in the final implementation [1]

The distribution shown above in Figure 1 is the sum of the four functions taken to a power of ⅕. We found that without this power component the particles converged almost instantly, and not always to the correct particle. By adding in the power of ⅕, we were able to slow down the convergence.

Resampling Procedure

We resampled the particles according to the weight of each particle. More particles are sampled from those particles with higher weight. The resampled particles are from the original pool of particles and the number of total particles are the same. The resampled particles will spread out after we apply motion model on them.

In the implementation, we used the piecewise_constant_distribution which conveniently takes a list of unnormalized weights and resamples the integer indices of the particles based on the corresponding weights.

Parameters

Number of initial particles: **10000**
Motion model variances: **x - 1 grid cell, y - 1 grid cell, theta - 1 degree**
Sensitivity of observation model (the power we raise it to): **0.2**
Gaussian function of the observation model: **sigma_g = 10 grid cells**
To speed up the weighting and to prevent overfitting, only look at every nth laser angle: **n = 3**

Future Work

To further increase the accuracy of the filter, it is possible to tune the particle filter more finely.

In fact, tuning various the various parameters of our observation model and sampling method comprised most of the work we did to get a functioning particle filter.

The code currently runs at 1-2 frames per second using 20000 particles. To speed this up, we could parallelize the code. One of the advantages of using a Bayes filter like the particle filter is that each particle can be dealt with separately until the resampling step, which makes it ideal for parallelization.

Our implementation of the particle filter does not solve the kidnapped robot problem. One way we could deal with this problem is to distribute some percentage of particles during each resampling step in random positions throughout the entire map. In this way, there will always be some probability that the particles can converge to the correct location.

Another feature that could be added (and which in hindsight we wish we had implemented) is low variance resampling. We believe that implementing low-variance resampling would help our filter converge to one clump of particles more slowly, and would increase the robustness of the algorithm to different types of maps.

References

1. Probabilistic Robotics. Sebastian Thrun, Wolfram Burgard, Dieter Fox.
2. http://www.cs.cmu.edu/~16831-f14/notes/F14/16831_lecture02_prayana_tdecker_humphreh.pdf