CS8803: STR, Spring 2016. Lab 1: Robot Localization

Chih-Li Sung, Yang Tian, Xiao Wang

Georgia Institute of Technology

February 24, 2016

1 Introduction

The goal in this project is to find out the global localization of a lost robot. The robot is operating in a building with nothing but odometry and a laser rangefinder. Fortunately, a map, shown in Figure 1, and a deep understanding of particle filtering could help it localize. In this report, we utilize these resources and implement a particle filtering algorithm to localize the robot. The existing resource is briefly described in Section 2. The detail of the particle filtering algorithm is presented in Section 3. Some results are shown in Section 4. Some extra credits are performed in Section 5. The conclusion is given in Section 6.

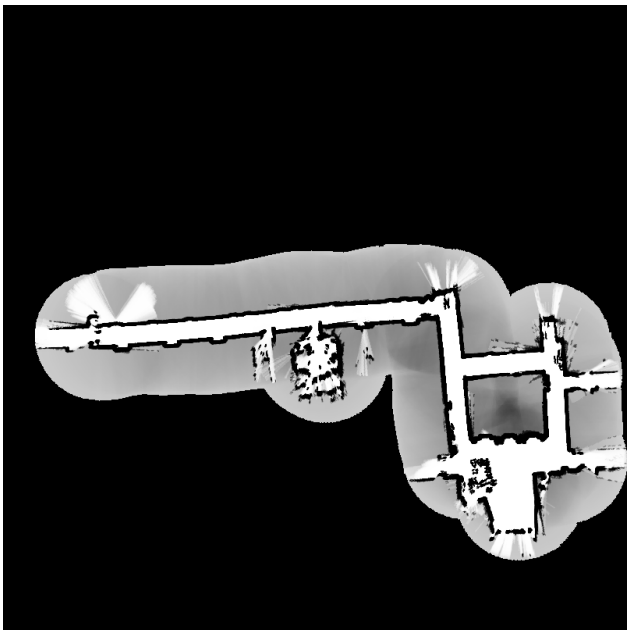

Figure 1: *The map of the building.*

2 Data Description

The file of raw data includes the following data:

1. Image of map (Figure 1), where the values shows the probability for occupancy, including -1 (unknown), 1 (occupied with probability 1), 0 (unoccupied with probability 1), and 0.5 (occupied with probability 0.5).

2. Odometry data: coordinates of the robot in standard odometry frame.

3. Laser data: 180 range readings which read coordinates of laser in standard odometry frame when the laser reading was taken.

2.1 Preprocessing

There are two types of data needing to be imported, i.e. the raw map file (wean.dat) and robot data (robotdata$*$.log). For the former data, we simply use the function provided by Prof. Byron Boots and add other necessary functions and map type definition according to [2]. The only noticeable thing is that when we print the map, we defines unknown points and occupied grids as black, grids unoccupied with probability of 1 as white, and uncertain grids as gray. For robot log data reading, we define a data log class, which consists of robot's coordinates and laser's coordinates in the standard odometry frame, as well as 180 laser range readings. Here we preprocess position data by using the map resolution 10.

3 Algorithm of Particle Filter

In order to explore the location of the lost robot, since the motion information of robot (odometry data) and the measurement of environment (laser data) are provided, the particle filter algorithm is considered. Following the same notation of Textbook [1], the particle filter algorithm is stated in Algorithm 1. In the following sections, line 3, 4 and 8 are described in details.

Algorithm 1 Algorithm Particle Filter(χ_{t-1}, u_t, z_t)

1: $\bar{\chi}_t = \chi_t = \emptyset$
2: **for** $m = 1$ to M **do**
3: sample $x_t^{[m]} \sim p(x_t|u_t, x_{t-1}^{[m]})$
4: $w_t^{[m]} = p(z_t|x_t^{[m]})$
5: $\bar{\chi}_t = \bar{\chi}_t + < x_t^{[m]}, w_t^{[m]} >$
6: **end for**
7: **for** $m = 1$ to M **do**
8: draw i with probability $\approx w_t^{[i]}$
9: $x_t^{[i]}$ to χ_t
10: **end for**
11: **return** χ_t.

3.1 Line 3 in PF Algorithm: Motion Model

Although technically odometry are sensor measurements, not controls, we can treat odometry measurements as controls. Fortunately, Textbook [1] provides the motion model which derives the robot motion based on the coordinates of the robot in odometry frame (x, y and θ). The motion model is stated in Algorithm 2 following the same notations in Textbook [1]. In order to make sure that the new possible locations are indeed inside this building, in line 13 of Algorithm 2, we add the condition that if the map value at x', y' is larger than 0.8 (the probability for occupancy is more than 0.8; $m(x', y')$ denotes the map value at coordinate (x', y')), then return the new location, otherwise return the original location.

Algorithm 2 Motion Model(u_t, x_{t-1})

1: $\delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$
2: $\delta_{\text{trans}} = \sqrt{(\bar{x}' - \bar{x})^2 + (\bar{y}' - \bar{y})^2}$
3: $\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$
4:
5: $\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \textbf{sample}(\alpha_1\delta_{\text{rot1}} + \alpha_2\delta_{\text{trans}})$
6: $\hat{\delta}_{\text{trans}} = \delta_{\text{trans}} - \textbf{sample}(\alpha_3\delta_{\text{trans}} + \alpha_4(\delta_{\text{rot1}} + \delta_{\text{rot2}}))$
7: $\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \textbf{sample}(\alpha_1\delta_{\text{rot2}} + \alpha_2\delta_{\text{trans}})$
8:
9: $x' = x + \hat{\delta}_{\text{rot1}} \cos(\theta + \hat{\delta}_{\text{rot1}})$
10: $y' = y + \hat{\delta}_{\text{trans}} \sin(\theta + \hat{\delta}_{\text{rot1}})$
11: $\theta' = \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}$
12:
13: **if** $m(x', y') > 0.8$ **then**
14: $\quad x_t = (x', y', \theta')^T$
15: **else**
16: $\quad x_t = (x, y, \theta)^T$
17: **end if**
18: **return** x_t

3.2 Line 4 in PF Algorithm: Measurement (Sensor) Model

Since laser range finder data are provided in raw data, we refer *beam model of range finder* from Textbook [1] as the measurement model and follow the same notations in the following.

$$p(z_t^k|x_t) = z_{\text{hit}} \cdot p_{\text{hit}}(z_t^k|x_t) + z_{\text{short}} \cdot p_{\text{short}}(z_t^k|x_t) + z_{\text{max}} \cdot p_{\text{max}}(z_t^k|x_t) + z_{\text{rand}} \cdot p_{\text{rand}}(z_t^k|x_t), \quad (1)$$

where $p_{\text{hit}}, p_{\text{short}}, p_{\text{max}}$ and p_{rand} are shown in [1] and $z_{\text{hit}}, z_{\text{short}}, z_{\text{max}}$ and z_{rand} are the corresponding weights. In particular, p_{hit} is a *truncated normal distribution*, that is,

$$p_{\text{hit}}(z_t^k|x_t) = \eta\mathcal{N}(z_t^k; z_t^{k*}, \sigma_{\text{hit},t}^2)$$

if $0 \leq z_t^k \leq z_{\max}$ and otherwise $p_{\text{hit}}(z_t^k|x_t) = 0$, where z_t^{k*} is the mean, $\sigma_{\text{hit},t}^2$ is the variance, $z_{\max}$ is the maximum sensor range, and η is the normalizer such that $\int_0^{z_{\max}} p_{\text{hit}}(z_t^k|x_t) = 1$. Here we take $z_{\max} = 800$ and $\sigma_{\text{hit},t}^2 = (\frac{z_t^{k*}}{1.5} + 2)^2$. . The reason we choose the adaptive variance of the measeuremnt model is that we assume the laser will get larger variance when its expected distance is larger, this is some kind of intrinsic attribute of the machine. Theoretically, we want the small disturb when expected distance is small have the similar probability with large disturb when expected distance is large. The mean value z_t^{k*} can be calculated from x_t and m via ray tracing. The algorithm of calculating mean values z_t^{k*} is shown in Algorithm 3, where input x, y and θ are the coordinate given the current location x_t. The idea of the algorithm is to repeat tracing a ray (x, y, θ) with length δ from location x_t until the ray hits a unoccupied obstacle ($m(x, y) \leq 0.5$). Here we take $\delta = 0.5$. From line 11 to 16 in the algorithm, the xy-coordinate of the laser is surely within the map. Line 19 can make sure the mean value is not too small, which might cause the convergence unstable.

The major issue of implementing the truncated normal distribution is computing the joint probability density of the measurements from 180 angles. If these 180 measurements are taken into account and assume they are independent, the joint probability density decreases exponentially and that causes some possible locations are missed due to the relatively small sampling probabilities. In order to deal with this problem, we only pick up the measurements at angle $j \times 10$ degree, $j = 0, 1, 2, \ldots, 18$. Moreover, the independence assumption is more reasonable in this case.

In the measurement model (1), we set weights $z_{\text{hit}} = 0.9, z_{\text{short}} = 0, z_{\max} = 0, z_{\text{rand}} = 0.1$.

Algorithm 3 Mean value z_t^{k*} in truncated normal distribution $(x, y, \theta, m, z_{\max})$

1: $x_{\max}$:= maximum value of x-coordinate from map m
2: $y_{\max}$:= maximum value of y-coordinate from map m
3: $\Delta x = \delta \cos(\theta)$
4: $\Delta y = \delta \sin(\theta)$
5: $z_t^{k*} = \delta$
6: $p = 1$
7: **while** $p > 0.5$ AND $z_t^{k*} < z_{\max}$ **do**
8: $\quad x = x + \Delta x$
9: $\quad y = y + \Delta y$
10: $\quad z_t^{k*} = z_t^{k*} + \delta$
11: $\quad$ **if** $x > x_{\max} - 0.001$ **then** $x = x_{\max} - 1$
12: $\quad$ **else if** $x < 0$ **then** $x = 0$
13: $\quad$ **end if**
14: $\quad$ **if** $y > y_{\max} - 0.001$ **then** $y = y_{\max} - 1$
15: $\quad$ **else if** $y < 0$ **then** $y = 0$
16: $\quad$ **end if**
17: $\quad p = m(x, y)$
18: **end while**
19: **if** $z_t^{k*} < 3$ **then** $z_t^{k*} = -1$
20: **end if**
21: **return** z_t^{k*}

3.3 Line 8 in PF Algorithm: Sampling with weighted probability

Resampling particles based on their corresponding weights has different implementations. Here we utilize the resampling algorithm mentioned in *Artificial Intelligence for Robotics*, a Udacity online course taught by Sebastian Thrun [3]. The algorithm is described as Algorithm 4. The reason why at each particle iteration we add a random value drawn from a uniform distribution from 0 to (2 × maximum of weights), is that by doing so we can add a random value whose expectation is exactly the maximum of weights. Then we can get particle index according to the weights distribution.

Algorithm 4 Resampling(weights, old particles)

1: Normalize weights s.t. the sum equals to 1
2: $\beta = 0$
3: index $\sim$ Unif$(0, \text{size of weights} - 1)$
4: **for** each particles **do**
5: $\quad\beta = \beta + \omega$, where $\omega \sim$ Unif$(0, 2 \times \text{maximum of weights})$
6: $\quad$**while** $\beta >$ weights[index] **do**
7: $\quad\quad\beta = \beta -$ weights[index]
8: $\quad\quad$index = (index + 1)% (size of weights)
9: $\quad\quad$pick particles[index] as a new particle
10: $\quad$**end while**
11: **end for**
12: **return** new particle

3.4 MCL agorithm

The main MCL algorithm is shown in Algorithm 5. Line 1 checks whether the difference between every continous two data satisfies some criteria, in which we think the robot doesn't move. Line 2 checks whether the data type is laser data, if it is laser data, we implement particle filter algorithm at every five steps. The reason is because that if we ignore all the laser data which lead not to move, the robot becomes more uncertain about the current location and that causes the convergence rate is very slow. Line 6 implements that robot moves according to motion model if the robot indeed moves, then do particle filter algorithm. In addition, we apply some threshold to avoid the result which suddenly converges to a wrong point. This is done by checking the changes in variance among all particles, where the the variance was defined by the the average of sample variances of x, y, θ from potential particles. If it is extremly small, we will ignore this laser data because it may cause an extreme point in these particles.

4 Result

The particle algorithm gives the results with one paricular location in log_1, log_3, log_4. The Figure 2 shows the final robot's location from log_1 data as an example. The more

Algorithm 5 MCL algorithm (x, y, θ)

if $\Delta x < 0.3$ AND $\Delta y < 0.3$ AND $\Delta\theta < 0.01$ **then**
  if data_type is "L" AND $t\%5 = 0$ **then**
    Sampling by **Algorithm Particle Filter** (χ_{t-1}, u_t, z_t)
  end if
else
  Move based on **Motion Model**(u_t, x_{t-1}) and sampling by **Algorithm Particle Filter** (χ_{t-1}, u_t, z_t)
end if
return new particle

results presented by video are shown in `https://www.youtube.com/watch?v=rKym_mHV6lo`. Moreover, the convergence rate of each data is shown Figure 3, where the y-axis is computed by the average of sample variances of x, y, θ from potential particles. However, in log_2, log_5, the results of convergence perform not so well. More details are presented in Section 6.

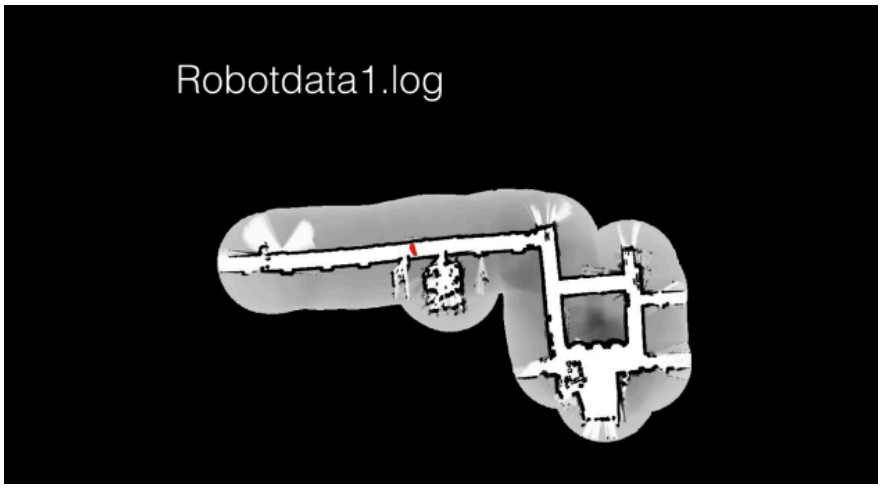

Figure 2: *The final robot's location from log_1 data.*

5 Extra credit

5.1 Kidnapped robot problem

Kidnapping a robot is about blinding a robot and taking it to another place without odometry and laser range data. We can simply fuse two data log and create an occasion where the robot is kidnapped to an unknown place. Using robot log can sufficiently tackle this problem, since the robot coordinates are described in the global frame. In other words, if the robots' coordinates have major change, then it seems that the robot has been somehow "kidnapped". More specifically, when processing each entry of the log data, we add an if sentence to determine the robot has been moved greatly in extremely short time. If so, we re-initialize all particles and start localization from scratch again.

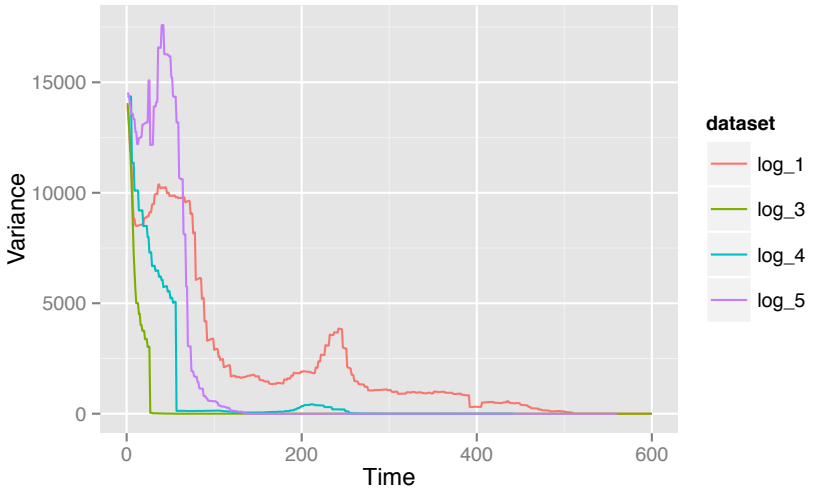

Figure 3: *Convergence rate in each dataset.*

5.2 Adaptive number of particles

For the adaptive number of particles, we use the KLD-sampling from Textbook Chapter 8 [1], which is shown in Algorithm 6. In the method, we eatimate the particles by reducing the KL-distance between the estimated distribution for $p(x_t|\hat{z_{1:t}}, x_{1,t-1})$ and the true distribuiton $p(x_t|z_{1:t}, x_{1,t-1})$. However, the true distribution is unknown. This method uses the idea by counting the volume covered by the current particles. Here we apply the bins of size $0.4 \times 0.4 \times \frac{\pi}{24}$. The too large bins lead to wrong convergence while too small bins make the sampling extremely small.

This method spreads more than 150000 particles to fullfill the requirement of KL-distance. After that, it decreases the number since it becomes more certain about where he is. Lastly, the number decreases to about 100, which makes the computation faster.

6 Conclusion and discussion

The particle filter algorithm performs well in this lab. Since we do not have the true location of the lost robot, we can not measure the difference by some metric. Nevertheless, we check the video generated and that shows it performs well for each data-log in terms of convergence rate.

The goal of this lab is that given the map data and the readings from robots, we would like to localize a lost robot. What we do in this lab is using particle filter algorithm, which includes the fundamental motion model and measurement model, to estimate the posterior distribution. In this algorithm, the motion model creates uncertainty in the robots' belief for its movement according to some gaussian model. However, the measurement model is very crucial and hard to implement in this lab since we want all particles to quickly converge to the true location while the convergence is robust for extreme cases. Sometimes the particle filter abruptly converges to wrong location since it is very sensitive to the initialization of the particles. Therefore, it is a tradeoff between the convergence rate and correct localization.

Algorithm 6 Algorithm KLD_sampling_MCL($\chi_{t-1}, u_t, z_t, m, \epsilon, \delta$)

$\chi_t = \emptyset$
$M = 0, M_\chi = 0, k = 0$
for all b in H do **do**
    $b =$ empty
end for
while $M < M_\chi$ or $M < M_{\chi_{\max}}$ **do**
    draw i with probability $\propto w_{t-1}^{[i]}$
    $x_t^M =$ **sample_motion_model**$(u_t, x_{t-1}^{[i]})$
    $w_t^M =$ **measurement_model**$(z_t, x_t^{[M]}, m)$
    $\chi_t = \chi_t + < x_t^{[M]}, w_t^{[M]} >$
    if $x_t^{[M]}$ falls into empty bin b **then**
        $k = k + 1$
        $b =$ non-empty
        if $k > 1$ **then**
            $M_\chi := \frac{k-1}{2\epsilon}\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}} z_{1-\delta}\}^3$
        end if
    end if
end while
return χ_t

The issue might be able to be solved by the EM algorithmm mentioned in Textbook [1], which can estimate the unknown parameters more efficiently.

In the questions of extra credit, the KLD-sampling method does substantially reduce the computaiton burden, but it requires more computation at the beginning of the algorithm.

7 References

1. Thrun, S., Burgard, W., and Fox, D. (2005). Probabilistic robotics. MIT press.
2. Reading map data: `http://robots.stanford.edu/cs226-04/classdata/class.c`
3. Resampling Wheel: `https://www.udacity.com/course/viewer\#!/c-cs373/l-48704330/e-48748082/m-48740082`