Lab3 - Particle Filter

Probabilistic Robotics

Songyou Peng
psy920710@gmail.com

April 2nd , 2016

1 Introduction

Particle filter can be used as the localization of robotics, which is also known as Monte Carlo localization. As we know, it is successfully applied to low-dimensional estimation problems, for example in environments with known maps [1]. In this report, we will discuss the implementation details of particle filter localization. Throughout the discussion, some observations about the particle filter and problems we faced will also be mentioned.

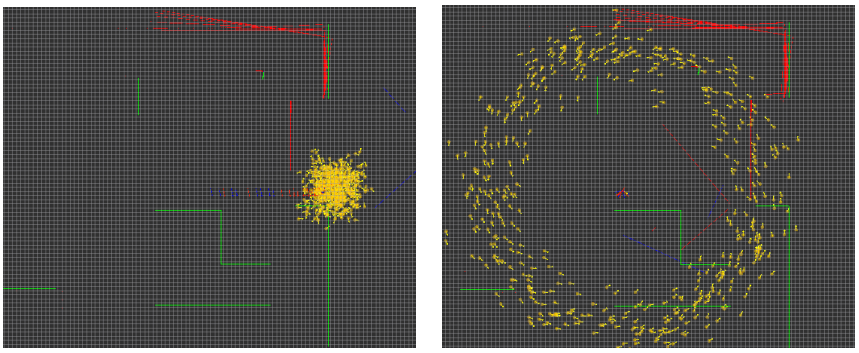
2 Implementation

The algorithm uses a bunch of particles to represent the distribution of likely states, with each particle representing a possible state, i.e. a hypothesis of the location of the robot [2]. In order to localize the robot, we need three steps: prediction, weighting, and resampling.

2.1 Prediction

In every iteration, we first predict the position `p_xy` and angle `p_ang` of every particle, both of which are in the world frame. What we do is to move particles based on an odometry difference $(\Delta x, \Delta y, \Delta\theta)$ between two consecutive measurements in the vehicle frame.

For the angle, we can simply add $\Delta\theta$ to the current `p_ang` because the change of angles is the same in both vehicle and world frame. However, as for the position, if we just directly add the $(\Delta x, \Delta y)$ to `p_xy`, all the particle will just move to the right as shown in Fig. 1(a). Because the $(\Delta x, \Delta y)$ is the movement under the vehicle frame and the robot only moves in its x direction in the frame, all the particle will only move to the right, which is the x direction in the world coordinate. We should transform the $(\Delta x, \Delta y)$ to the world coordinate according to the `p_ang` before we add it to `p_xy`. This was the first problem we faced. Meanwhile, in order to simulate the uncertainty of the prediction, some Gaussian noise should be added to `p_xy` and `p_ang`. The final correct prediction result is shown in Fig. 1(b) .



(a) Wrong prediction (b) Correct prediction

Figure 1: Illustrations for the results of prediction

2.2 Weighting

After predicting the position and angle of every particle, we need to calculate the weights for them. If the particle is close to the real odometry of the robot, high weight should be assigned, vice versa.

The first step of our implementation is using function *get_polar_line* to acquire the range and angle of lines obtained from the measurements and all the lines in given map. It should be noted that, for the sake of comparison, we should transform the given map lines from world frame to the vehicle frame because the measured lines are already in the vehicle frame.

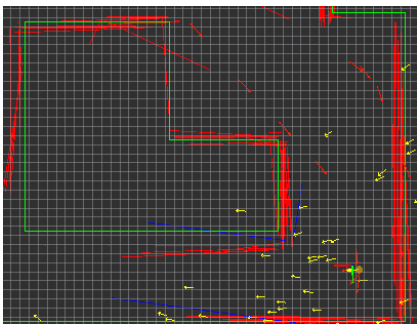


Figure 2: Weighting without resampling

Weights can be computed with a Gaussian. The weight of a measured line and a map line is the multiplication of the weights with respect to both range and angle. Therefore, once we acquire the weights for all the map lines, the weights for the current particle is the maximum value among them. The more similar ranges and angles between measured lines and given map lines, the higher weights particles have. Fig. 2 is the zooming-in version of the map. What

we can notice is that only one particle has large weight, with all others having really small weights.

if the robot senses 2 or 3 lines instead of just 1, we need to multiply the maximum weights of them instead of only using the highest. For example, if the three line segments are ranged like this _ _ _, the weight will not change much if the current particle is close to real position of the robot. However, if the lines are like _ | _, then if we only choose the highest, the weight may be wrong. Multiplication is a way to utilize all the information of the measurements.

As for the optional part, in order to deal with that special situation wrongly assigning weights, our solution is: we calculate the lengths of the current measured lines and compare them with the lengths of all lines in the given map. If the length of current measured lines is larger than a certain map line, we directly set the corresponding weight to 0. After using this method, we notice that that only particle with the significant weight can be obtained a bit faster.

2.3 Resampling

In weighting part, we found out that only one particle has a large weight. However, what we want is to refocus the particle set to regions in state space with high weights [3]. The more particles in a position, the more likely the robot is there. This is the resampling.

The resampling method we apply is not the independent random sampler, which randomly picks particles in the sampling space again. What we choose is "low variance sampler". The implementation detail is: we randomly pick a number r in the range of $[0, \frac{1}{M}]$, where M is the total number of particles. In every step, we add $\frac{1}{M}$ to r and check which part of sample weighting space the r is fallen in and assign the particle to this region. Finally, we have a new list of particles with the high-weight particles replicated and the low-weight ones removed. The result is shown in Fig. 3.

The main reasons for choosing this method is described as follows:

1. Cover samples more systematically. If the weight of a particle is larger than $\frac{1}{M}$, it will be at least sampled for once, which is reasonable.
2. When all the samples have the same weight, the sample space will not change. This can relieve the disadvantage of the loss of diversity.
3. Time complexity is $O(M)$, which is faster than the independent random sampler.

It should also be noted that we only resample the particle when the weight space has high variance, otherwise the resampling procedure may lead to a problem that some particles are replicated in unexpected place [4].

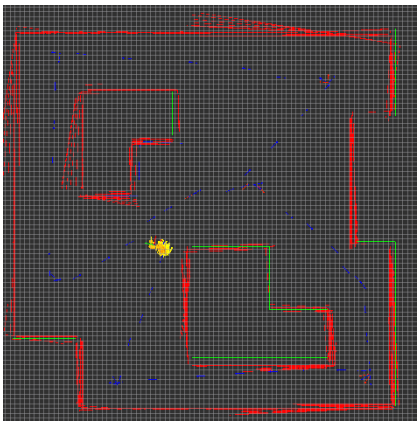


Figure 3: Resampling result

One last thing that need to discuss is: due to the fact that the initialization of the particles is random, every time we get different results. Most of time the results are correct, while sometimes it may lead to a totally wrong localization. We think it is also because initial particles don't spread out in the whole map, which kind of increases the risk leading to a wrong direction. When we increase the number of particles to 1000 (500 by default), the particle filter seldom fails but the computational time increases, which is not really suitable for real-time applications. Therefore, it is needed a trade-off.

3 Conclusions

In this lab we implement the particle filter to localize the robotics by iteratively predicting the states, weighting the particles and resampling them. We also compare the length of measured lines with lines in the given map in order to deal with some wrong weighting situations.

Meanwhile, we find out that the algorithm is not quite stable, depending on the number of particles and their initial status.

References

[1] Thrun, Sebastian. "Particle filters in robotics." In Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence, pp. 511-518. Morgan Kaufmann Publishers Inc., 2002.

[2] Wikipedia, The Free Encyclopedia, s.v. "Monte Carlo localization" (accessed April 2, 2016), https://en.wikipedia.org/wiki/Monte_Carlo_localization

[3] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. Probabilistic robotics. MIT press, 2005.

[4] Drew Bagnell,"Particle Filters: The Good, The Bad, The Ugly." School of Computer Science, Carnegie Mellon University. September 2012. Accessed April 2, 2016. http://www.cs.cmu.edu/~16831-f14/notes/F11/16831_lecture04_tianyul.pdf