Robot Localization Based on Particle Filter

Cesar Alan Contreras
Zongyuan ZHANG
Yongle HUANG
Runlong YANG

Yang LIU

Computer science school
University of Birmingham

I. INTRODUCTION

In the case of robotics and artificial intelligence, the first and most important task is to determine where the mobile robot is, in other words, is the 'Localization'. There are several methods for mobile robot localization, such as Kalman filter, Grid-based Markov Localization, and the Monte Carlo Localization (Particle filter). Understanding and analyzing the advantages and disadvantages of these methods is important for the developer to make an optimal choice over different cases (Gutmann and Fox, 2002). The most popular Localization method recently - Monte Carlo method or Particle filtering makes an optimal trade-off between the accuracy and robustness. The particle filter describes the probabilistic distribution based on sampling method. It samples the map according to the prior or the probability that are known, then give every particle a weight number as the symbol of the probability. After the iterations, the estimate pose of the robot will be converged accurately (Marchetti et al., 2006).

The basic framework of our particle filter algorithm is shown in Fig. 1.1. In our algorithm, we first scatter particles according to Gaussian Distribution around the initial given pose, which is the Initialization step. Then through further movement and observation (with noise), we update every particle's weight. Then we take out the 5% of the particles with lowest weight and spread them randomly into the map again, which is the Particle Cloud Pre-Selection step. We resample the remaining 95% particles so that the particles will converge around the ones with higher weight. Finally, in every iteration, we compute the estimate pose of robot by taking the mean value of the cluster particles' poses. We will demonstrate more details of each step in the following chapters.

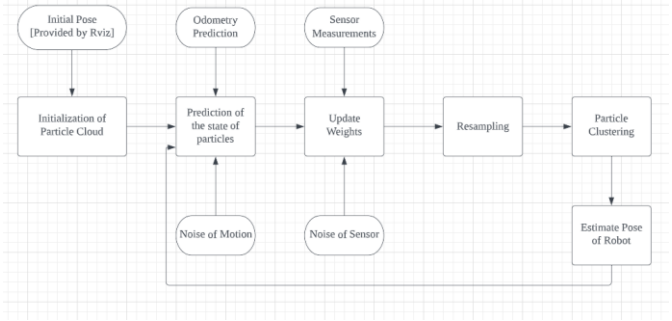

Fig. 1.1. workflow of particle filter

II. INITIALIZATION

There are several ways to initialize the particle cloud.

The first approach to do initialization is spreading particles randomly throughout the map. When the exact particle we are spreading are outside the map, we re-spread it randomly again until we have enough particles in the white space (inside the map).

As shown in the Fig. 2.1. below, when the total number of particles are large enough, there should be basically even particles in every area of the map. It is irrelevant to the initial given pose.

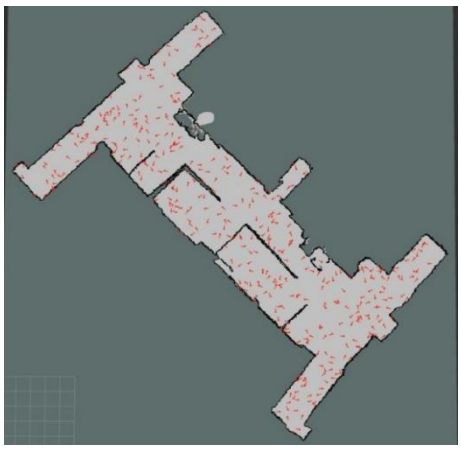

Fig. 2.1. result of spreading particles randomly

The second approach is putting a certain number of static particles in certain position in the map like the one in Fig. 2.2. When initializing, we don't need to compute the spreading of particles and it will make sure that there is even particles in every area in the map perceiving. It will help the particles converge to right position faster.

The disadvantage of this approach is that we have to adjust the static particles every time the map is changed. Even with small changes (e.g., a movement of one single table), that might block the static particles and influence the algorithm. We would like our algorithm to be robust and adaptive, so we didn't choose this approach.

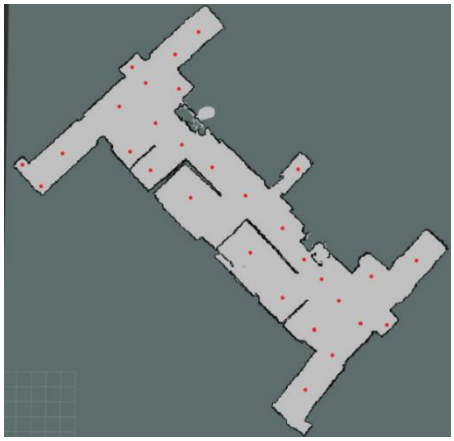

Fig. 2.2. result of spreading static particles

The third approach is spreading particles around the initial given pose (in this case, the initial pose is given by mouse clicking) according to Gaussian Distribution. As shown in the Fig. 2.3, This approach gives us a prior knowledge. When the initial pose is quite near the one and only true position of robot, the particles will converge faster to the true position because there were already sufficient amount of particles around the true position. But when the initial pose is far away from the true position, it will still work, but will slow the process down and take more time to make the particles to converge in the right position. Because compared to the first approach mentioned above, when initial pose is far away, it has a less probability of existing particles around the true position.

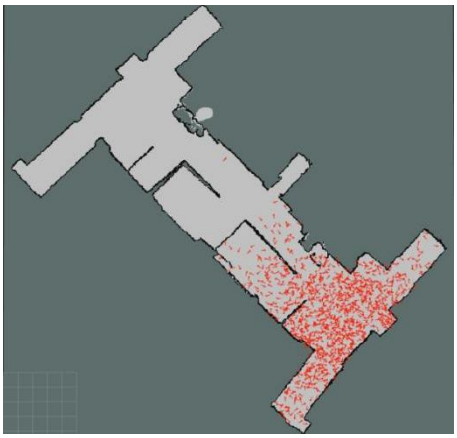

Fig. 2.3. result of spreading particles around initial position

We finally chose this initialization approach because we believe it is often the case that we would have a prior knowledge of the robot position in practical situation. For example, when doing auto-pilot car, it is often given a rough GPS position then continues to do a more precise localization or SLAM based on further vision measurement. In conclusion, it will help us to get a precise localization much faster when we have a roughly correct initial position.

When using this approach to spread our particles, there could be cases that our particle will fall outside the map, so we add a condition judgement to judge whether the particle is in the white space (inside the map), if not, spread the particles randomly inside any place in the map until we have enough number of particles in the map.

III. ROBOT'S ESTIMATE POSE

A. Background

When the particle filter algorithm finished updating the particle could, the robot localization algorithm needs to calculate the robot's estimated pose based on the newest particle cloud, so that the algorithm can get the next odometry value. The next odometry value is used for updating the particle cloud to the next stage and making the robot location algorithm a sustainable closed loop. So it is important to get a precise robot estimated pose in order to update the particle cloud correctly.

B. Basic logic for calculating the estimated pose

In the particle filter, each particle is a possible location of the robot. In the resampling step, we copy more particles around the higher-weight particles and delete some of the lower-weight particles. So, after the resampling step, more particle means higher weight. In the other words, the robot's pose is more likely in the place where the density of particles is higher. Since the algorithm only seeks one estimated pose, the basic logic of calculating the estimated pose is to find the location with the highest particle density.

An intuitive way to determine whether the particle density at a certain position is large is to calculate whether the standard deviation of the distance between each particle and a certain point is small. A simplified model can be used to demonstrate the relationship between the standard deviation of distance and particle density. Set several randomly distributed particles on an x-axis, then the standard deviation of the distance between one group of particles (x_n~x_m) and the origin is shown in the following equation:

$$\sigma = \sqrt{\frac{\sum_{i=n}^{m}(x_i - \bar{x})^2}{m-n}} \quad (1)$$

While $\bar{x}$ is the mean of abscissa of the selected group of particles. So, if every particle's abscissa in the selected group is close to $\bar{x}$, the σ will be small. Furthermore, in a two-dimensional particle cloud, the Euclidean distance between the particle and the origin can be used to obtain the standard deviation instead of the abscissa on the x-

axis. Then take the particle cloud as a sample, screen out a particle group with a smaller distance standard deviation, and then use this group of particles as a sample for the next screening, until the distance standard deviation of the last group of particles screened out is smaller than the manual-set threshold. Then it can be considered that the density of the last group of particles is large enough, and the average value of their coordinates and angles can be used as the estimated pose of the robot.

C. Implementing by code

The main logic of the code is a loop function. In each loop, the program calculates the standard deviation of distances of the particle groups filtered from the particle cloud, and this standard deviation will be compared with the threshold as the loop exit condition.

If the standard deviation of distance is higher than the threshold, the loop will continue. In this condition, the program uses whether the Euclidean distance between the particle and the origin falls in the interval $\left(\bar{d}-\sigma, \bar{d}+\sigma\right)$ as a filter condition, which σ is the distance standard deviation of the last filtered particles set and $\bar{d}$ is the average distance, filters out a new group of particles as the particle set used in the next loop.

Otherwise, if the standard deviation of distance is lower than the threshold, the program will exit the loop, and arithmetically average the x-coordinate and the y-coordinate of the last group of filtered particles to obtain the average coordinate as the estimated robot pose.

D. Summary

Estimating the position of the robot is actually to remove the sparser particles first, and then take the average coordinates of the remaining particles. Using the method of measuring the standard deviation of distances from the origin can cause problems when two groups of particles have similar densities and similar distances from the origin. This problem can be solved by considering the standard deviation of the distance from two points at the same time or considering the standard deviation of the x and y coordinates at the same time, but these solutions will lead to an increase in computing power demand.

IV. PARTICLE CLOUD REDISTRIBUTION

If the particle cloud is in a state of convergence, most of the low-weight particles scattered on the map will be removed through the resampling step. Since the particles in the particle filter represent the possible positions of the robot, all the particles converge in one place, which means that the estimated robot pose can only be in this cluster. If the robot has been moved to a new position (kidnapped robot problem) or the particles are not converging at the real robot's pose, the program will not be able to relocate the estimated robot pose to the correct pose and will fall into “self-lock”.

One solution is to remove some of the lowest-weighted particles (in our code 5% particles in the particle cloud) before the resampling step, then re-scatter them randomly on the map, and last resample the other particles. In the next loop, if the pose of one of the re-scattered particles is close to the real robot pose, this particle will have a higher weight so that the particle cloud will re-converge towards the correct robot pose during the next resampling step. Keeping a set of low-weight particles removed from each loop ensures that the particles always have the potential to converge somewhere else if the current cluster position is not correct.

V. SELECTION OF THE NUMBER OF PARTICLES

In a particle filter, the posterior distribution is approximated by particles, which means with the higher number of particles in a region, the probability is higher that the true state will be in that region. In other words, the higher the number of particles sampled, the better it reflects the true situation.

However, in practice, a larger number of particles means that the laptop must compute more particles, which leads to an increase in convergence time. Whereas a smaller number of particles will make the calculation less, and the estimated accuracy will be reduced. When the estimated accuracy is reduced to a certain level, the particles will not converge. So it's a trade-off to choose an appropriate number of particles in the particle filter.

In the selection of the appropriate number of particles for this assignment, we choose the convergence time as the standard for selecting the appropriate number of particles. The selection strategy is to choose as many particles as possible within an acceptable range of convergence time to achieve greater estimated accuracy.

To choose the appropriate number of particles, we test different numbers of particles to obtain a convergence time. The test method is to move the robot along a specified path and record the convergence time. To eliminate the bias, we selected four paths that could cover the entire map, and the convergence times for different numbers of particles are shown in Tab.5.1.

Tab. 5.1. Convergence time for different numbers of particles

Number of particles	Convergence Time				Mean Convergence Time
	Path 1 (Bottom left to top right)	Path 2 (Top left to bottom right)	Path 3 (Top right to bottom left)	Path 4 (Bottom right to top left)	
1000	8.59s	8.19s	8.37s	8.57s	8.43s
800	8.19s	7.70s	7.88s	6.91s	7.67s
600	7.34s	6.16s	6.70s	6.56s	6.69s
400	3.57s	3.39s	3.71s	3.44s	3.5275s
200	2.62s	2.92s	2.86s	2.75s	2.7875s
100	4.48s	4.16s	4.28s	4.72s	4.41s

Comparing the mean convergence time for each different number of particles could be plotted as Fig.5.1.

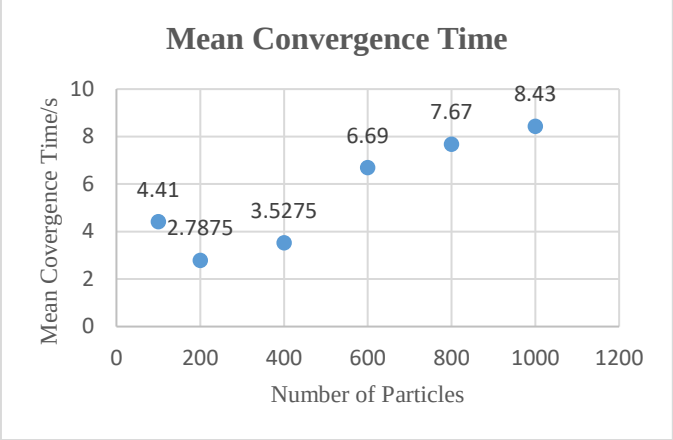

Fig.5.1. Mean Convergence Time for each different number of particles

As shown in Fig.5.1, we can easily find the minimum mean convergence time is 2.7875s, so we choose 200 as the number of particles used in our code.

VI. CONCLUSION

After several simulation runs, it can be verified that the robot localization program can quickly and accurately determine the robot's pose in a given map. Compared with AMCL model, AMCL model needs a roughly correct initial pose input to make the particle cloud converge, and if the particle cloud converges in a wrong pose, it will be impossible to be fixed. But in our algorithm, a correct initial pose input will only make the converge faster. It means that even if the initial pose is totally wrong, the particle cloud will eventually converge. In both cases, our program converges faster than the AMCL model. Besides that, our algorithm also uses particle cloud redistribution to solve the wrong converged pose problem and the kidnapped robot problem which AMCL model can't solve.

The complete program code is in the following Git hub link: *https://github.com/Cesar514/Particle-Localization/blob/main/pf_localisation/src/pf_localisation/pf.py*

REFERENCES

[1] Gutmann, J.-S. . and Fox, D. (2002) An experimental comparison of localization methods continued. IEEE/RSJ International Conference on Intelligent Robots and System, 454-459 (1). doi:10.1109/irds.2002.1041432.

[2] Marchetti, L., Grisetti, G. and Iocchi, L. (2006) A Comparative Analysis of Particle Filter Based Localization Methods. RoboCup 2006: Robot Soccer World Cup X, 4434: 442–449. doi:10.1007/978-3-540-74024-7_44.