Simultaneous Localization and Mapping (SLAM)

Assignment 2 - Particle Filter

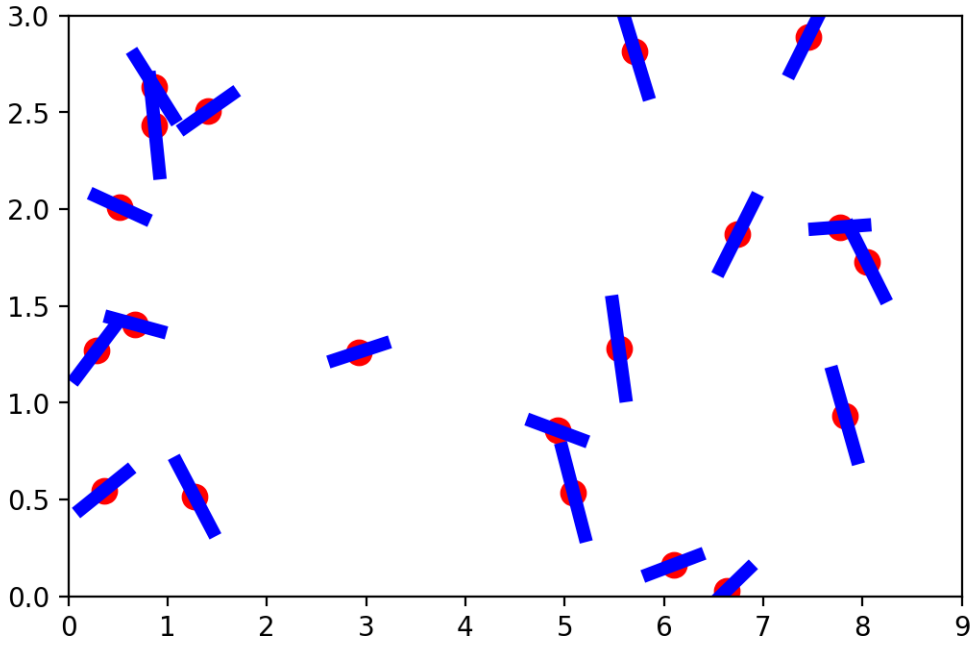
Name: 傅子豪(Fuhoward) Student ID: B06502018

I used Python to implement the assignment1 this time. I will show the performance of my Particle Filter, and I will also briefly illustrate the Particle Filter that I design and some findings in this assignment.

A. Environment:

Every time running the program, my program will randomly generate an environment with a given number of obstacles with random positon and orientation.

E.g. environment (9m * 3m) with 20 obstacles:
The red points are the centroid of the walls.



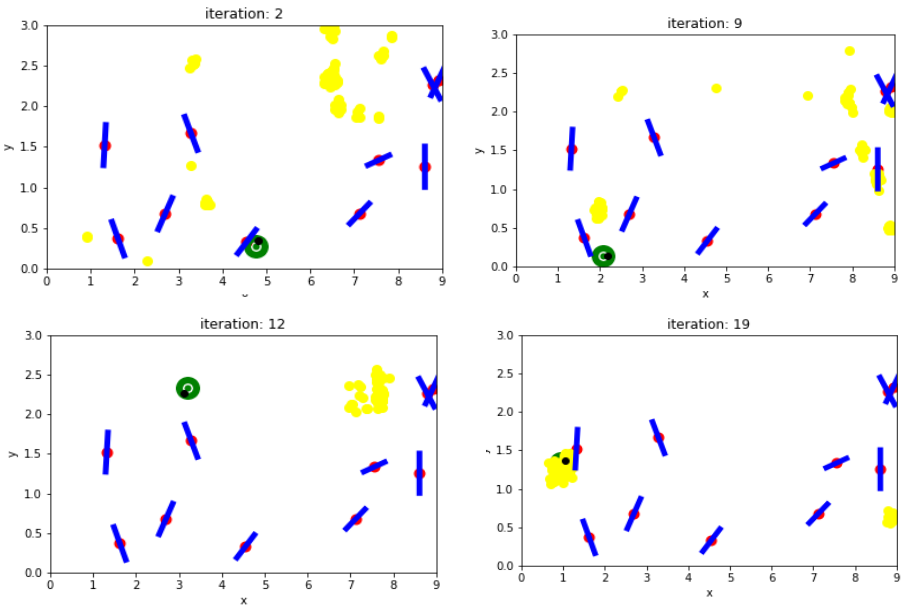
B. Program flow:

a. Generate particles in the environment
b. Robot will move based on the motion model
c. The particles will make the same movement as the robot
d. The robot and the particles will detect surrounding obstacle by sensor model
e. According to the sensing information, every particle will get a weight based on the correlation to the robot. With higher correlation, the particle weight will be higher.
f. Based on the weights, I will resample the particles in the environment
g. Repeat step. b ~ f, to locate the real position of the robot

C. Performance of the particle filter algorithm in the environment for 10, 20, and 40 obstacles:

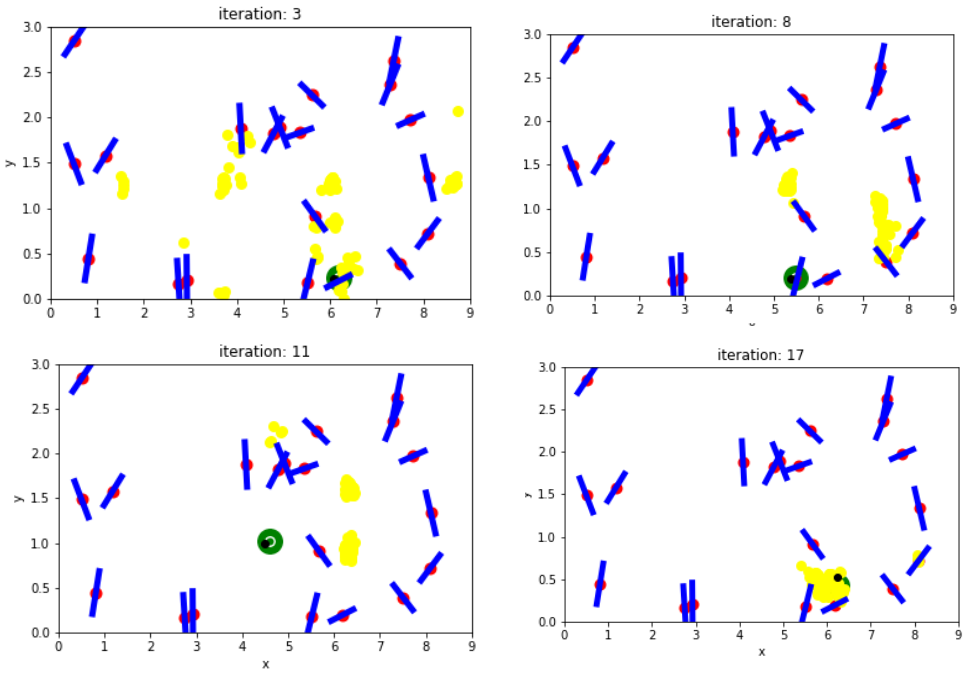
To show the performance of the particle filter for various number of obstacles, I apply 100 particles to the environment for 10, 20, and 40 obstacles.

a. 10 obstacles:
In the iterations, particle filter can correctly locate the position of the robot. However, in the beginning of the implementation, it is particularly hard for the particles to converge to the correct position of the robot, because the number of obstacle is too small that most of the time the sensor can't sense any obstacle in every iteration. Because of the lacking of sensing information, the particle filter can't work well in the environment with few obstacles.



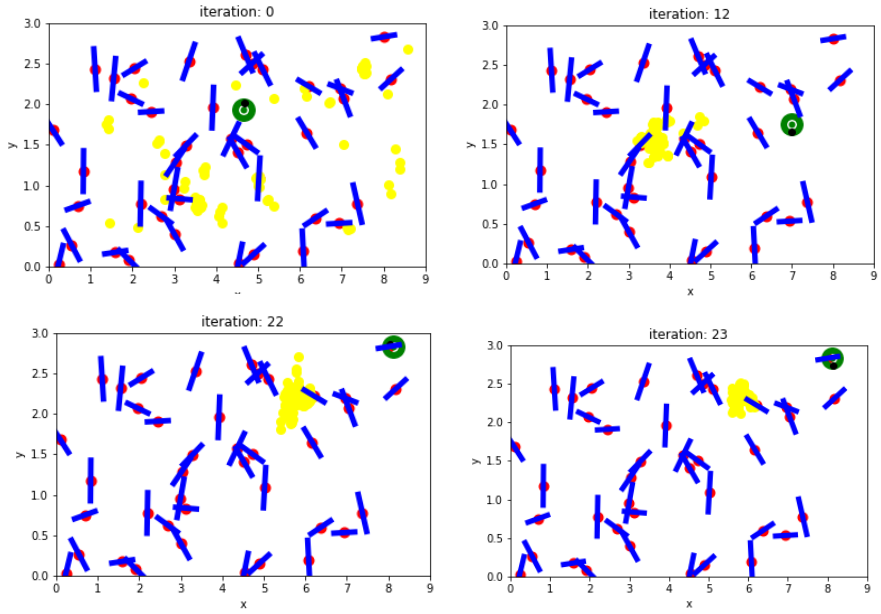
b. 20 obstacles:

In the iterations, particle filter can correctly locate the position of the robot. Moreover, it is easier for the particle filter to converge in the environment of 20 obstacles than in the environment with only 10 obstacles. I think it is because with more obstacles, the probability of sensing for particles and robot is higher.



c. 40 obstacles:

Before implementing, I think that the performance in the environment of 40 obstacles will be as well as the condition of 20 obstacles. However, I figure that it is actually harder for particles to converge. Although the particles eventually can converge to the position of the robot, its speed of convergence is lower. I think the reason is that the obstacles locate densely. As the result, it is more possible to sense the assignment of the obstacles which seems similar to the robot sensing.



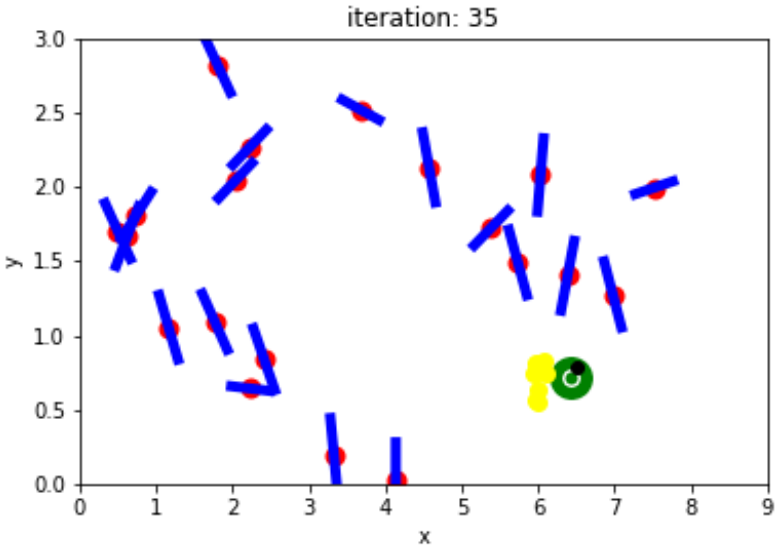
D. Performance of particle filter algorithm using 10, 100, and 1000 particles

To clearly show the performance of the particle filter based on different number of particles. I will apply the algorithm to the environment of 20 obstacle with 10, 100, and 1000 particles.

a. 10 particles:

The speed of the converge will be slow, because the range of searching for the particles is smaller. Moreover, the particles are usually close to each others because the sample of the particles are small, so when resampling, the result will be dominate by some of the particles.

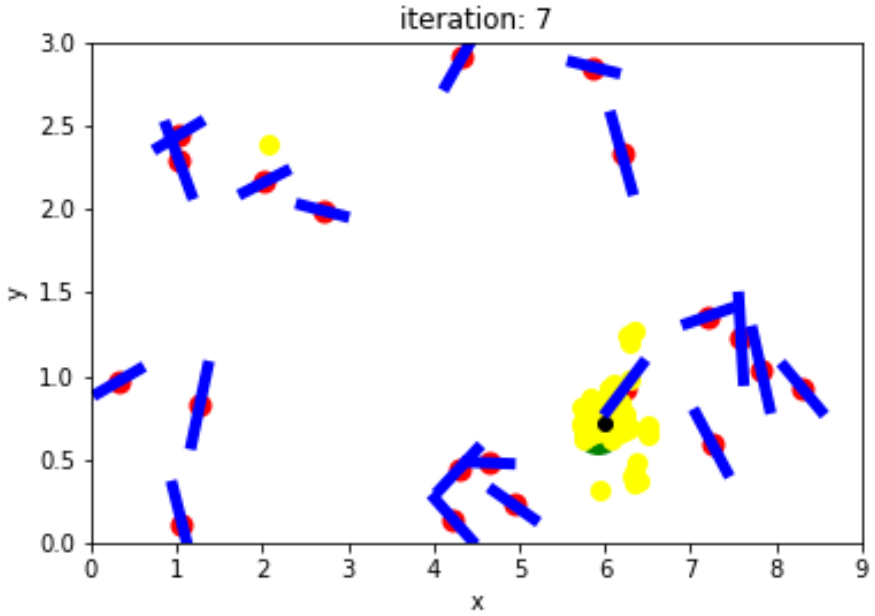
E.g. converge at iteration 35



b. 100 particles:

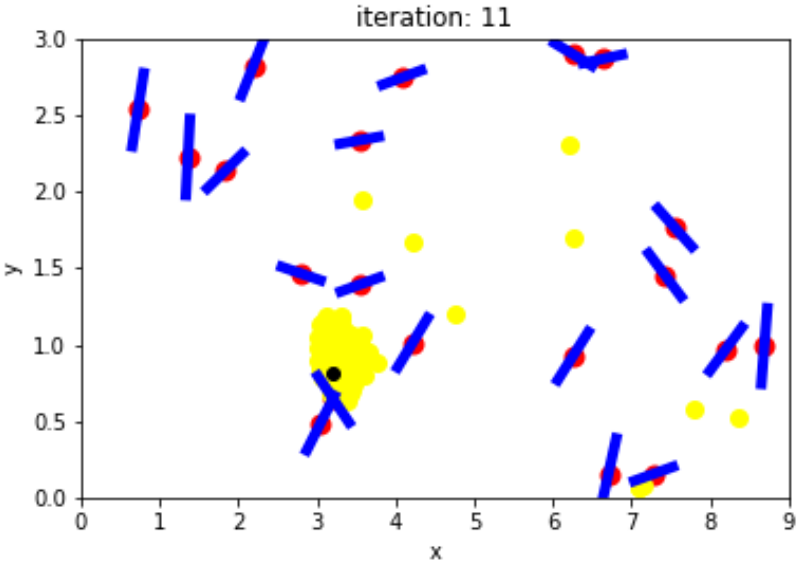
With 100 particles, the range of searching becomes bigger, so the speed of convergence gets faster.

E.g. converge at iteration 7



c. 1000 particles:

The speed of convergence with 1000 particles is as well as the speed with 100 particles. However, the searching range of the particles will be too large for this environment, so sometimes the precision of the localization will be reduced as well.



E. Discussion:

(1) Number of particles

After implementing particle filter by myself, I figure that it is extremely important to choose the right number of particles to different environment. I think that when the environment is larger, we should use more particles for localization at first, and when the convergence occurs, we should decrease the number of the particles to narrow the searching range for faster convergence.

As the experiment I had done above, if we use 10 particles, although it can localize the robot at the end, the searching range will not be enough to localize the robot in most of the time, and the speed of convergence will be slow as well. On the other hand, when we use 1000 particles, it can converge fast enough to localize the robot. However, due to the large searching range, sometimes it will generate several convergences at the same time.
As the result, I think the better way to implement particle filter might be choosing a large enough number of particles at first, and gradually reduce the number of particles to optimize the localization.

(2) Weight function:

I think that the hardest part in this assignment is that I have to design a proper weight function by myself, in order to optimize the performance of the particle filter. After many times of try and error, I first designed my weight function as shown below:

$$\begin{aligned} &Total\ weight \\ &\quad = 100 * distance\ weight + relative\ angle\ weight \\ &\quad + relative\ orientation\ weight \end{aligned}$$

$$distance\ weight = \frac{1}{\sum abs\big((robot\ distance\ to\ wall) - (particle\ distance\ to\ wall)\big)}$$

$$relative\ angle\ weight = \frac{1}{\sum abs((robot\ angle\ to\ wall) - (particle\ angle\ to\ wall))}$$

$$relative\ \ orientation\ \ weight = \frac{1}{\sum abs((robot\ \ orientation\ \ to\ \ wall) - (particle\ \ orientation\ \ to\ \ wall))}$$

However, although the particle can converge, I think the performance was not good enough. So, I add a penalty rules to the total weight as shown below:

(1) If the # walls robot detected > # walls particle detected:

$$Total\ \ weight\ \ after\ \ penalty = total\ \ weight\ -\ (\#walls\ \ robot\ \ detected - \#\ \ walls\ \ particle\ \ detected) * 100$$

(2) If the # walls robot detected < # walls particle detected:

$$Total\ \ weight\ \ after\ \ penalty = total\ \ weight\ +\ (\#walls\ \ robot\ \ detected - \#\ \ walls\ \ particle\ \ detected) * 100$$

(3) If the # walls robot detected = # walls particle detected:

$$Total\ \ weight\ \ after\ \ penalty\ =\ total\ \ weight$$

In my algorithm design, I consider the distance and the # walls being detected dominates the weight function, and in the end the performance improved.