1 Particle filter (50 points)

(a) (5 points) Complete the code blank in the `sample_motion_model` function by implementing the odometry motion model and sampling from it. The function samples new particle positions based on the old positions, the odometry measurements δ_{rot1}, δ_{trans} and δ_{rot2} and the motion noise. The motion noise parameters are:

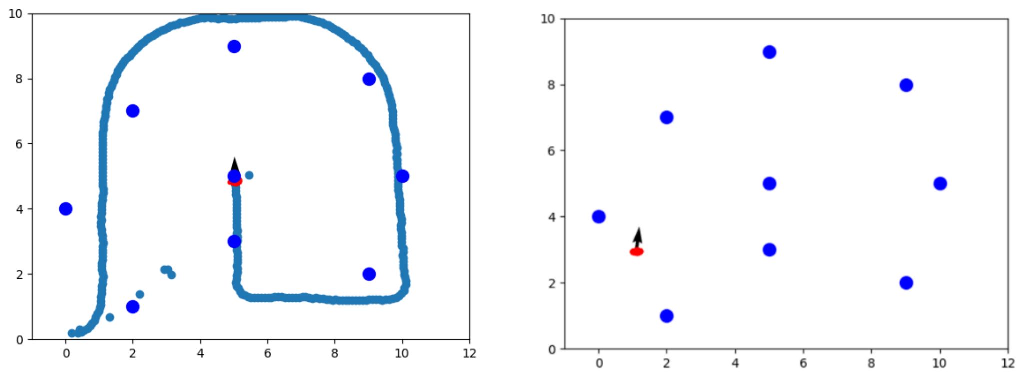
$$[\alpha_1, \alpha_2, \alpha_3, \alpha_4] = [0.1, 0.1, 0.05, 0.05.]$$

(b) (5 points) Complete the function `eval_sensor_model`. This function implements the measurement update step of a particle filter, using a *range-only* sensor, i.e., the sensor only measures the distance of the robot to the object but not the relative angle. It takes as input landmarks positions and landmark observations. It returns a list of weights for the particle set. See the particle filter lecture for the definition of the weight w. Instead of computing a probability, it is sufficient to compute the likelihood $p(z \mid x, l)$. The standard deviation of the Gaussian zero-mean measurement noise is $\sigma_r = 0.2$.

(c) (5 points) Complete the function `resample_particles` by implementing stochastic universal sampling. The function takes as an input a set of particles and the corresponding weights, and returns a sampled set of particles.

code is in the attachment

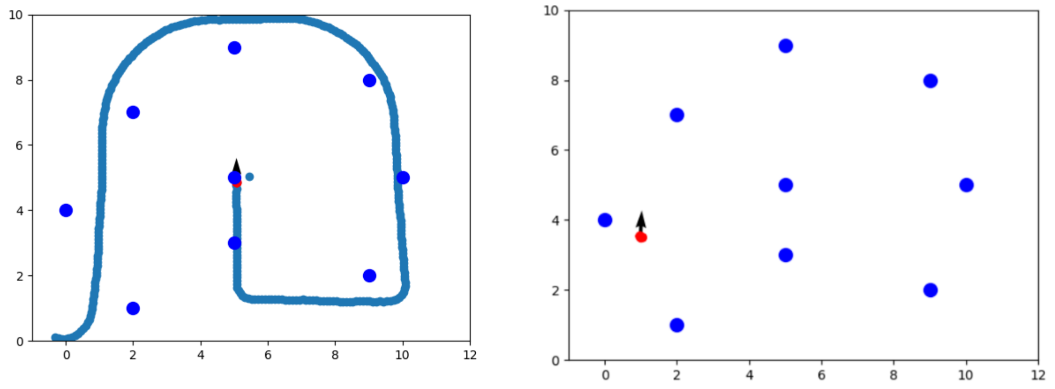
result:



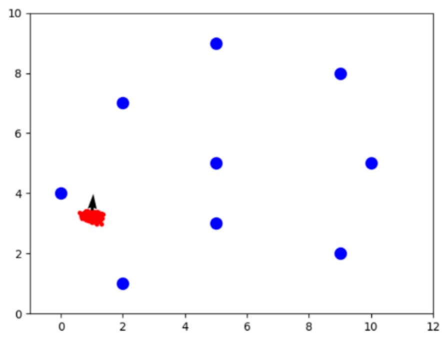
(d) (10 points) Add in the python code the part for taking into account the bearing measurement in the particle filter localization.

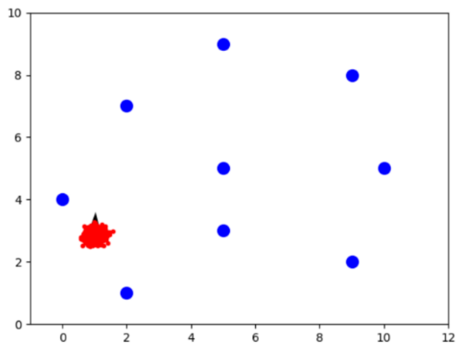
code is in the attachment

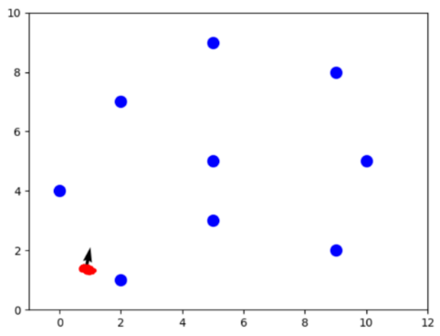
result:

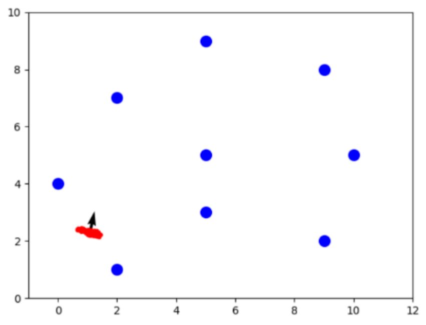


(e) (10 points) Change the noise level (i.e. the value of Q and R) and the number of samples used, discuss the performance of particle filter localization using different settings with some figures and numbers, e.g., how the estimation error and computational time change when these parameters vary.

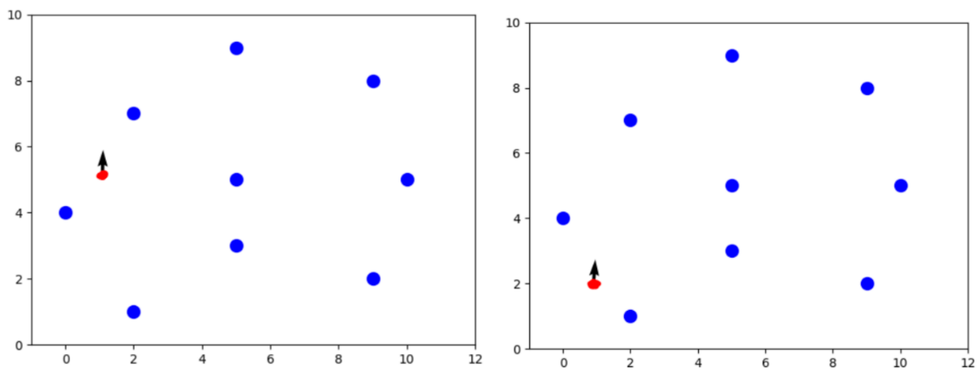

noise = [0.2, 0.2, 0.1, 0.1] sigma_r = 0.2


noise = [0.5, 0.5, 0.3, 0.3] sigma_r = 0.2


noise = [0.1, 0.1, 0.05, 0.05] sigma_r = 0.5


noise = [0.1, 0.1, 0.05, 0.05] sigma_r = 0.9

(f) (15 points) Read the KLD-sampling.pdf in the doc folder, which presents the KLD sampling algorithm that we briefly discussed in class. Implement the KLD sampling algorithm and compare its performance with the default re-sampling algorithm in (c).



Because using KLD can decrease the sample number in each step, the calculation speed is faster than the oroginal algorithm. The figures above show the task process at the same time using two sample methods.

2 (50 points) Fast SLAM implementation

(a) (10 points) Complete the code blank in the `sample_motion_model` function by implementing the odometry motion model and sampling from it. The function updates the poses of the particles based on the old poses, the odometry measurements $\delta_{rot1}, \delta_{trans}, \delta_{rot2}$ and the motion noise. The motion noise parameters are:

$$[\alpha_1, \alpha_2, \alpha_3, \alpha_4] = [0.1, 0.1, 0.05, 0.05].$$

How is sampling from the motion model different from the standard particle filter for localization that you saw and implemented in Problem 1?

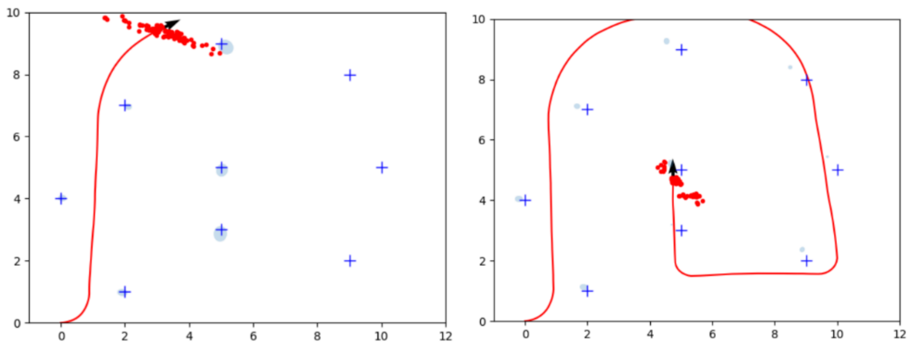
(b) (10 points) Complete the code blanks in the `eval_sensor_model` function. The function implements the measurement update of the Rao-Blackwellized particle filter, using range and bearing measurements. It takes the particles and landmark observations and updates the map of each particle and calculates its weight w. The noise of the sensor readings is given by a diagonal matrix

$$Q_t = \begin{pmatrix} 1.0 & 0 \\ 0 & 0.1 \end{pmatrix}. \tag{1}$$

How is the evaluation of the sensor model different from the standard particle filter for localization that you saw and implemented in Problem 1?

(c) (10 points) Complete the function `resample_particles` by implementing stochastic universal sampling. The function takes as an input a set of particles which carry their weights, and returns a sampled set of particles.

How does the resampling procedure differ from resampling in the standard particle filter for localization that you saw and implemented Problem 1?

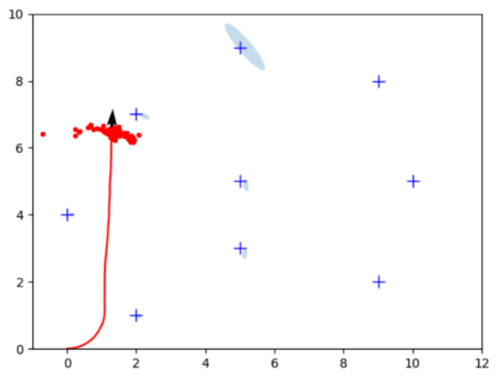


(a)

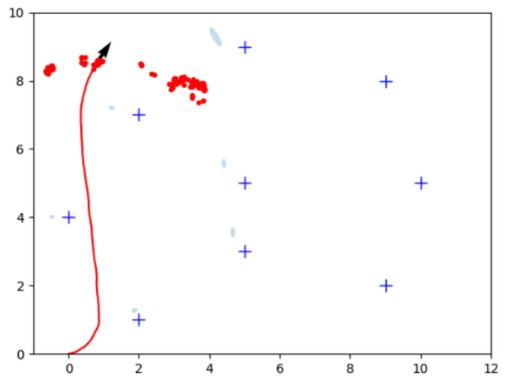
In PF, the function needs to return new particles.

In FastSLAM, the function needs to store the particles in history.
The algorithm is similar.
(b)
In PF, we already know all of the landmarks' ids and ranges.
In FastSLAM, we need to face to conditions: the landmark is observed for the first time or the landmark was observed before. It also needs to use measurement model to calculate the weights of each particle.
(c)
The algorithm is similar.

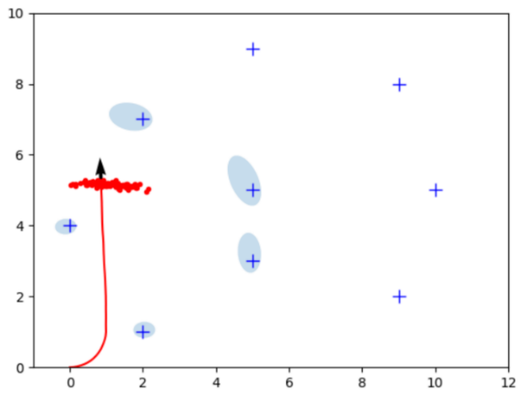
(d) (10 points) Try your implementation with different values for Q_t and $[\alpha_1, \alpha_2, \alpha_3, \alpha_4]$ and report your observations using figures, numbers, or analysis.



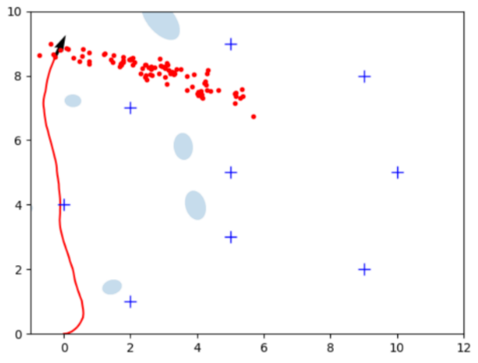
noise = [0.2, 0.2, 0.1, 0.1] Q_t = np.array([[0.1, 0],[0, 0.1]])



noise = [0.4, 0.4, 0.2, 0.2] Q_t = np.array([[0.1, 0],[0, 0.1]])

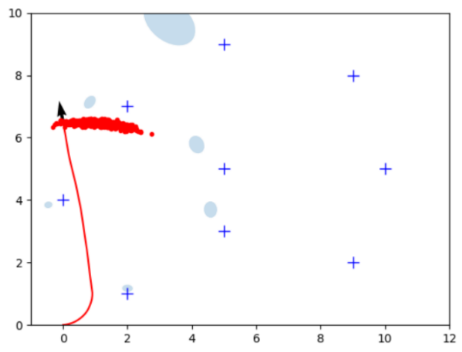


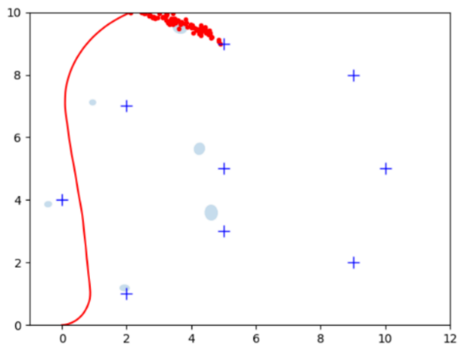
noise = [0.1, 0.1, 0.05, 0.05] Q_t = np.array([[2.0, 0],[0, 0.5]])

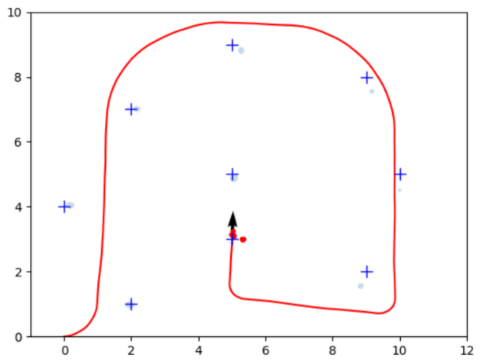


noise = [0.4, 0.4, 0.2, 0.2] Q_t = np.array([[2.0, 0],[0, 0.5]])

(e) (10 points) Try your implementation with different number of samples and report your observations about SLAM quality and computational cost using figures, numbers, or analysis.


sample=1000


sample=100


sample=10

The slam qualty is quite similar, if sample number is large, the result is more stable. But sample number influences the computional cost obviously.