[Robotique mobile]Particle Filter localization

yu-tianchi

January 2021

1 Introduction

In this practical work, we will study a Simultaneous Localization and Mapping (SLAM) method that builds a map of an unknown environment using an Extended Kalman Filter (EKF).

2 Influence of the environment

For this question, use the default parameters of the provided code. By default, the data association is assumed to be known, ie for each perceived landmark, the corresponding landmark in the map is identified without ambiguities.

2.1 Question 1

Modify the number and position of landmarks and the robot trajectory and explain what you observe (on the map quality and the evolution of errors, in particular around the time when the loops are closed) in the following situations : a short loop and a dense map with many landmarks inside the robot perception radius ; a long loop and a dense map with many landmarks all along the loop; a long loop and a sparse map with only few landmarks near the start position

Answer:

The trajectory is controlled by the controller v and ω, to make a small loop, we only need to change the radius of the loop.

1) a short loop and a dense map with many landmarks inside the robot perception radius.

Set the v in calc_input function as 1m/s and set the yaw_rate as 0.5rad/s, then we have the radius of the loop as 2m. The maximum sensing distance of the robot sensor is setting as 10.0 now, so we only need to make sure that there are many landmarks inside of the observation radius.

We can notice that, if we have enough landmarks inside of the perception radius reasonably, the estimated results have a good and regular performance. The error of x and y is limited. The curves of variances of x and y display as sin function, which is quite reasonable because of the circle trajectory. In the case we have, only the error of orientation has some obvious variations, while the variance is stable and small. So we can tell the EKF performs well with short loop.

2) a long loop and a dense map with many landmarks all along the loop

Set the v in calc_input function as 1m/s and set the yaw_rate as 0.1rad/s, then we have the radius of the loop as 10m. It can be taken as a long loop. Then we put many landmarks all along the loop.

Ideally, the variances of position of robot should also perform as sin/cos function, and the variance of orientation should be stable. However, at around the 300 step of estimation, there is a perturbation or divergence, and the errors of coordinates x and y increase apparently. The variance of orientation changes too. It is normal because during the long loop, the error of observation increases with the error of motion. Good thing is that it will come back when the end of the cycle starts to meet the beginning. The robot finds some landmarks met before(and it know the id), the variance becomes stable suddenly at around 400 step. And we set the running time double, we can see the strange divergence never happens in the rest time.

Note that the errors of positions displays as a regular periodical function.

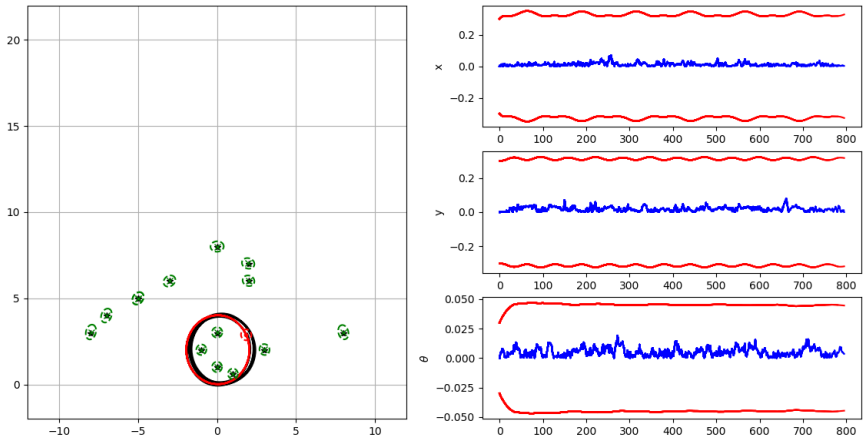


Figure 1: One short loop with a dense map

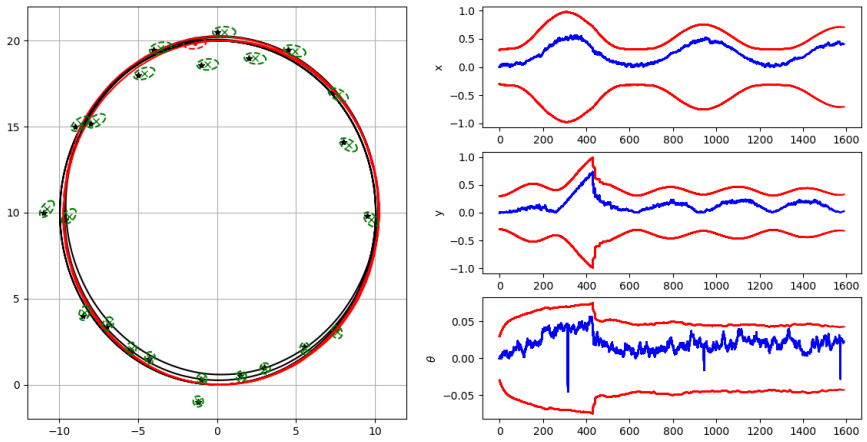


Figure 2: One long loop with a dense map(landmarks all along the loop)

3) a long loop and a sparse map with only few landmarks near the start position

Also, set the v in calc_input function as 1m/s and set the yaw_rate as 0.1rad/s, then we have the radius of the loop as 10m. It can be taken as a long loop. Then we put few landmarks near the start postions of the loop.

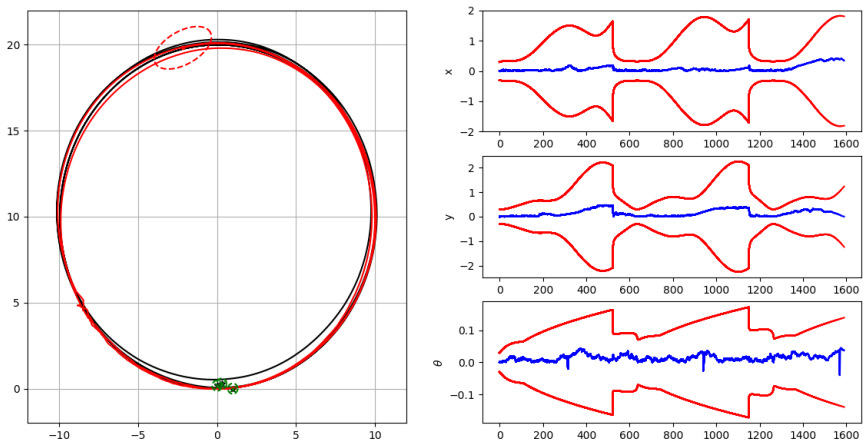


Figure 3: One long loop with a sparse map(only few landmarks near the start position)

Obviously, at the beginning the performance of estimated result is very nice. With the distance between the landmarks and robot growing, the errors seem to increase. Also the variances diverge, until the robot gets back to the radius of perception. These landmarks help robot corrects its fault every time when them meet, however, the trajectory seems to get worse bit by bit(the divergences of cycles seems to be larger than the precedent time).

2.2 Question 2

Answer the same question when the data association is performed using the Mahalanobis distance (KNOWN _DATA _ASSOCIATION = 0). You may have to tune the M_DIST_TH parameter depending on your environment.

1) a short loop and a dense map with many landmarks inside the robot perception radius.

Set the v in calc_input function as 1m/s and set the yaw_rate as 0.5rad/s, then we have the radius of the loop as 2m.

It seems to easy associate all the landmarks. And the performance is similar with the situation knowing landmarks id.

2) a long loop and a dense map with many landmarks all along the loop

Set the v in calc_input function as 1m/s and set the yaw_rate as 0.1rad/s, then we have the radius of the loop as 10m. It can be taken as a long loop.

One more step, here we change the threshold of Mahalanobis distance for data association from 9 to 6. And we can tell that, with a good data association, the robot can reduce the error and follow the true trajectory much better than the previous one.

3) a long loop and a sparse map with only few landmarks near the start position

Also, set the v in calc_input function as 1m/s and set the yaw_rate as 0.1rad/s, then we have the radius of the loop as 10m. It can be taken as a long loop.

It is similar with the last question, once the robot associates the landmarks at the starting position.

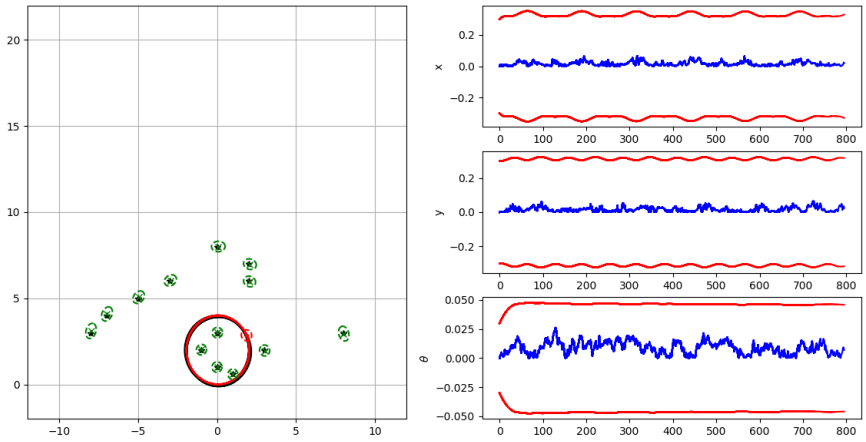


Figure 4: One short loop with a dense map

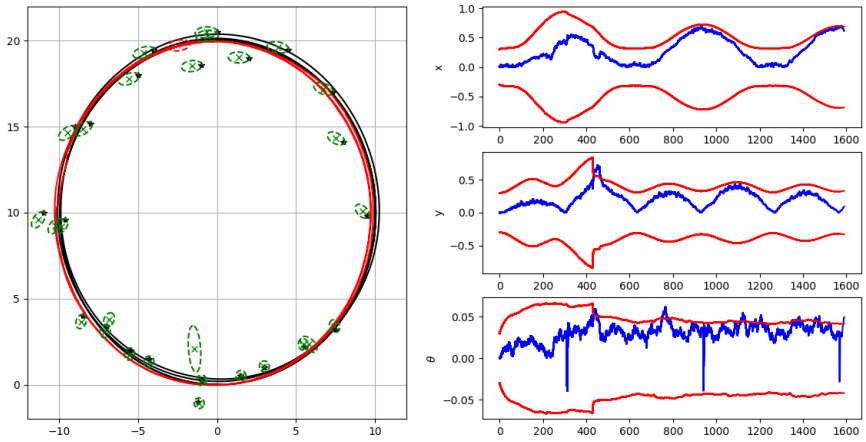


Figure 5: One long loop with a dense map(threshold of MD = 9)

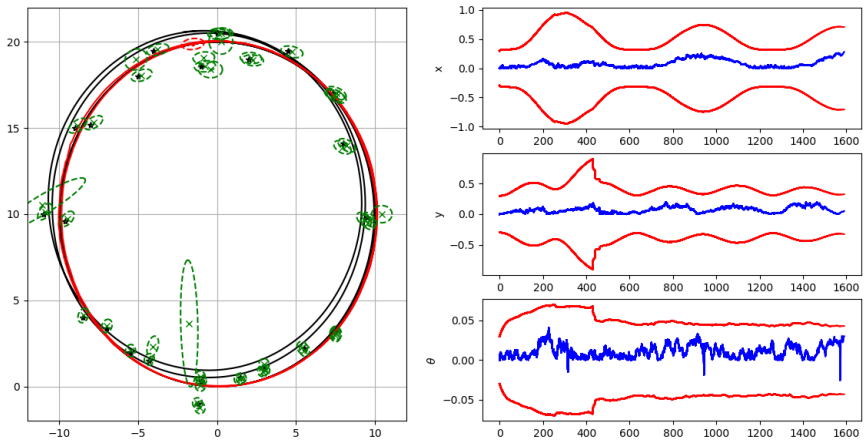


Figure 6: One long loop with a dense map(threshold of MD = 6)

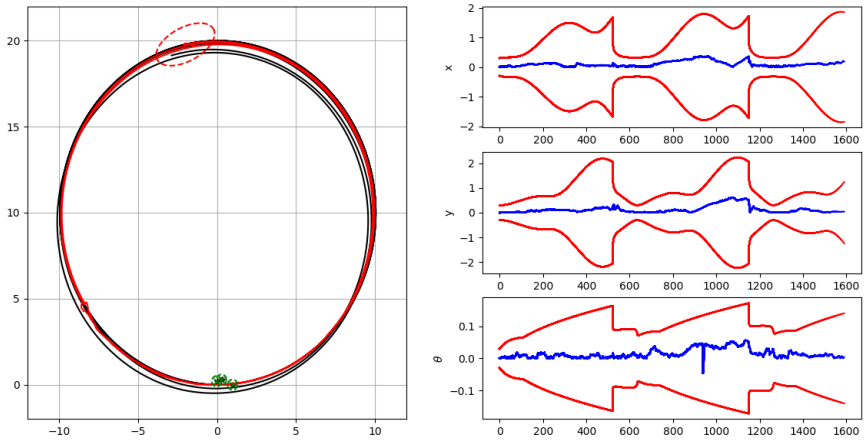


Figure 7: One long loop with a dense map(landmarks all along the loop)

3 Probabilistic models

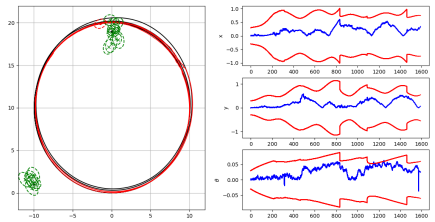
For the this question, keep the configuration with unknown data association (KNOWN_DATA_ASSOCIATION = 0) and an environment with a large loop and a sparse map

3.1 Question 3

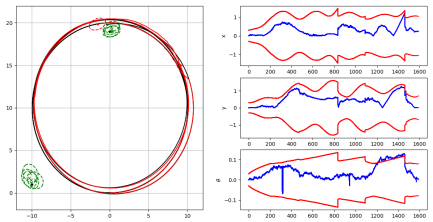
Change the estimated noise values Q and Py so that they are (1) smaller, (2) equal or (3) larger than the values used for simulation (Q_Sim and Py_Sim). What happens in each case for the filter performance, the filter consistency and the map quality? What seems to be the best configuration ?

Answer:

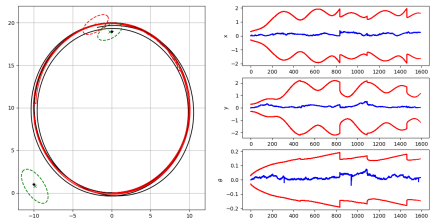
In this part, we keep the threshold of Mahalanobis distance as 9, and change the values of Q and Py.



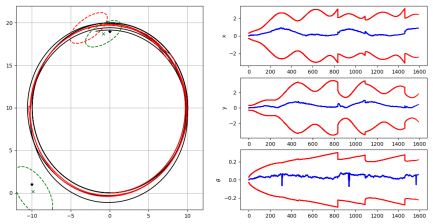
(a) pic1.smaller noise



(b) pic2.same noise



(c) pic2.(2 times)bigger noise



(d) pic2.(5 times)bigger noise

Figure 8: Long loop and sparse map with different noise

Actually, the performance is related the data association. With a large noise value(reasonably large), the landmarks data is easier to be associated, so the filter consistency is much better than the smaller and equal noise. The data association is getting better with the increase of noise value. Also, the errors of position and orientation are more stable. However, the bigger noise could cause one problem of losing reality. If the noise of motion and observation is too big, then we will have a really bad map quality and wrong trajectory. Because the big noise means that it is hard to know the real and precise position. Here is one example in the Fig.9. So we should know that, the performance could also be influenced by the noise itself.

In conclusion, we can take Q and Py as 2 or 3 times of the simulator's noise(with the same conditions and constraints).

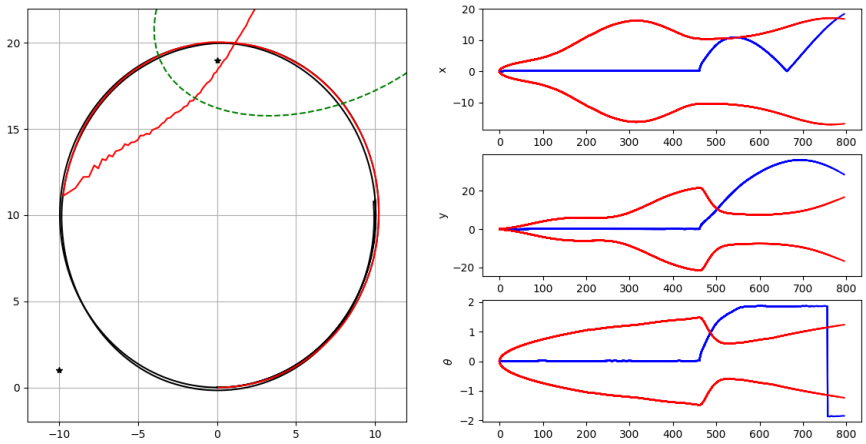


Figure 9: Long loop and sparse map with really big noise

4 Undelayed initialization

4.1 Question 4

We will now modify the code to perform bearing only SLAM (i.e. SLAM using only the direction of landmarks, not their distance).

Modify the code of the filter to use only the landmark direction in the Kalman correction. Implement a simple undelayed initialization for new landmarks by adding several landmarks along the perception direction with growing covariances.

Answer:

This question is implemented in the file of 'ekf_slam_undelayed_initialisation.py'.

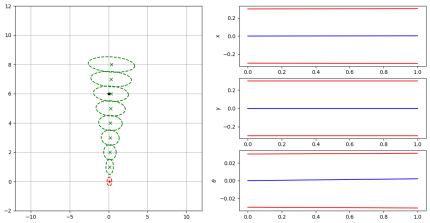
Keep the parameters as the best performance we have concluded in the previous questions, only change the running time as 20s, and change the size of map to make the performance clearly.

The purpose of this part is to calculate several possible positions of the same landmark, and then remove those incorrect. So we need to build a list of landmarks on the line when we know the direction of one landmarks. Here I set the initial possible landmarks as 4, and the distance in y direction between these landmarks is 1m(starting from the estimated position), then use the 'calc_landmark_position' function to calculate the position of landmarks. It should return all the positions of the landmarks we want(with the same direction).

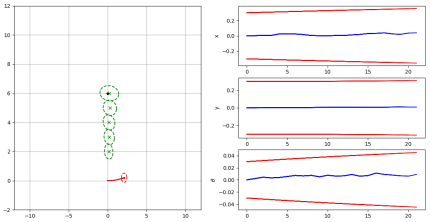
Because we don't need the distance between landmark and robot, we need to replace the detected distance by our new landmarks.

To append and remove the possible landmarks, we need to mark the states of the landmarks. Every step we calculate the frequency/weight of the landmarks by function of $\frac{e^{ld[current]}}{\sum e^{ld[i]}}$. Set the thresholds(0.01 in this case), we delete the landmarks with frequency below than the threshold.

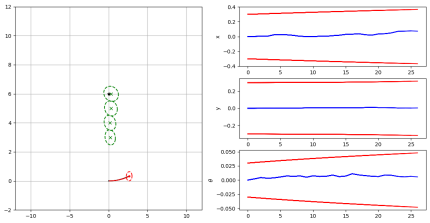
Here is the implementation of my example. (Step only one real landmark in the map)



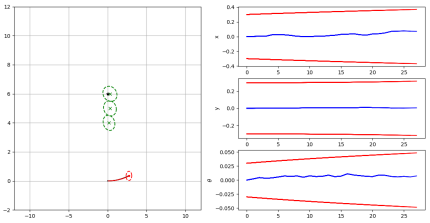
(a) pic1



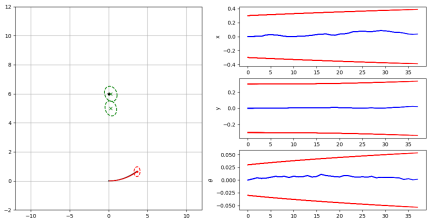
(b) pic2



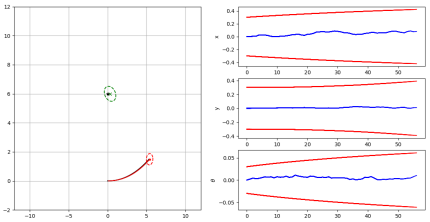
(c) pic3



(d) pic4



(e) pic5



(f) pic6

Figure 10: Step showing for the undelayed initialisation SLAM