Solution to analysis in Home Assignment 4

Lizi Teng + lizi(cid)

Analysis

In this report I will present my independent analysis of the questions related to home assignment 4. I have discussed the solution with Qun Zhang, but I swear that the analysis written here are my own.

1 Smoothing

a)

We can see directly from the figure 1 and figure 2 that the smoother has a better estimate than the CKF. The smoother estimate is less noise and fit the true trajectory a lot better. Also we can see that the smoother decrease the error covariances.

The smoother estimate fit the true trajectory better because the smoother implement the estimate with the measurement of all time i.e. $p(x_k|y_{1:K})$, but the filter implement the estimate with the measurement up to the current time i.e. $p(x_k|y_{1:k})$. And the covariances are decreased because of $P_{k|K} = P_{k|k} - G_k[P_{k+1|k} - P_{k+1|K}]G_k^T$, the smoother covariance $P_{k|K}$ is smaller than the filter covariance $P_{k|k}$.

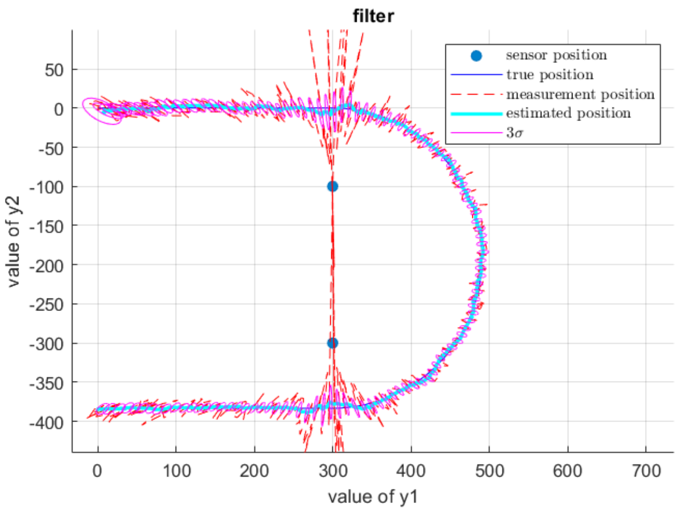


Figure 1

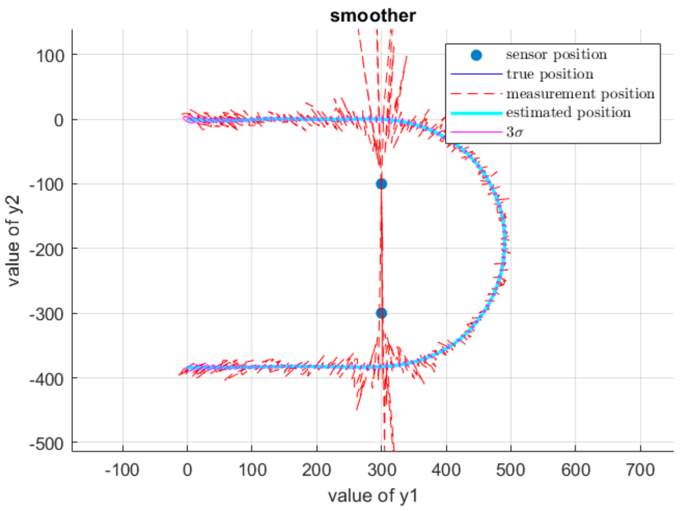
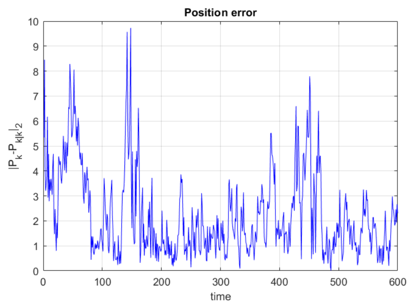
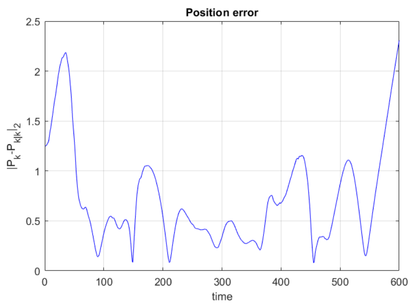


Figure 2

And the figure 3 shows that the smoother error is lower than the filter error generally. But they share the same error for the last estimate because for the last position $K = k$, the filter and smoother share the same estimate.



position error for filter



position error for smoother

Figure 3

b)

We introduce an outlier at one time instance k = 300, and the measurement with the outlier can be seen in figure 4.

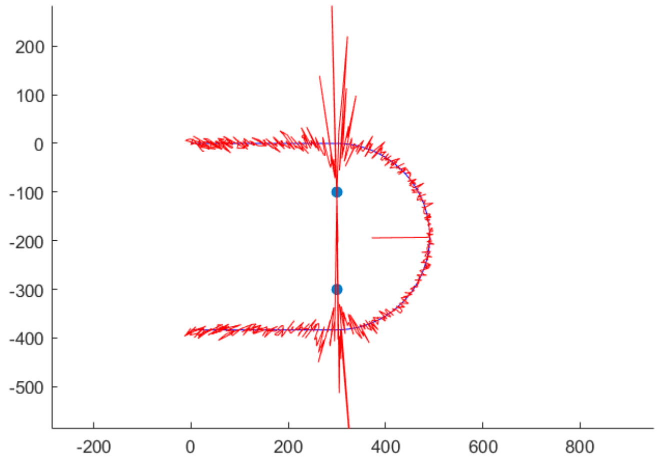


Figure 4: The measurement with outlier

As the figure 5 shows, the filter estimate has a small fluctuation that deviates from the true value when the outlier is introduced but the smoother

estimate is hardly affected by it. The smoother can handle the presence of an outlier better than the filter.
Both the estimate don't show too much error when the outlier happen, that is because the motion model is CT model, and would cause the estimate not to be influenced too much by erroneous measurements. And the smoother performs a lot better because it knows the information of the whole measurement and the motion model, and can smooth the outlier based on the future measurement.

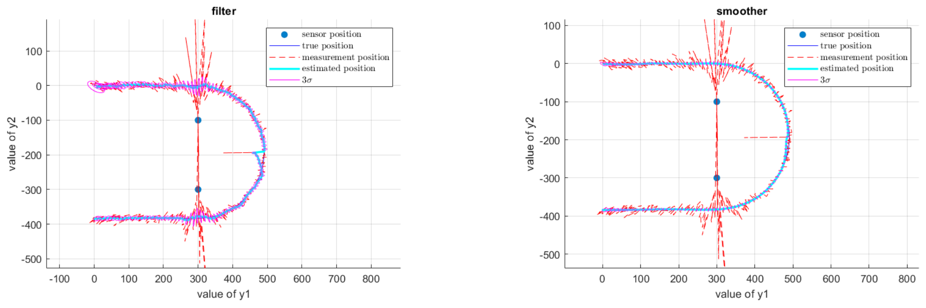


Figure 5

2 Particle filters for linear/Gaussian systems

a)

- Figure 6 shows the true trajectory and the different filter estimates over time. And with particle amount $N = 10000$, the PF without resampling doesn't perform very well, but the estimate of KF and PF with resampling almost coincide.
  We can see from the table below that the MSE changed with particle number for different filter. When the number of particles increase, the two PF perform better and gradually approach the MSE of KF. Also, the change of number has bigger influence to the performance of PF

without resampling.

	N = 25	N = 100	N = 5000	N = 10000
KF	1.3821	1.3821	1.3821	1.3821
PF-without sample	7.6872	3.5696	2.2221	1.4122
PF-with sample	1.5204	1.4180	1.3763	1.3811

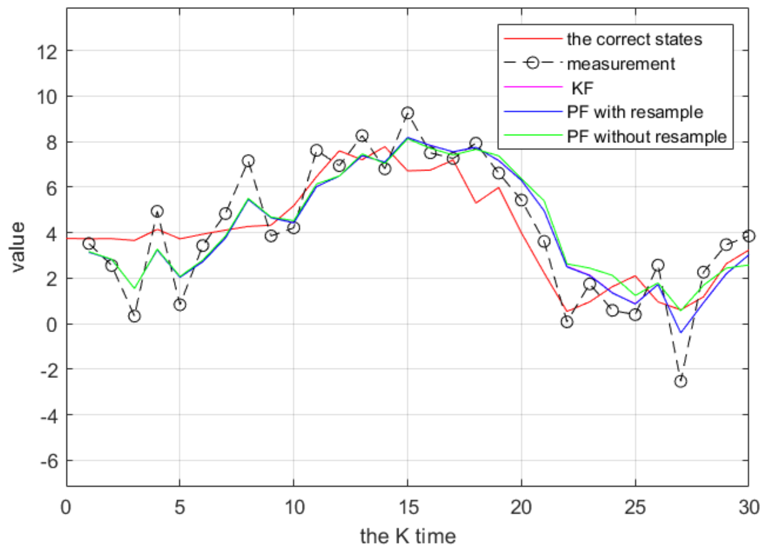


Figure 6

- The following figure shows the comparison of Errorbar with KF, PF with resampling and PF without resampling.
  And we can see that the PF without resampling doesn't perform as well as the other two.

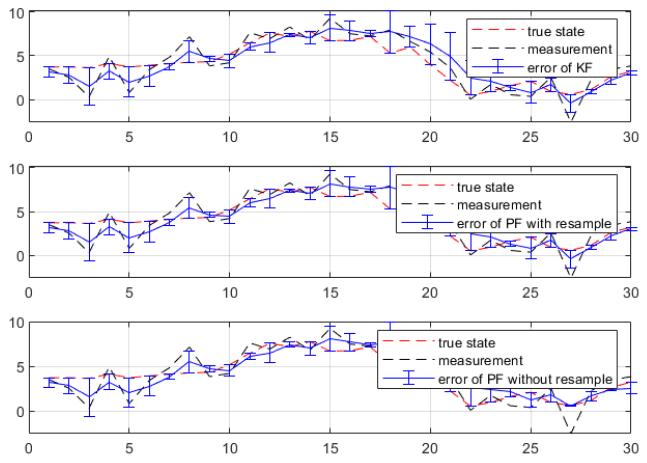


Figure 7

- In figure 8, we can see the posterior densities of those two PFs compared with KF. After tunning the, we get the kernal size as 0.5, 0.6, 0.5 for $k = 1$, $k = 15$, $k = 30$. And the densities of PFs are similar to KF.

  As we can see, the densities of PF with resampling have better approximation to the density of KF for $k = 15$ and $k = 30$.

  For the tuning, when the kernal is smaller we have sharper representation of each individual particles in the visualization. The particle locations and their contribution to the posterior density can be represent more detailed. And if the PF performs good, we can have distributions fit the KF distribution better.

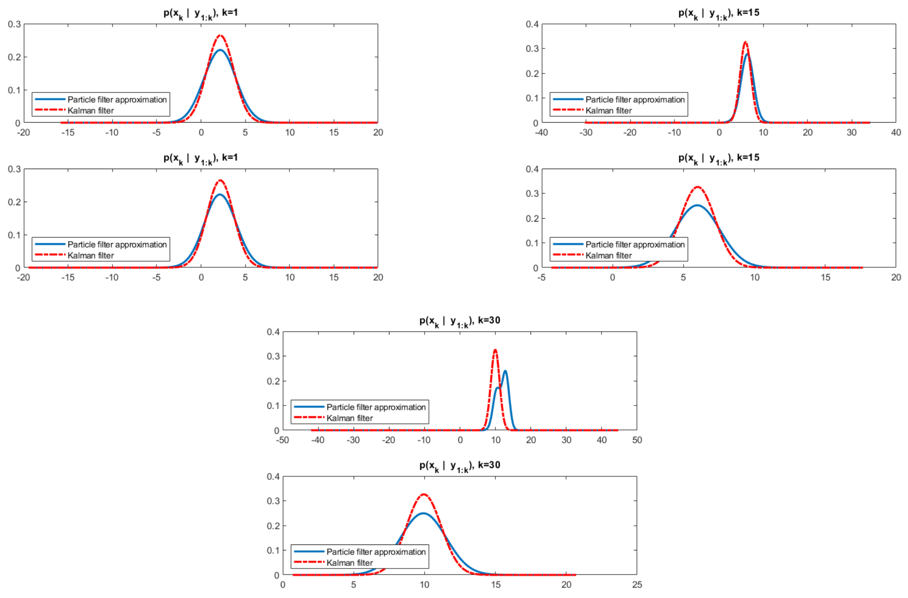


Figure 8: with all kernal size=1

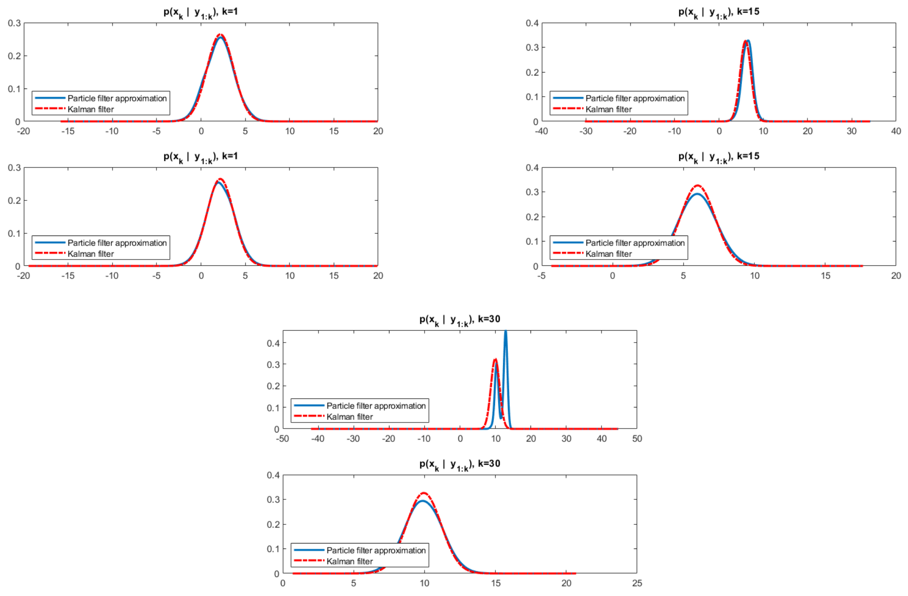


Figure 9: after tuning

b)

We can conclude from the following figure that the PF with resampling compared to the PF without resampling has a better performance with wrong prior. And the Kalman filter has the best performance among the three.
The Kalman filter doesn't relay on the initial prior too much, so it can approach the true state really fast. And the PF with resampling is a bit slower than KF, but it still approached because the resampling step allow accurate particles to propagate and survive, effectively mitigating the impact of an incorrect prior. And the PF without resampling can hardly approach the true states with the wrong prior, because the wrong particles that represent the wrong prior will remain.

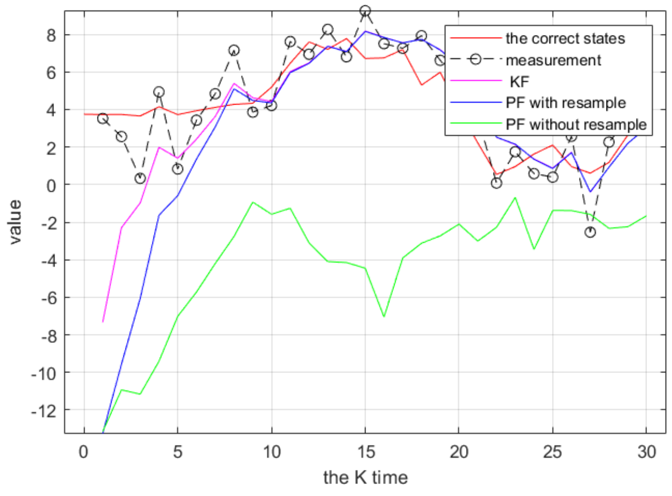
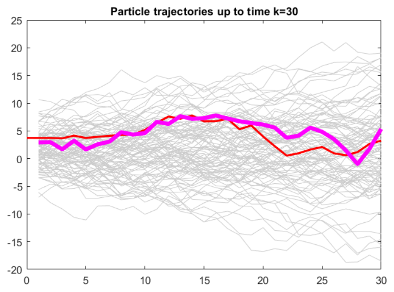


Figure 10

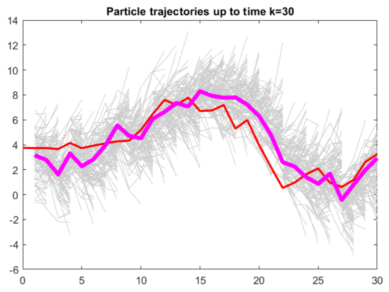
c)d)

From figure on left side we can see, the particles in PF without resampling tend to disperse because of the uncertainty with the trajectory of motion model. And lower weights will be assigned to particles that are far from the true state. That will cause that fewer particles can make influence to result

in the filter, and could undermine the performance of the filter.
The right one shows the performance of PF with resampling. During the resampling step, particles with lower weights are more likely to be replaced by particles with higher weights, and it means that the particles after resampling are more likely to be close to true states. Also the weights are set to be $\frac{1}{N}$ every iteration, so the particles that can contribute to the result are more. Those changes can improve the performance of PF.



The particles trajectory of PF without resampling



The particles trajectory of PF with resampling

Figure 11

3 Bicycle tracking in a village

a)b)

Firstly, I Generate a sequence of positions,X_k^p using $MapProblemGetPoint.m$. Here is the figure of trajectory.

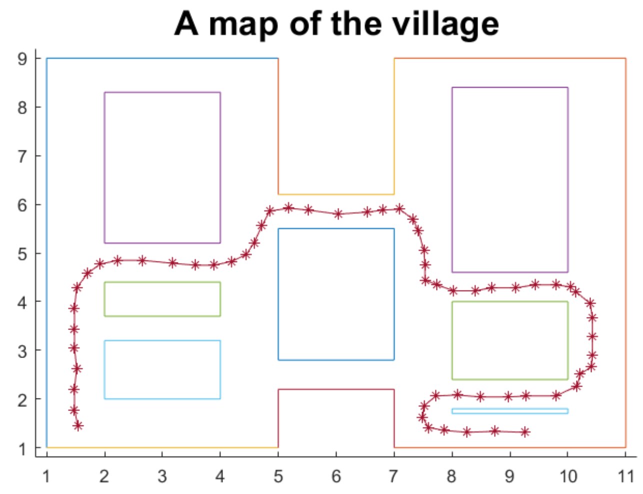


Figure 12: The measurement with outlier

Then with the measurement model, we get the measurement:

$$y_k = x_k^v + r_k$$

Where $x_k^v = x_k^p - x_{k-1}^p$,and I set σ_r=0.05.
Here is the figure of velocity measurement.

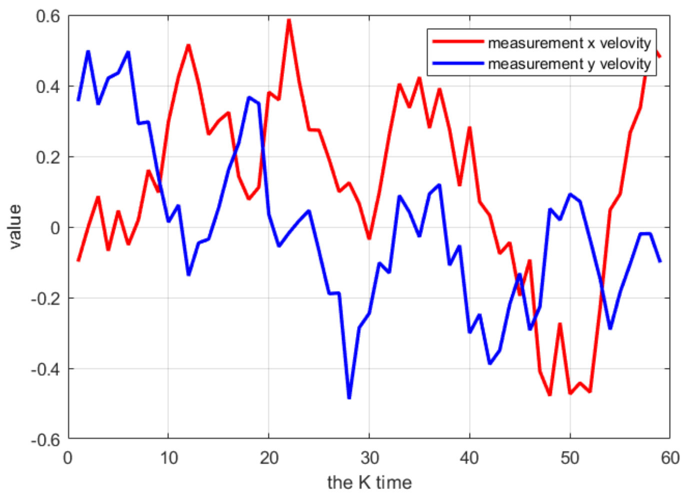


Figure 13: The measurement with outlier

c)

I try to use CV model as the motion model of this task. I set the state to be $[x, y, vx, vy]^T$ and the motion model would be :

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ vx_{k+1} \\ vy_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ vx_k \\ vy_k \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ qv \\ qv \end{bmatrix}$$

where $T = 1s$. And the measurement model is :

$$\begin{bmatrix} vx_k \\ vy_k \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ vx_k \\ vy_k \end{bmatrix} + \begin{bmatrix} rv \\ rv \end{bmatrix}$$

Now, if we want to view the map as a component in the motion model. The prediction step can be seen as :

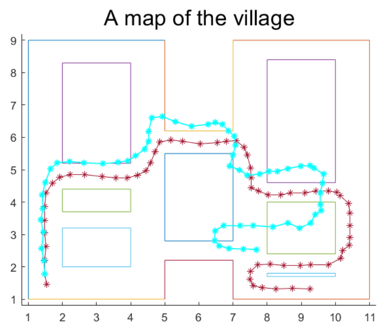
$$p(x_k|x_{k-1}, M) \propto p(M|x_k) * p(x_k|x_{k-1})$$

$$p(x_k|y_{1:k-1}, M) = \int p(x_k|x_{k-1}, M)p(x_{k-1}|y_{1:k-1})dx_{k-1}$$

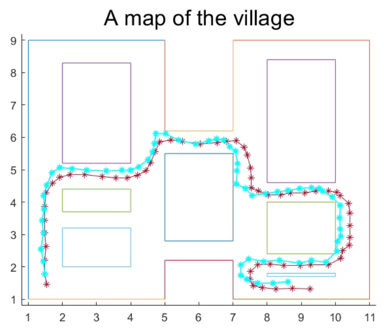
We can see that the linear transition of turns to a nonlinear one. With the particle filter, this change happens when we calculate the position of particles for next step.
And if we try to view the map as a measurement, we use the information as an additional measurement, we use the *isOnRoad.m* to determine whether each particle is on the road. If the particle is not on the road, the weight of the particle will be set to zero. So these particles will have no influence to the estimate.
I think the computational complexity of both is almost the same, since we all added a judgment for each particle in each time step.

d)

We use the information of the map as measurement in the PF and use the CV motion model above to implement the PF.
We get the initial state from the true trajectory and the velocity measurement. The initial state is $[1.537; 1.456; 0; 0]$, and after tuning we get the $\sigma_v = 1$ for Q and $\sigma_r = 0.05$ for R.
And here are the figure shows the performance of the PF with/without the map information, the blue line shows the estimate, and the red one is the true trajectory:



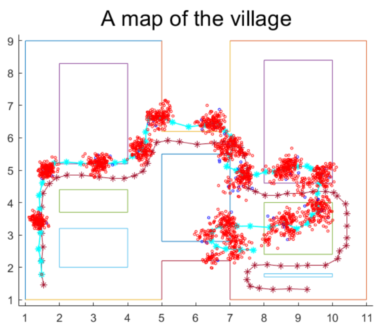
The estimated trajectory without map information



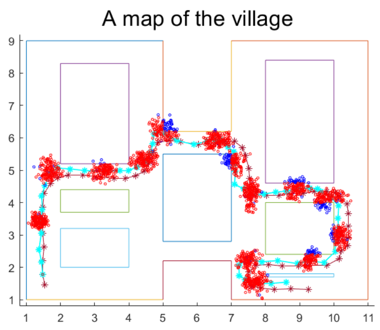
The estimated trajectory with map information

Figure 14

And here is the figure shows the particles every 5 times to visualize the filter performance.The blue line shows the estimate, and the red one is the true trajectory, the red points are the particles with weight bigger than 0, and the blue ones are with 0 weight:



The estimated trajectory without map information



The estimated trajectory with map information

Figure 15

e)

I can not position the bicycle for most of the time when I select the initial point randomly.
Firstly we get random initial points with the information of map. And these random points are showed on map:

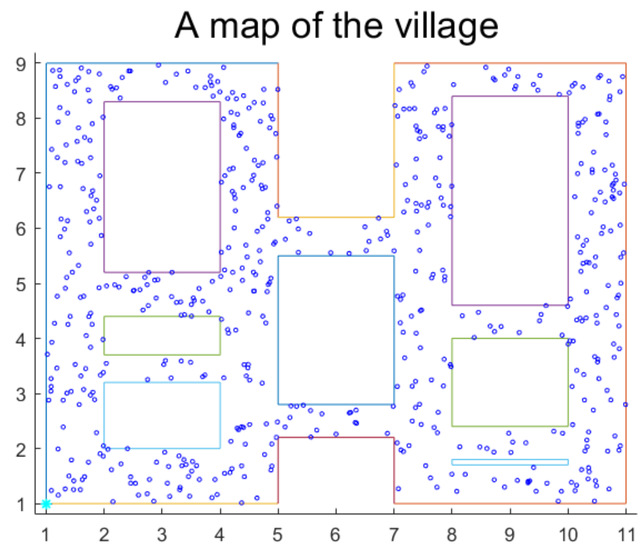


Figure 16

And then we randomly select one point from the points and see that as the initial state of the PF. For most cases the result are shown like the figure below. The blue line shows the estimate, and the red one is the true trajectory, the red points are the particles with weight bigger than 0, and the blue ones are with 0 weight.

As we can see that with wrong initial state, the first few steps would generate fake particles with the velocity measurement and estimate a trajectory with similar shape but wrong position. However when the fake trajectory goes into the buildings on the map, the weight of most of the particles will be zero and the filter failed. But sometimes the initial state is really close to the true initial place and the PF will perform just like the right picture of figure 8.

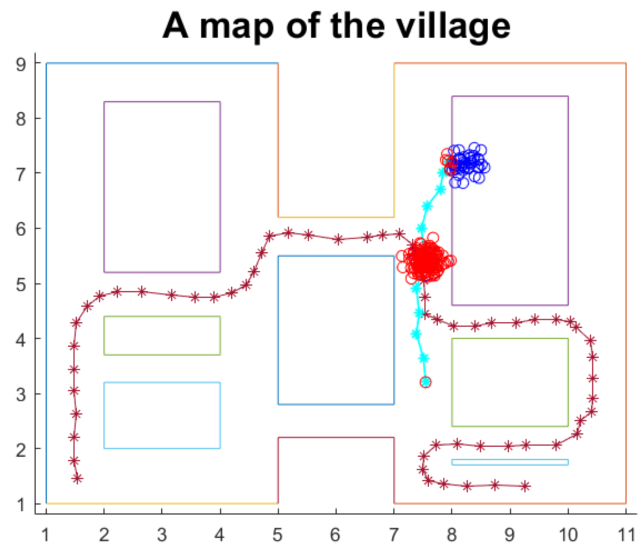


Figure 17