Solution to analysis in Home Assignment 4

Daniel Söderqvist (danisode)

May 22, 2023

Analysis

In this report I will present my independent analysis of the questions related to home assignment 4. I swear that the analysis written here are my own.

1 Smoothing

(a) In this task we are supposed to plot the positions of the sensors, the positions corresponding to the measurements, the true position sequence, the filtered and smoothed position trajectories as well as their corresponding 3σ-covariance contours for 5:th of the filter and smoothing estimates. We start by defining all that we know and use the given code to generate the states. We also use the noise covariance matrix Q given from the the previous HA3. We define all the function handles for both the motion model and measurement model, as well as the sigmaPoints function. To generate the measurements we use the previously constructed genNonLinearMeasurementSequence().

With everything defined we can now use the newly constructed function nonLinRTSsmoother() that both generate the estimates from a Kalman Filter but also the estimates with smoothing. The function given in the previous HA getPosFromMeasurement() converts the measurements into the Cartesian coordinate system. And the function sigmaEllipse2D() is used to plot the covariance ellipses. Finally we calculate the error between the true states, the Kalman filtered estimates and the Smoothing estimates. The results are ploted in the following figures (1.1) and (1.2). The first figure is showing the plot with everything that was asked, see description above and the second figure is showing the error.

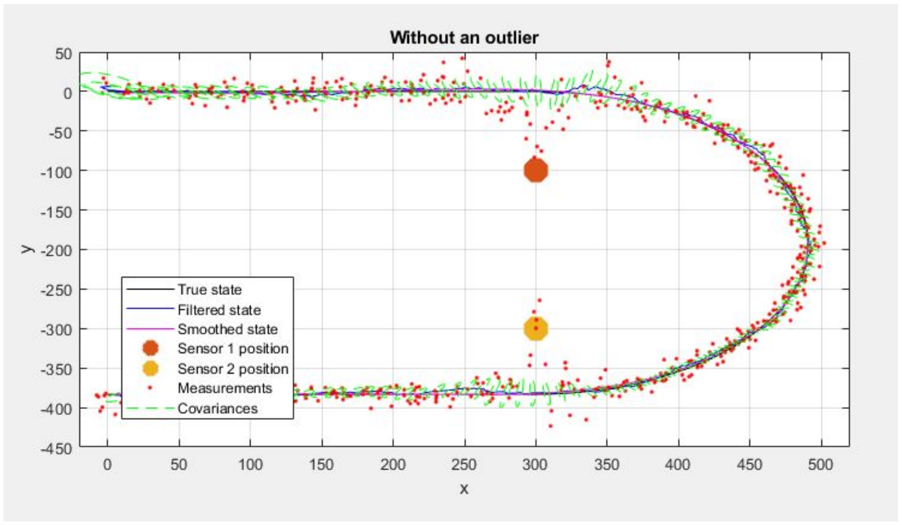

Figure 1.1: Figure showing the true state, measurements, sensor positions, covariances, filtered estimates and smoothing estimates.

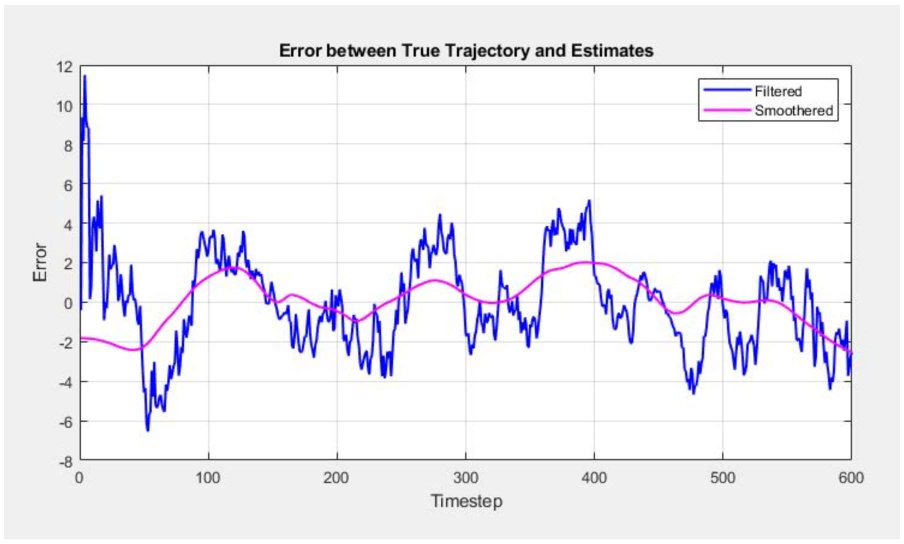

Figure 1.2: Figure showing the error between the true state and filtered estimates as well as the error between the true states and the smoothed estimates.

The conclusion drawn from the results are that the trajectories are pretty hard to see without zooming in a lot but we can see that the smoothed estimates follows the true state pretty well while the filtered in some places are a bit off. If we also look at the error between the filtered/smoothed estimates and the true states we can see that the error is more noisy and larger which means that by just using filtered estimates from forward approximation we get a worse estimation than by also use backward smoothing. This seems to reduce the noise and also make the error overall smaller. The reason for the error to be smaller is that the smoothed estimates have access to more information and data than the filtered estimates. The filtered estimates takes only information before and until that time-step into consideration while for the smoothed estimates it takes all states, and all measurements into consideration which should make it more precise with less covariance and better estimations than the filtered.

(b) In this task we are supposed to do the same thing as in the previous question but instead add an outlier (some noise) to the 300:th timestep. So we try a pretty extreme case and add 5 as noise to the 300:th measurement data (Y(300)). And then we simulate and calculate error etc in the same manner as in the previous sub-task. The results are shown in the following figures (1.3) and (1.4) down below. Where we in both of the figures show the same thing as in the previous sub-task:

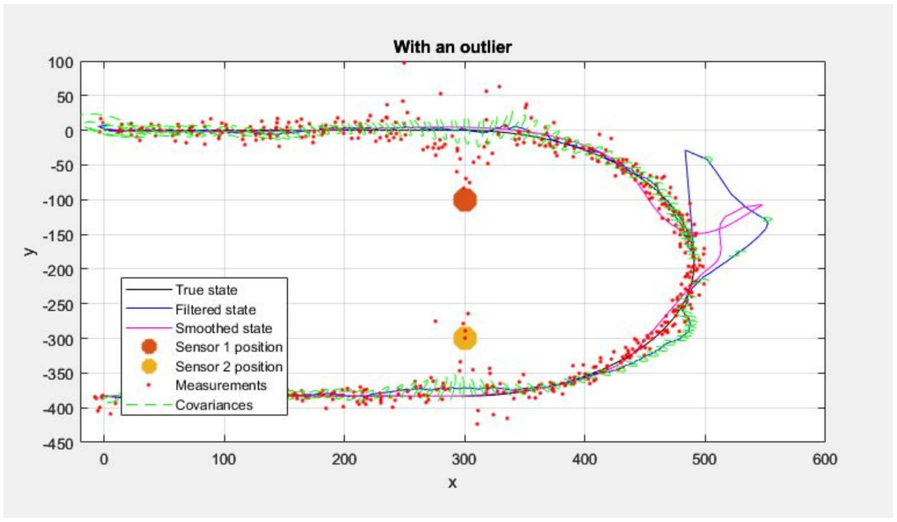

Figure 1.3: Figure showing the true state, measurements, sensor positions, covariances, filtered estimates and smoothing estimates but with an added outlier at time-step 300.

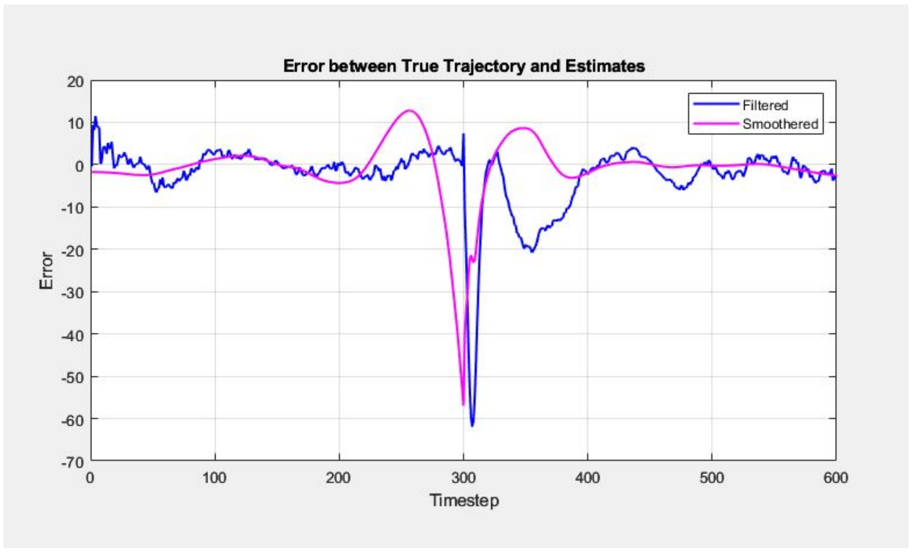

Figure 1.4: Figure showing the error between the true state and filtered estimates as well as the error between the true states and the smoothed estimates but with an added outlier at time-step 300.

We can see that in this case the outlier is affecting the estimated trajectory a lot both for the filtered and the smoothed ones. If we compare the filtered and the smoothed trajectory by looking at both figures showing both the plots and the errors we can draw the conclusion that: The filtered estimates is not affected by the outlier for as long as the smoothed estimates since it just takes the previous states and the previous measurements as well as the measurement that time-step into account. Therefore the kalman filter doesn't react until it reads the measurement with the outlier. The smoothed estimates takes everything into consideration since it have access to all the states and all the measurements. This will make the smoothed trajectory react before the actual outlier but will get less noisy and smoother than the filtered trajectory when it actually happens and after when it try to get back to the true state again or closer to it. One might say that in this case the filtered data is better right before since it follows the true states before at these time-steps while the smoothed is better during and after since the plot is somewhat better and the errors are less during these time-steps.

2 Particle filters for linear/Gaussian systems

(a) In this task we are supposed to use and compare a Kalman filter with a particle that uses both resampling and not. We start by defining all given data. In this task we know that we have a linear problem which mean we need to generate states and measurements with linear models which we constructed functions for in previous HA. Then we simply use the function for the Kalman Filter which we also constructed in previous HA. To this HA we made a function constructing a particle filter which we also will use with both resampling and without. The first thing thing we are supposed to do is comparing the MSE (mean squared error) between the true states, the kalman filter, the particle filter without resampling and the particle filter resampling. This is done by simply using the built in function mse() in matlab. The results from that task is presented down below:

- MSE for the Kalman Filter = 1.1687.
- MSE for the Particle Filter without resampling = 2.2461.
- MSE for the Particle Filter with resampling = 1.1562.

We can draw the conclusion that the Kalman Filter and the Particle filter with resampling gives fairly the same MSE. To note is that I use in this example 300 particles which is quite much and should give a better estimate closer to the Kalman Filter. Since we have a linear problem and a Kalman Filter is a optimal choice for those problems we are happy if the particle filter perform as good as the kalman filter. Without resampling we can see that the MSE is almost double the MSE for the Kalman filter which means that it doesn't perform as good at all. To also note is that less particles might be enough for the particle filter with resampling to perform as good as the kalman filter. We tried one example where we got fairly the same MSE for both the filters with 150 particles (it differs a lot from trial to trial as well). In order for the particle filter without resampling to perform as well as the kalman filter we need much more particles. I tried an example with an extreme case of 1 000 000 particles where the MSE got fairly equal between all of the filters.

What we can conclude is that the particle filter without resampling has a way worse performance and it might be because of the degeneracy (if 1 or few particles has a summed weight close to 1 and the rest of the particles a weight close to 0) which will make the particle filter perform worse over time. That's why we need a very large amount of particles to counter this. With resampling re-evaluate and give all particles the same weight which takes away the risk of degeneracy. We were also supposed to plot the true trajectory as well as the KF trajectory, PF without resampling trajectory and PF with resampling trajectory. We were hinted to plot these with the corresponding error compared to the true state. This could be done with the errorbar plot function in matlab. To start with we therefore calculated the corresponding error then we used the plot command to plot everything asked for. The results are shown down below in figure (2.1):

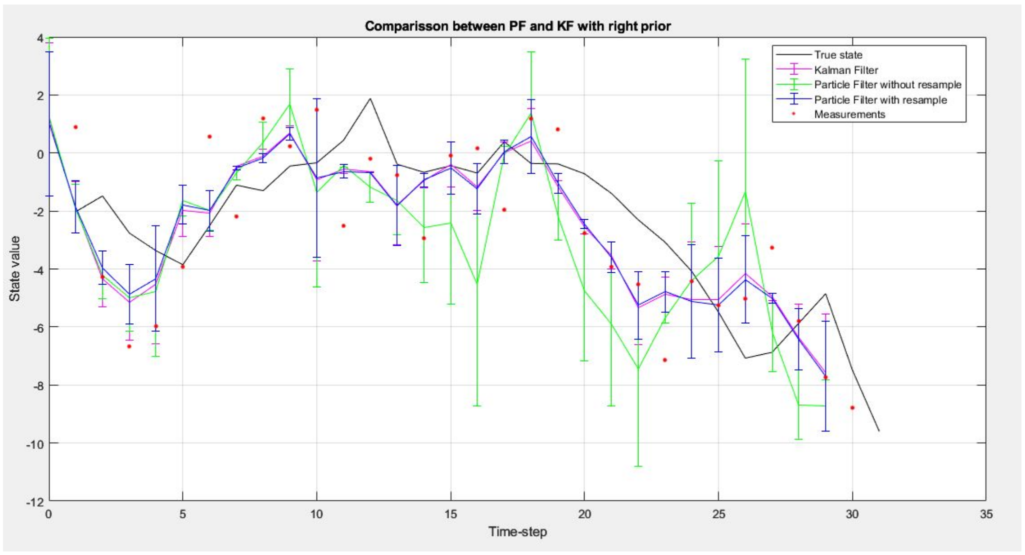

Figure 2.1: Figure showing true trajectory, measurements, KF trajectory, PF without resampling trajectory and PF with resampling trajectory. All with their corresponding error as errorbars.

We can see in this figure as well as when inspecting the MSE that with 300 particles the PF with resampling and the KF are performing almost equally good. This can be visually seen both on the trajectory and the errorbars. For the PF without resampling we can see that the performance is worse as expected as well with much larger errobars.

The last thing to do is to choose 3 different time-steps and plot their corresponding densities for all the filters. The choice of time-steps was 5, 15 and 25 and the figures down below are showing the difference between the Kalman filter and the particle filters with and without resampling, each figure showing each time-step. See figures (2.2-2.4):

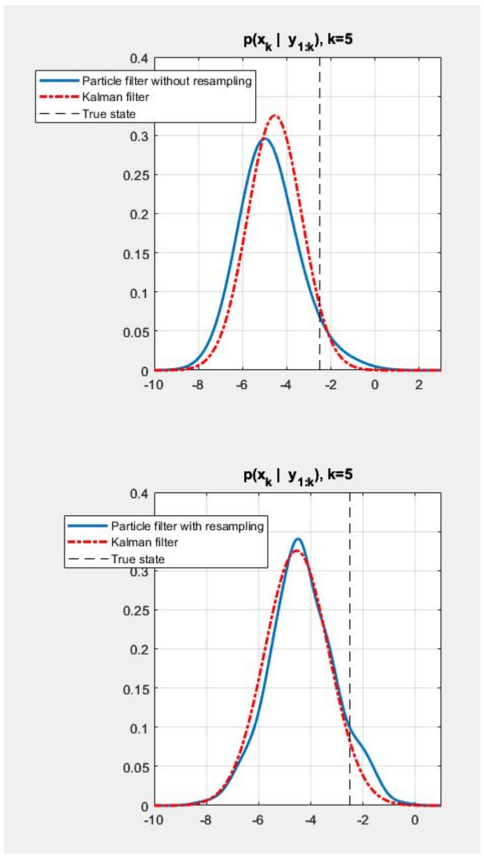

Figure 2.2: Figure showing true state, and the PDF for the KF and the PF without and with resampling for time-step = 5.

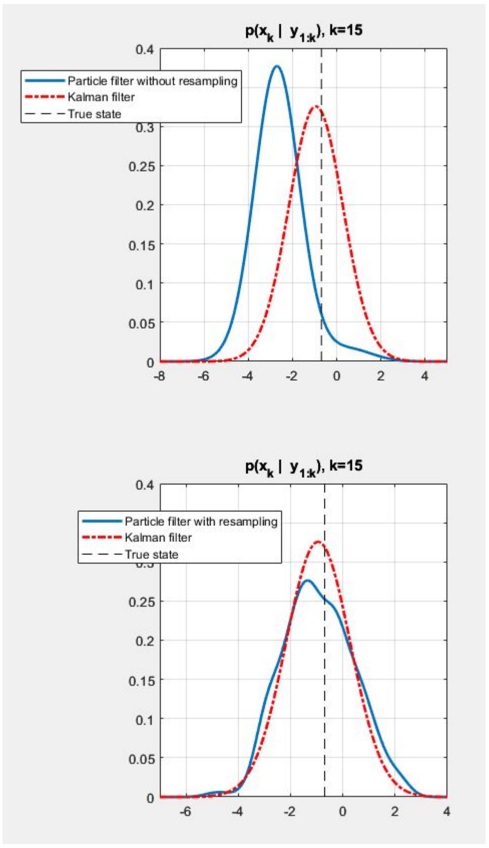

Figure 2.3: Figure showing true state, and the PDF for the KF and the PF without and with resampling for time-step = 15.

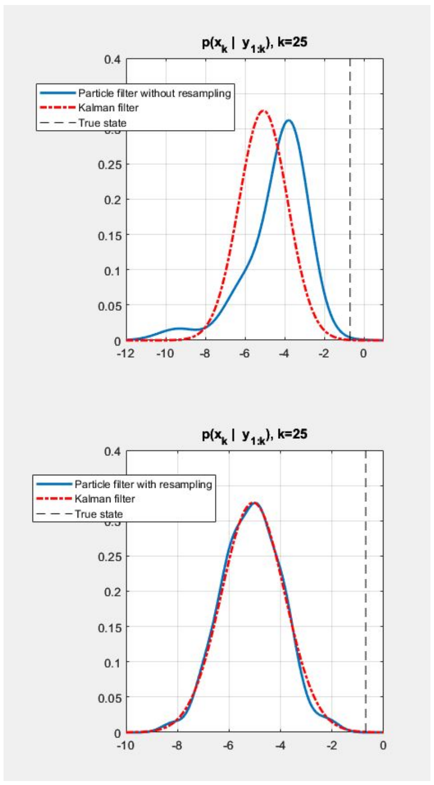

Figure 2.4: Figure showing true state, and the PDF for the KF and the PF without and with resampling for time-step = 25.

Even in the density functions we can see that the performance between the PF with and without resampling differs alot. With resampling the density funtion are almost equal to the KF which we also could see in the previous figures. One thing to note is that in the beginning, in figure (2.2) the performance is almost equally good or way better then the other time-steps if we compare with and without resampling on the PF.

I explained in the beginning that PF will suffer from degeneracy over time which means that in time-step 5 it hasn't been that affected yet which means that we get a somewhat good performance. Compared to the other time-steps where we can see that there is a larger difference between the KF and PF without resampling than the KF and the PF with resampling. This result just strengthens that statement.

(To note is that i went with sigma = 1 since i got a result i could draw some good conclusions from anyways. There might be a better tuning though but i was happy with the result.)

(b) In this task we are supposed to try the same thing as before but now with a totally wrong prior to see how fast the different filters adapt. The figure down below are showing the results where i will provide 1 plot without errorbars and 1 with in that order, see figure (2.5): The conclusion drawn

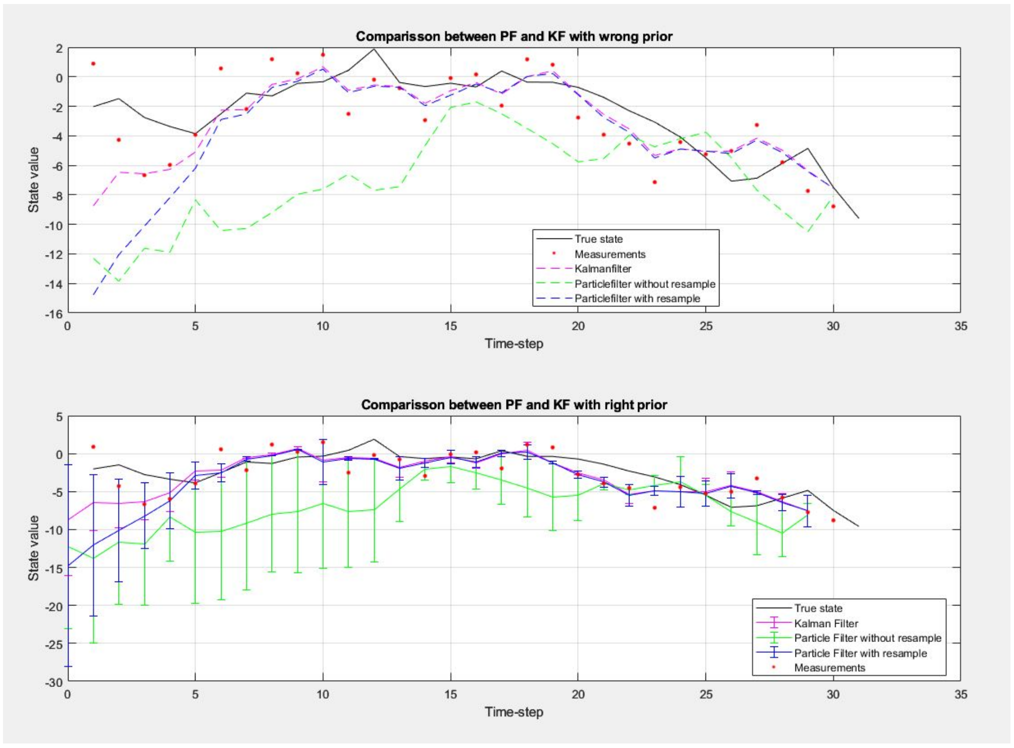

Figure 2.5: Figure showing true trajectory, measurements, KF trajectory, PF without resampling trajectory and PF with resampling trajectory. All with their corresponding error as errorbars. This when using the wron prior.

from this is the same thing as we have mentioned. The KF is the best filter since it adapts and get back to the right trajectory fastest with the smallest errors. This is because KF is the optimal filter for linear problems. We can see that with 300 particles the PF with resampling is not doing so bad and gets fairly close to the KF trajectory just a few time-steps later which means this is a good enough option as well. The performance of the PF without resampling is way worse in any way as expected.

(c) In this task we are supposed to plot all of the particle trajectories without resampling. In the question it is asked to plot 100 particles but the provided function had a maximum amount of 50 particles which i had to change to be able to plot 100 particles. I implemented the given functions in my PF and ploted the results which we can see in the figure down below:

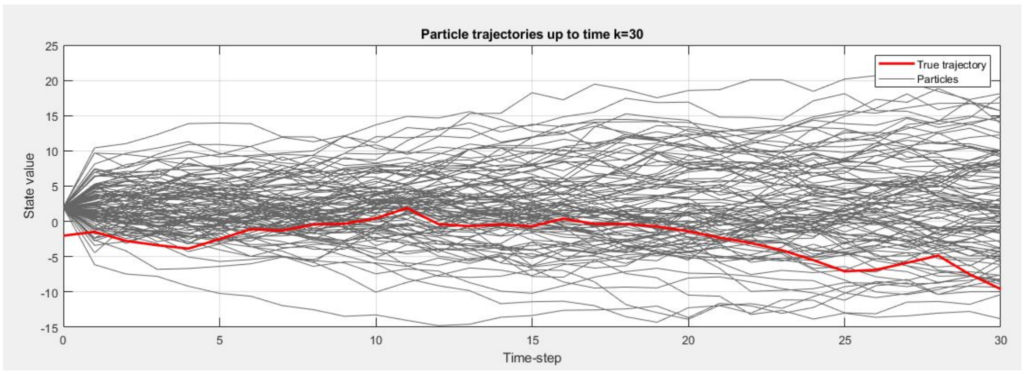

Figure 2.6: Figure showing true trajectory and all the 100 particle trajectories when using the PF without resampling.

We can see from the results that the more time-steps the particle travels the more they spread and drift away from the true trajectory. As i explained in the beginning of this task, when we use a PF without resampling the particles suffer from degeneracy which I explained previously will make the particle drift away and the error is getting larger over time which means that if we have a large amount of time-steps the PF will perform worse and worse. This is well illustrated in this figure which means that we strengthens also this statement.

(d) In this task we are supposed to do the same thing as in the previous task but now with the resampling instead. The results are plotted down below in figure (2.7):

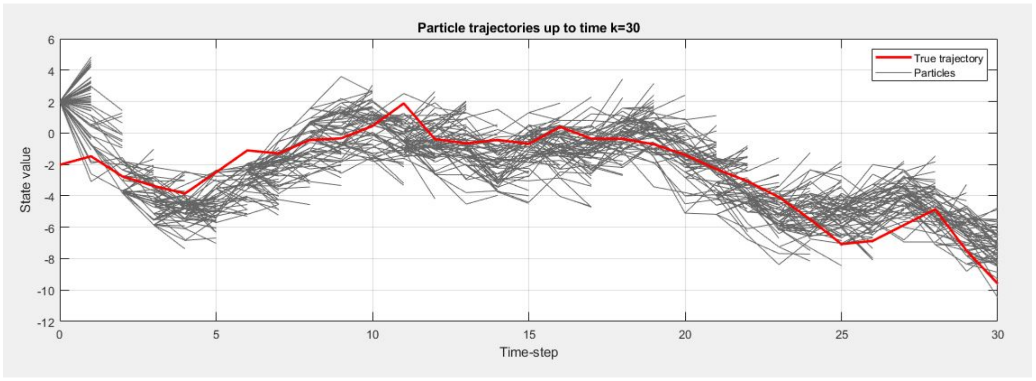

Figure 2.7: Figure showing true trajectory and all the 100 particle trajectories when using the PF with resampling.

We can in this figure see the opposite of what i described earlier. When we resample we don't give the particles the chance to degeneracy which means that the performance we have early on in the simulation we will have all the time-steps. This means that it doesn't matter how many time-steps we simulate we get the same good performance over time. This is also illustrated well in the figure where we can see that the particle trajectories are along the true trajectory all the way. This also strengthens that the performance of the KF and PF with resampling was equally good. We can see that with 100 particles as well we get a fairly good estimation of the true trajectory.

3 Bicycle tracking in a village

(a) In this task we are supposed to illustrate a trajectory of which a bicycle would travel through a city with different streets and houses etc. I clicked which was asked and then constructed a trajectory with it's belonging coordinates and saved these. The results was then ploted as following, see figure (3.1):

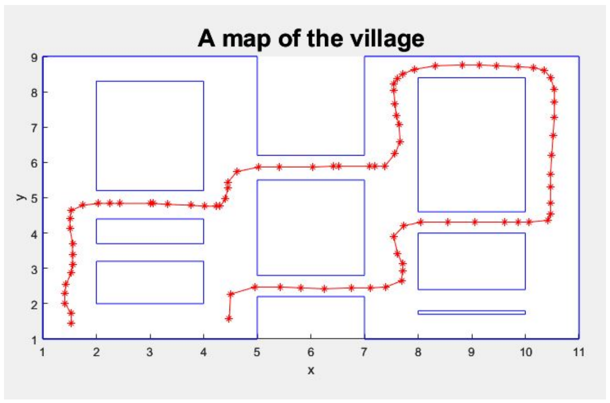

Figure 3.1: Figure showing a trajectory of a bicycle through a city.

(b) In this task we are supposed to calculate the velocity vector which was done by taking the velocity in each step and add some noise. By using the diff command for both the x-coordinates and y-coordinates separately we got the velocities from the position. The noise wasn't defined in the task but i set it to be R = diag([0.001 0.001]) since i didn't want it to affect the velocities to much to see if the result was good enough. The noise could easily be changed afterwards but in the velocity vector ploted in the following figure that's the noise that was used, see figure (3.2): We can

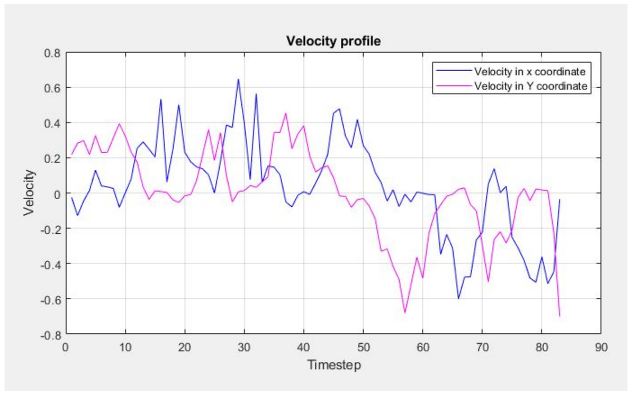

Figure 3.2: Figure showing the velocity in both x and y for each time-step.

see from the figure that the plots agree well with the bicycle trajectory in subtask a).

(c) I was kinda lost of what i actually was supposed to do here and i had to little time to figure it out.