Solution to analysis in Home Assignment 3

Yongzhao Chen(yongzhao@chalmers.se)

May 3, 2023

1 Approximations of mean and covariance

1.1 a

Kind of similar to HA2, the key code for 1a is listed here:

```
% Loop over the number of samples
for i = 1:numSamples
    % Sample the state vector x from the state density
    x = mvnrnd(mean_x, cov_x)';

    % Compute the dual bearing measurement vector y using the state
       vector x
    y = dualBearingMeasurement(x, s1, s2);

    % Add random sample noise to the measurement vector y
    y = y + mvnrnd(zeros(2, 1), cov_r)';

    % Store the sample measurement vector y
    samples_y(:, i) = y;
end
```

My `numSamples` set as 10,000.

1.2 b

Here I created a new function `MeanCovarianceY` based on `nonLinKFupdate`, since we do not need to update prior of x and its covariance, we only care about the mean and covariance of the measurement.

After modification, acutally its structure is the same with `nonLinKFprediction`.

It is not coincidence, for in this question we only care about positions that our state is the same with the measurement. So if we directly input h and R to replace the parameters f and Q in the function nonLinKFprediction, it can works.

Here is the table for 2b.

-	**EKF_MEAN**	**EKF_cov**
scenario 1	[0.1974; 1.3734]	[0.0017 0.0015 ;0.0015 0.0060]
scenario 2	[2.3562; 2.3562]	[0.0500 0.0100 ;0.0100 0.0020]
scenario 3	[-0.5880, 2.1588]	[0.0092 -0.0111 ; -0.0111 0.0148]
-	UKF_mean	UKF_cov
scenario 1	[0.1983 ;1.3743]	[0.0017 0.0015 ;0.0015 0.0059]
scenario 2	[2.3269; 2.3550]	[0.0600 0.0108 ;0.0108 0.0020]
scenario 3	[-0.5949; 2.1524]	[0.0099 -0.0112 ; -0.0112 0.0151]
-	CKF_mean	CKF_cov
scenario 1	[0.1983 ;1.3743]	[0.0017 0.0015 ;0.0015 0.0059]
scenario 2	[2.3265; 2.3550]	[0.0566 0.0105 ;0.0105 0.0020]
scenario 3	[-0.5948; 2.1523]	[0.0097 -0.0112 ; -0.0112 0.0150]

Table 1: Table For Three Densities

1.3 c

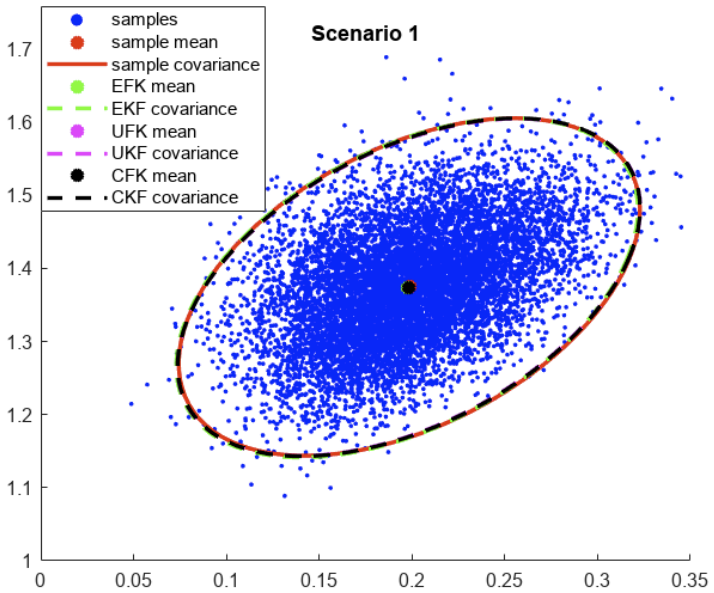

Figure 1: Density1

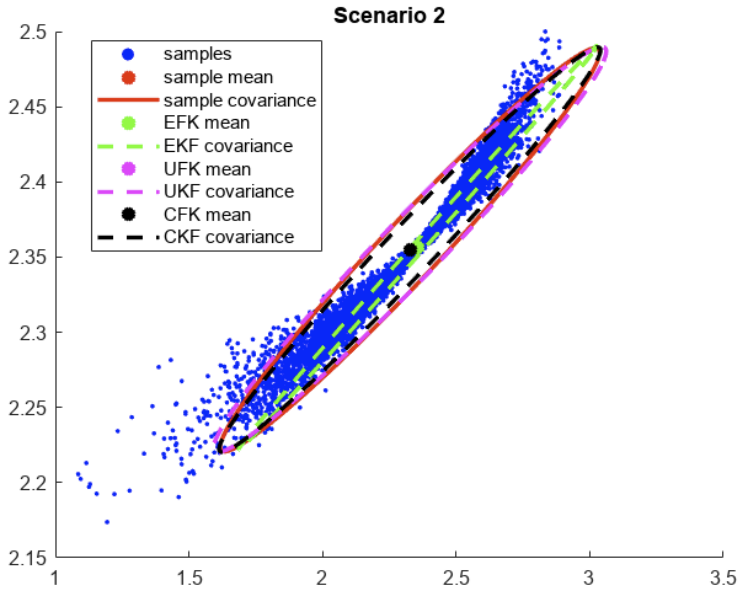

Figure 2: Density2

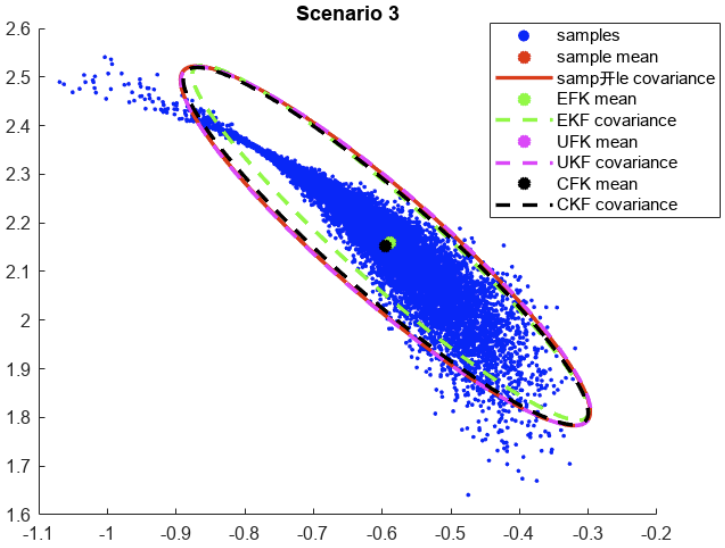

Figure 3: Density3

Note: The covariance curves are drawn by the function `sigmaEllipse2D` *created by myself in previous function, which depicted 3 sigma curves.*

In three figures, the aprromaxiation methods perform well that their mean and covariance are really close to the sample value.

But for `EKF` method, it only perform well in figure 1, in both 2 and 3, there are different extent of divergences, especially in 2, where its covariance has almost completely shifted away from the curve of sample covariance curve.

1.4 d

In all three pictures, the approximate means/covariances are nearly as the same places with the sample mean/covariance. That is because the approximation methods `UKF, CKF` all fairly useable in nonlinear measurement model.

But for `EKF` method, it perform much worser in figure 2 and figure 3. That is because the `EKF` assume the model is linear at the operating point, while in scenario 2 and 3 the model is too nonlinear for it.

UKF and CKF perform always good because it do not set requiements for the model.However, compared to EKF methods, they are more computationally intensive and more expensive to use.

If I were an engineer, I would definitely use the EKF method without a doubt in a very linear situation, because it is faster and cheaper. For mildly nonlinear cases, the EKF can be considered according to the actual needs, like in Figure 3, where the EKF can be used if the accuracy is not too demanding. But for extreme nonlinear cases, as in Figure 2, I would not choose the EKF method.

2 Non-linear Kalman filtering

2.1 a

Part a and b have lots of pictures, I listed them and discuss together.

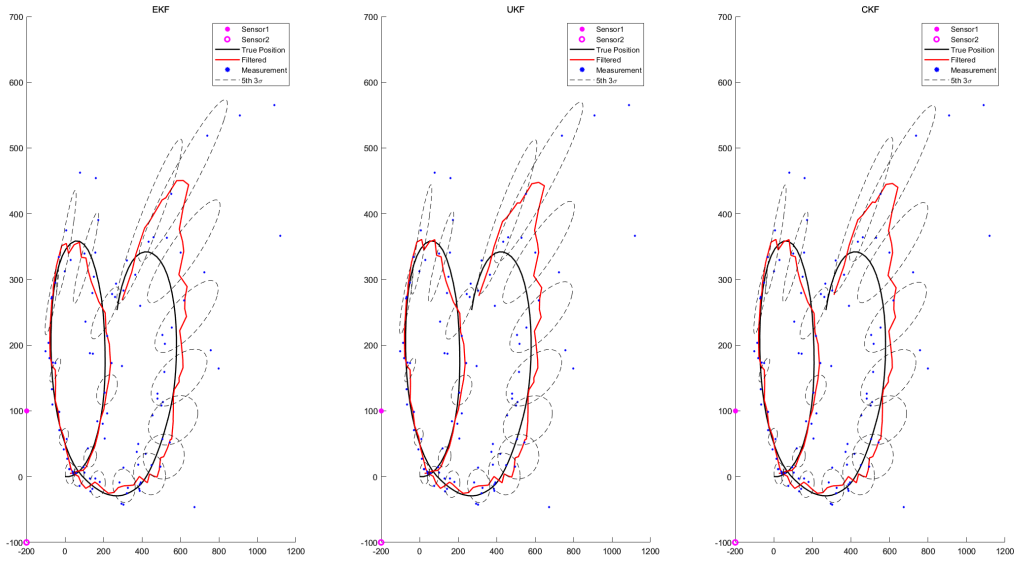

Figure 4: Normal Case

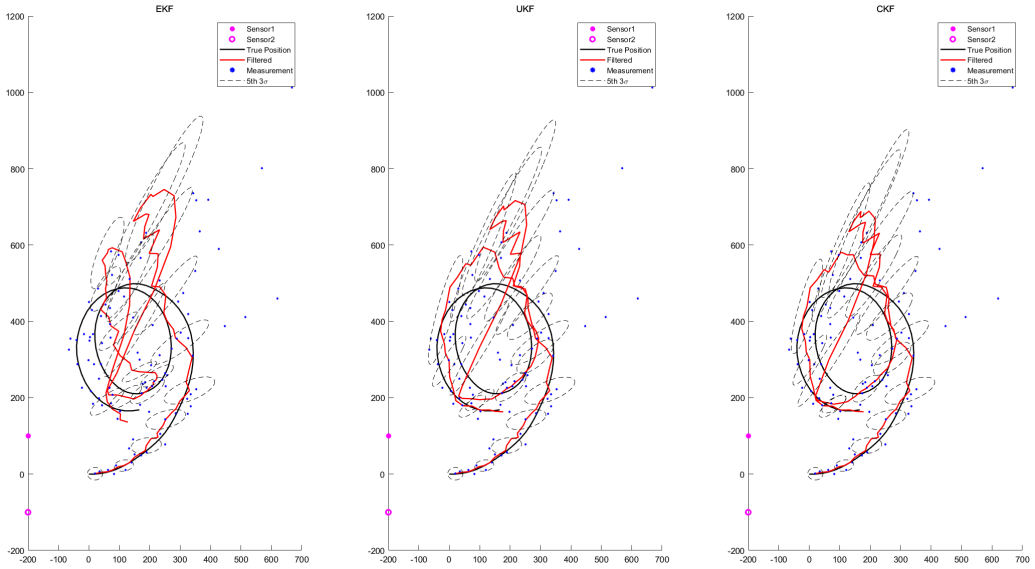

Figure 5: Extreme Situation (all bad)

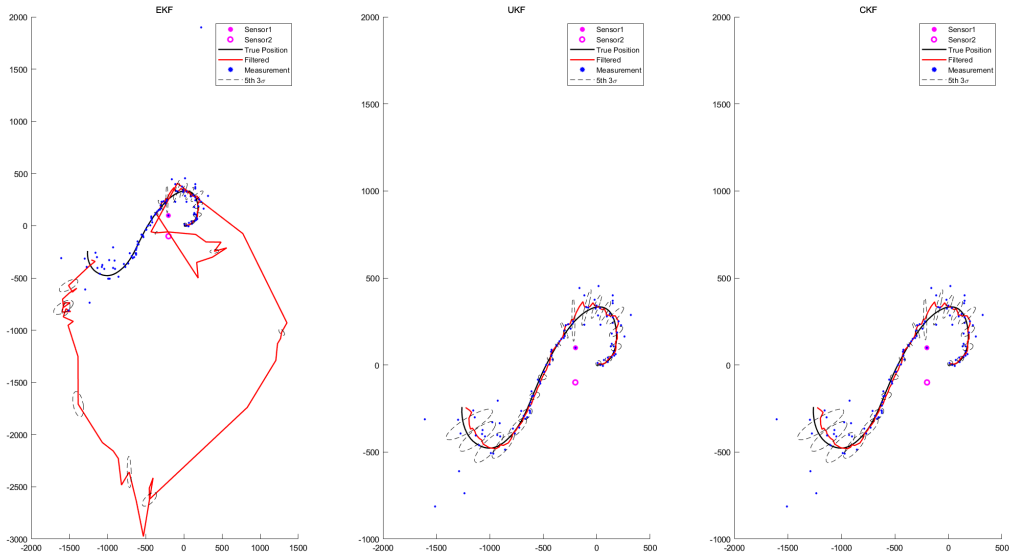

Figure 6: Extreme Situation (EKF bad)

2.2 b

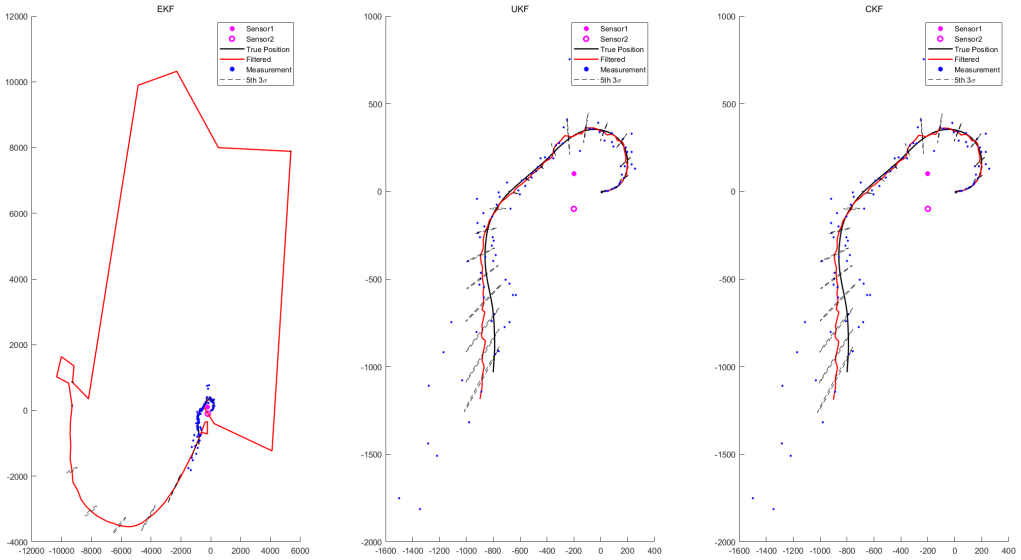

Figure 7: Extreme Case 2

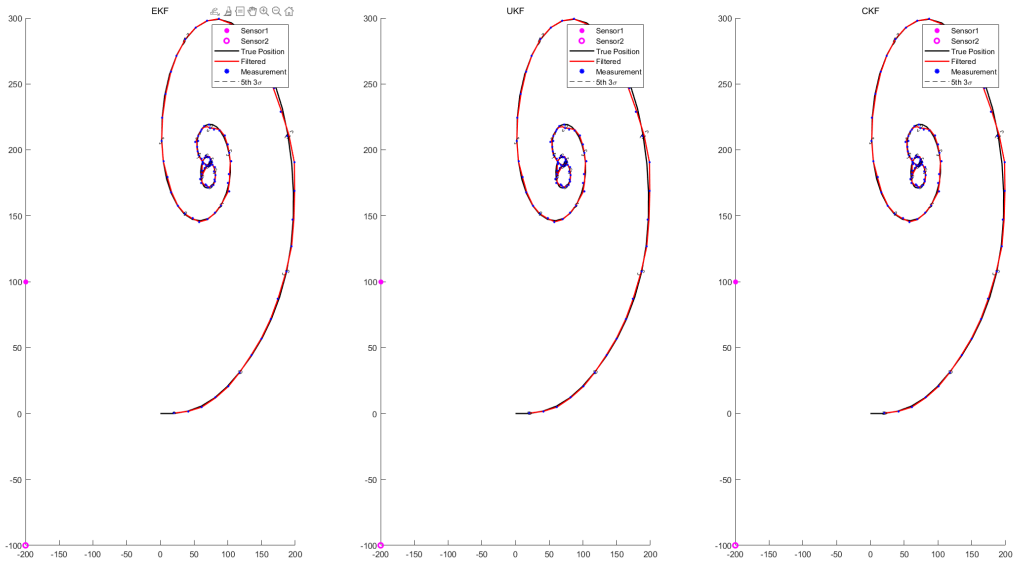

Figure 8: Normal Case 3

2.3 discuss a&b

For this question, I tried a very large number of times. In the vast majority of cases, there is no significant difference between EKF and UKF/CKF, it achieves tracking very well, and its covariance is not significantly larger compared to the other two, and only in a few cases can be distinguished by the naked eye, like in figure 5.

The covariance is well represented by the unvertainty, can be seen in Figure 4.

All methods are poor when the two trajectories are nearly coincident, showed in 5

The failure of the EKF can be observed when the car surrounds the sensor at a relatively close distance. This is because at very close distances to the sensor, as if two points were taken on a circle of small radius to connect the lines to represent the arc between them, the nonlinear of the car is so severe that the EKF fails. These cases showed in 6 and 7

To be honest, although the extreme situation will fail, in my experience, the reliability of the EKF is amazing.

2.4 c

Upon the figures, I print the mean and covariance of the estimation errors. In this way it can be much more clear to evaluate the quality of the filter.

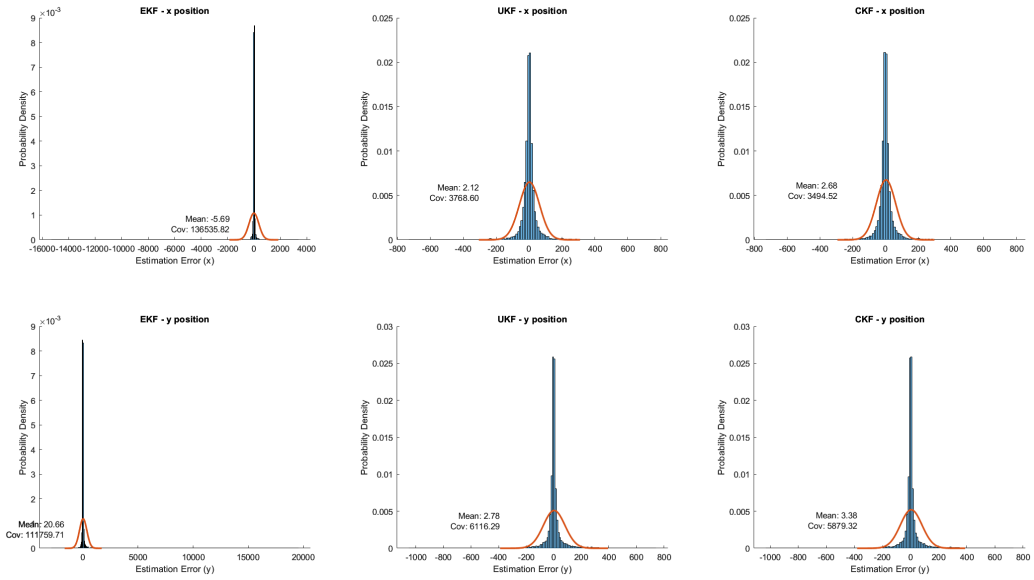

Figure 9: Case 1

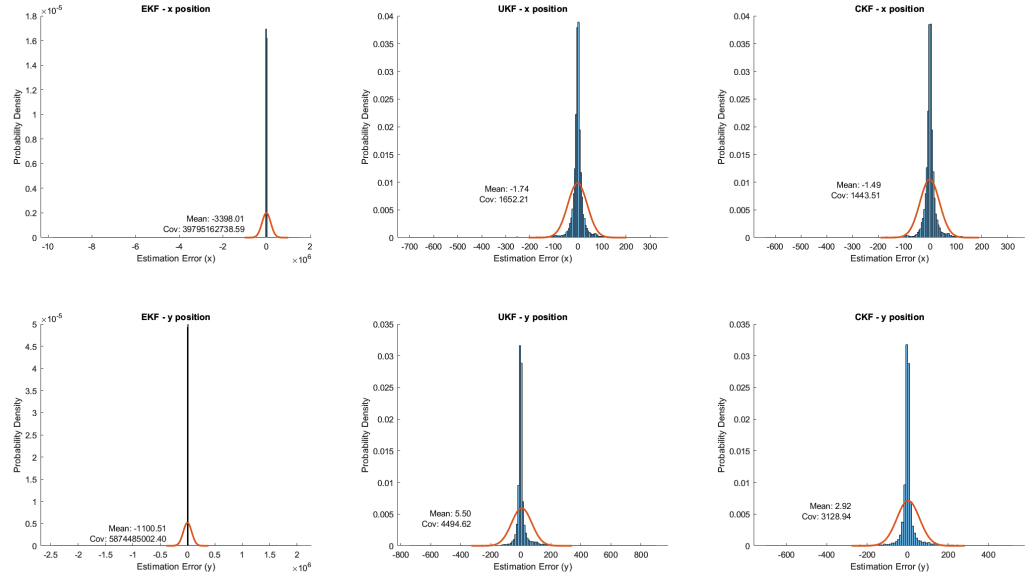

Figure 10: Case 2

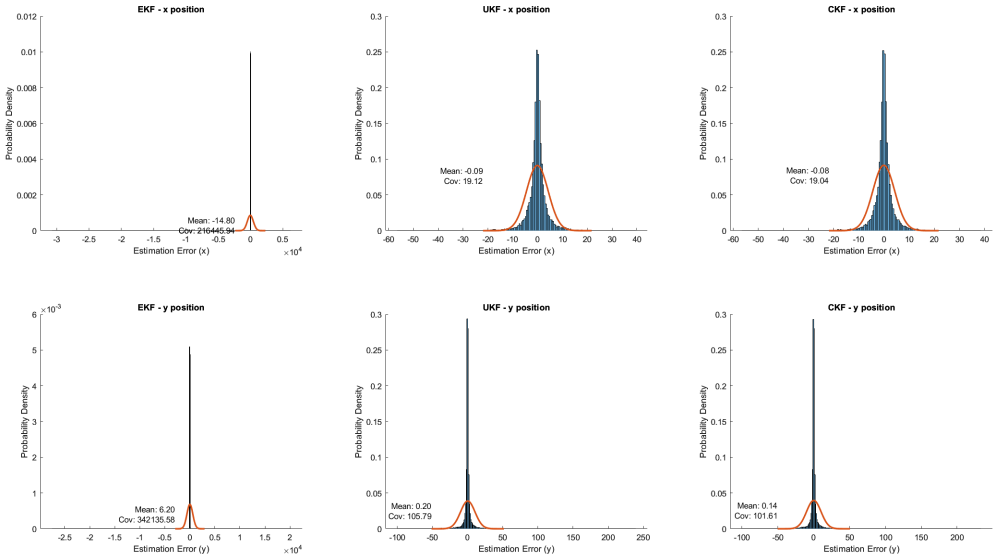

Figure 11: Case 3

From figures above, concentrate on the Mean value (the closer to zero, the better the quality), we can see that EKF has the worst performance within three methods, while the UKF and the CKF has nearly the same performance.

The histogram of EKF has nothing bussiness with Gaussian distribution, and that explains why its mean so far away from zero. From the histograms of UKF and CKF, we can see the histograms for x look quite like Gaussian. But when turns to y, since it is a nonlinear measurement model, the more the nonlinear, the less the histograms look like Gaussian.

And the normalpdf curves are drawn within the $\pm 5\sigma$ since from the figures in Task a we can see the measurement points are so scattered that the nosies are too big, which leads to many points do not lie within the $\pm 5\sigma$ zone. So even with normalization, the curve can only looks quite like Gaussian but not coincident to it like the previous homeworks.

3 Tuning non-linear filters

For this part, mainly use the code from question 2 a and b.

Firstly, Draw the figure containing the true positions, filtered positions, sensor positions, the measurements, 5σ curves, and the true position sequence. This

figure can very graphically tell us the tracking effect of the filter and the general covariance.

Secondly, draw the histogram of the error. This can help us evaluate the quality of the filter.

3.1 a

3.1.1 Tuning big

σ_v=1 σ_w=pi/180

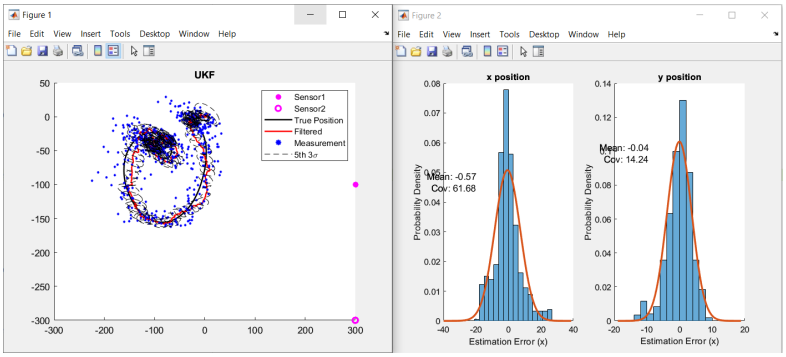

Figure 12: Begin Value

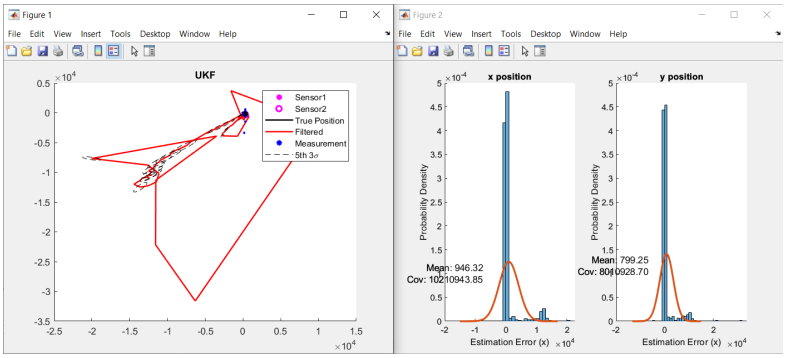

Figure 13: σ_v times 1000, σ_w keeps 1

Firstly use large process noise, which means the filter now do not believe the state velocity from motion model, and it can now track the true state.

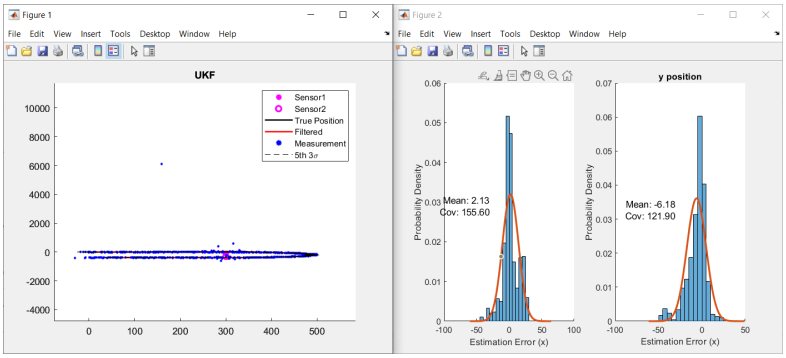

Figure 14: σ_w times 1000

Then use large σ_w, which means the filter now do not believe the rotation speed of the motion model, it can still track the true state, since the tracking states are positions not the gesture. But it still make the filter quality worser.

Then increase both, we should see the seem result with only increase σ_v

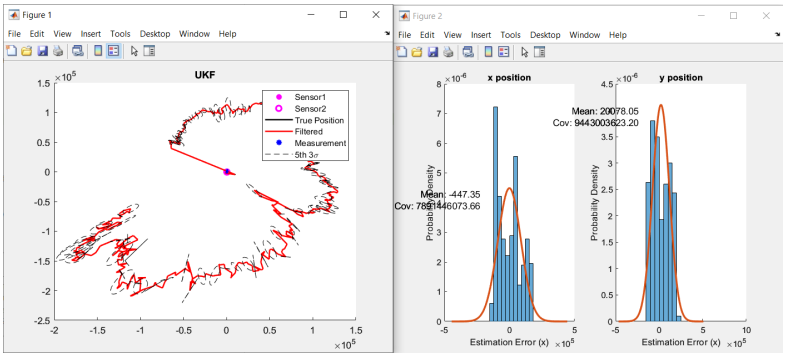

Figure 15: Both times 1000

And it fits my guess.

3.2 Tuning down

From tuning big part, I guess turn the σ_v down enhance the quality, while σ_w only matter a little.

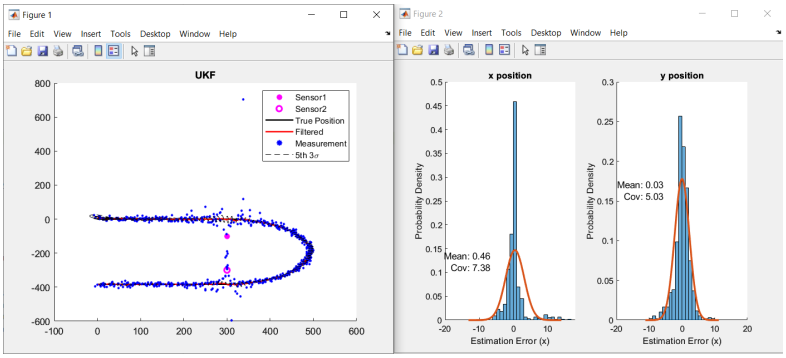

Figure 16: $\sigma_v/1000$

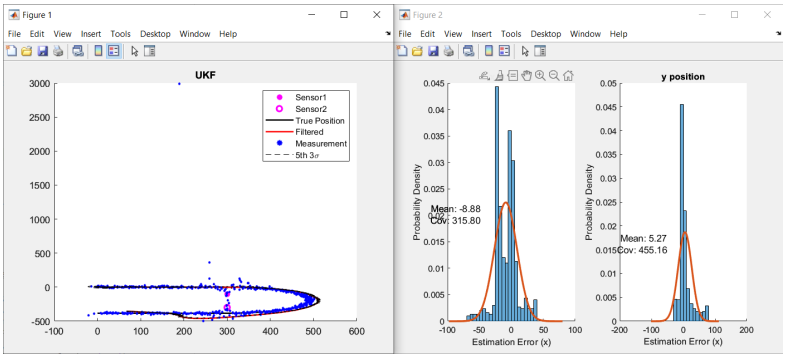

Figure 17: $\sigma_w/1000$

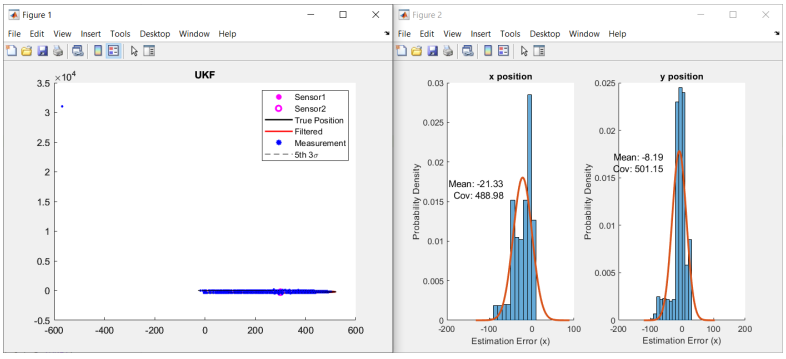

Figure 18: Both / 1000

But my guess is not fully correct. Now the motion covariances are two small that we only believe the motion model, which make the system lag behind the change.

When both parameters are really small, the filter quality drops.

3.3 b

I tuned σ_v as 50 and σ_w as $pi/180 * 60$. Under these parameters the filer can always track the true state and the mean and covariance of the state x is better then the beginning value.

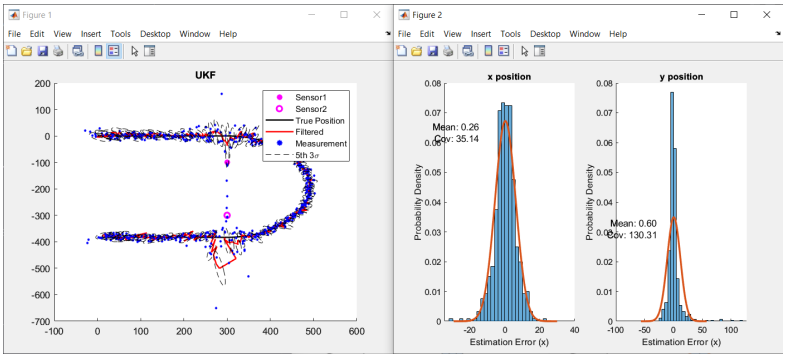

Figure 19: Mytune

3.4 c

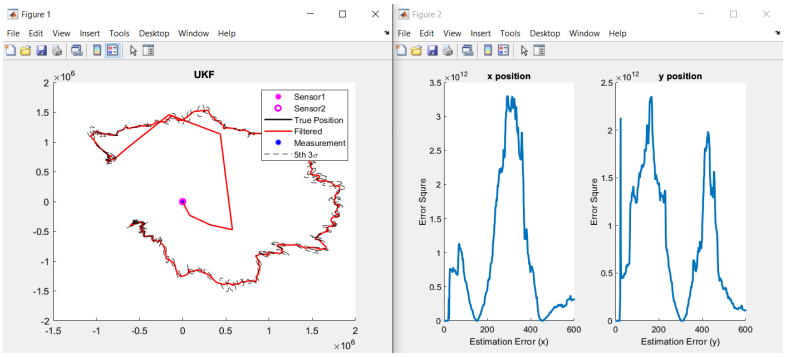

Figure 20: Toobig: Both times 1000 1000

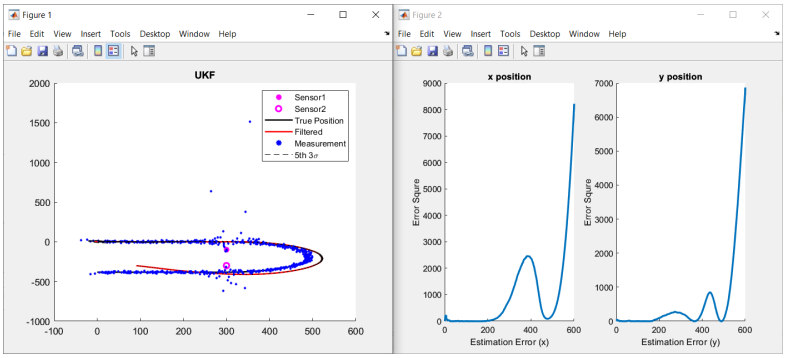

Figure 21: Too small: Both are divided by 1000

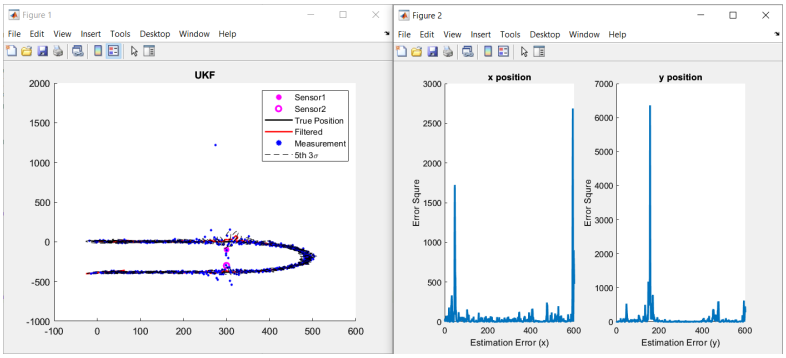

Figure 22: Welltuned

From figures above, it is obvious that the well tuned one has the minimum average position errors, which means it perform the best, and matches the left side state curve.

3.5 d

It is not possible to have accurate estimates over the whole sequence. Since we have turn in the true state, we need flexibility to follow this trun, or we will loose the track of the true state.

There is a conflict when tuning for different parts of the true trajectory, for in some parts the system do not change its state that the parameters can set very small while in some parts the car may turn direction, need bigger parameters.

We would set a much smaller parameter for a completely straight line, and a much larger parameter for a purely turning session.

For changing from straight to turn, since it generally picks up more changes after this operation, I think its parameter should be larger to get more flexibility than changing from turn to straight, which may keep driving in a straight line.

To conclusion, the purpose of introducing motioin covariance into the motion model is to increase the flexibility of the system and to make it robust enough to cope with changes such as sudden acceleration and steering of the vehicle.

It is true that we can get very small error for a given case, but this is not migratory at all and is a meaningless behavior.