ME425 HW3

Localization of a Robot Using Simple, Kalman and Extended Kalman Filters

Edin Guso 23435
17 May 2019

Contents

Simple Filter

Problem Definition

In the first part of the homework it is asked to localize a static robot which takes distance measurements to an object using the recursive least squares algorithm. It is given in the problem that the sensor has a variation of 20 and that the results are uniformly distributed between 1 and 11.

Development and Algorithm

In the beginning of the recursive least squares algorithm development, it is assumed that the prediction is a linear combination of different information. Also, the sum of all the coefficients should be equal to 1 since otherwise the prediction would be too large.

$$\hat{x} = \omega_1 x_1 + \omega_2 x_2, \qquad \omega_1 + \omega_2 = 1$$

Now, the problem is to obtain the optimal ω_1 and ω_2. In order to obtain these optimal values, we need to find when the variance of $\hat{x}$ is minimum. After some probability calculations and further simplifying the problem by letting

$$\omega_1 = 1 - \omega, \qquad \omega_2 = \omega$$

The equation for the variance of $\hat{x}$ is

$$\hat{\sigma}^2 = (1-\omega)^2 {\sigma_1}^2 + \omega^2 {\sigma_2}^2$$

Since we want to minimize this value, we need to take its derivative with respect to ω and set it equal to 0. After some more calculations, it is found that the optimal ω is

$$\omega = \frac{{\sigma_1}^2}{{\sigma_1}^2 + {\sigma_2}^2}$$

When we plug in this optimal ω into the predicted state equation, the optimal predicted state is found to be

$$\hat{x} = \frac{{\sigma_2}^2}{{\sigma_1}^2 + {\sigma_2}^2} x_1 + \frac{{\sigma_1}^2}{{\sigma_1}^2 + {\sigma_2}^2} x_2$$

Furthermore, the variance of the optimal predicted state can be calculated as

$$\sigma^2 = \frac{{\sigma_1}^2 {\sigma_2}^2}{{\sigma_1}^2 + {\sigma_2}^2}$$

In our problem definition, the two information sources are the previous optimal predicted state ($\hat{x}(k)$) and the measurements ($x(k)$) with variance V. Knowing this, we can rewrite the equations which will be used in the recursive algorithm as

$$\omega(k) = \frac{\sigma^2(k)}{\sigma^2(k) + V}$$

$$\hat{x}(k+1) = \hat{x}(k) + \omega(k)(x(k) - \hat{x}(k))$$

$$\sigma^2(k+1) = (1 - \omega(k))\sigma^2(k)$$

Implementation

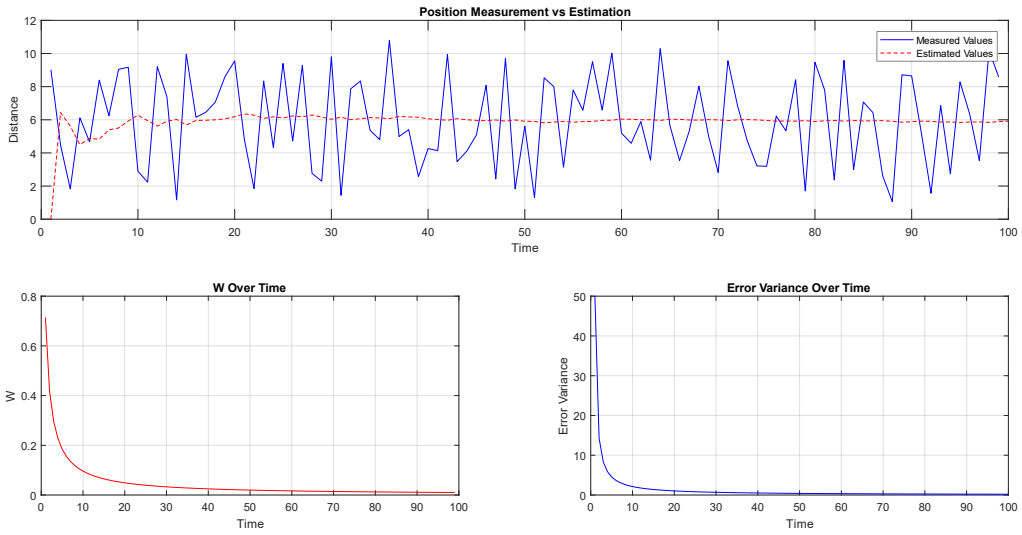
In the recursive least squares algorithm developed above, it should be noted that the first optimal predicted state and the first variance of the optimal predicted state should be initialized since they are used in the equations. In our case, it was given in the homework document that the state should be initialized to 0 and the variance should be initialized to 50. The logic behind these is that we have no idea about the state so it should be set as any arbitrary value and since we are very uncertain of our prediction the variance should be set to a high value.

The implementation of the algorithm in Matlab can be seen below.

SimpleFilter.m

```
clear all; clc;

%recursive least squares algorithm

x(1) = 0;
error_var(1) = 50;
sensor_var = 20;

k = 1;
while k<100
    meas(k) = rand() * 10 + 1; %value between 1-11
    w(k) = error_var(k) / ( error_var(k) + sensor_var );
    x(k+1) = x(k) + w(k) * ( meas(k) - x(k) );
    error_var(k+1) = ( 1 - w(k) ) * error_var(k);
    k = k + 1;
end

fig = figure('Name', 'Results');
subplot(2, 2, [1 2]);
plot(meas, 'b'); hold on; plot(x, 'r--');
legend('Measured Values', 'Estimated Values');
xlabel('Time'); ylabel('Distance'); grid on;
title('Position Measurement vs Estimation');

subplot(2, 2, 3);
plot(w, 'r'); xlabel('Time'); ylabel('W');
title('W Over Time'); grid on;

subplot(2, 2, 4);
plot(error_var, 'b'); xlabel('Time'); ylabel('Error Variance');
title('Error Variance Over Time'); grid on;

set(fig, 'Position',  [50, 100, 1500, 700]);
saveas(fig, 'SimpleFilter.svg');
```

Results and Discussion



The results obtained are displayed in the figure above. The predicted state varies a lot in the first few time steps (which can also be shown with the variance graph). This can simply be explained with the fact that at the beginning, the robot is highly uncertain of its position and therefore the measurements (which are very noisy) affect the predicted state more than the previous state. However, over time it builds up belief (the variance decreases), which causes the previous predicted state to contribute more into the calculation of the next predicted state compared to the measurements. After some time, the value converges to 6, which is the mean of the measurements.

ω was high in the beginning, showing that the innovation factor ($x(k) - \hat{x}(k)$) was dominant in the predicted state calculation. As explained previously, this makes sense since the previous predicted state is highly uncertain. Over time, ω converges to a value close to 0 since the previous predicted state is dominant in the predicted state calculation.

Finally, the error variance starts from the initial value as 50 and quickly converges to 0. The reason it converges to 0 is because the algorithm was developed based on the principle of minimizing error variance.

Kalman Filter

Problem Definition

In the second part of the homework, we were required to design and implement a Kalman Filter for the position state estimate of a holonomic robot. All the noise in the measurements (both the distance and the bearing angle measurements), as well as the noise associated with the motion model of the robot are said to be Gaussian.

Development and Algorithm

The main idea of the Kalman Filter is to predict the optimal state using two main steps. These two main steps are very similar to the simple filter in terms of logic. The state is estimated using the previous optimal state and corrected using the measurements in order to obtain the current optimal state.

First step is the predictor step which estimates the current state using the process model. Current estimated state $\hat{x}^-(k+1)$ of the system is found by using the previous optimal state $\hat{x}(k)$ and current input to the system $u(k+1)$.

$$\hat{x}^-(k+1) = A\hat{x}(k) + Bu(k+1)$$

In the equation given above A and B are matrices which represent the system model. However, in this case, both A and B are 2x2 identity matrices since we are dealing with a holonomic robot. Current covariance matrix associated with the estimated state (a priori covariance) $P^-(k+1)$ depends on the previous covariance matrix associated with the optimal state (a posteriori covariance) $P(k)$, matrix A and process noise covariance matrix Q. The equation is given as

$$P^-(k+1) = AP(k)A^T + Q$$

Then, there is an intermediate step which calculates the current Kalman Gain matrix $K(k+1)$ using the a priori covariance matrix $P^-(k+1)$, state to measurement transformation matrix H and measurement noise covariance matrix R. The H in this case is a 2x2 identity matrix. This observation is made with the fact that the measurements distance d and bearing angle α are polar coordinates and can be transformed into the cartesian coordinates easily. The equation for Kalman Gain calculation is as follows

$$K(k+1) = P^-(k+1)H^T(HP^-(k+1)H^T + R)^{-1}$$

The second step is the corrector step. Here, the measurements are used to correct the predicted state and obtain the optimal state. Current optimal state $\hat{x}(k+1)$ is calculated by using the current predicted state $\hat{x}^-(k+1)$, current Kalman Gain matrix $K(k+1)$, current measurements $z(k+1)$ and the state to measurement transformation matrix H. Following is the equation

$$\hat{x}(k+1) = \hat{x}^-(k+1) + K(k+1)(z(k+1) - H\hat{x}^-(k+1))$$

Finally, the current a posteriori covariance matrix $P(k+1)$ is calculated using the current Kalman Gain matrix $K(k+1)$, state to measurement transformation matrix H and current a priori covariance matrix $P^-(k+1)$. The equation can be seen below

$$P(k+1) = (I - K(k+1)H)P^-(k+1)$$

Implementation

As in the simple filter, some of the values have to be initialized at the beginning so that the algorithm can run. These values were the optimal state and a posteriori covariance matrix. First optimal state has been initialized to (0,0) since that is the starting point of the robot and a posteriori covariance matrix has been initialized to a 50*I (identity matrix).

The code written in Matlab to implement the Kalman Filter and plot the results can be seen below.

KalmanFilter.m

```
 clear all; close all; clc;
%Initial values
Q = 0.01 * eye(2);
R = [0.01 0; 0 0.01];
X_real(:,1) = [0 0];
X_optimal(:,1) = takeMeasurement(X_real(:,1), R);
P(:,:,1) = 50 * eye(2);
%Matrix values
A = eye(2);
B = eye(2);
H = eye(2);
k=1;
while k<100
    U(:,k+1) = calculateInput(k);
    X_real(:,k+1) = moveRobot(X_real(:,k), U(:,k+1));
    X_real(:,k+1) = X_real(:,k+1) + addNoise(X_real(:,k), Q);
    Z(:,k+1) = takeMeasurement(X_real(:,k+1), R);
    X_apri(:,k+1) = A * X_optimal(:,k) + B * U(:,k+1);
    P_apri(:,:,k+1) = A * P(:,:,k) * transpose(A) + Q;
    K(:,:,k+1) = P_apri(:,:,k+1) * transpose(H) * inv( H *
P_apri(:,:,k+1) * transpose(H) + R );
    X_optimal(:,k+1) = X_apri(:,k+1) + K(:,:,k+1) * ( Z(:,k+1) - H *
X_apri(:,k+1) );
    P(:,:,k+1) = ( eye(2) - K(:,:,k+1) * H ) * P_apri(:,:,k+1);
    k = k + 1;
end
fig1 = figure;
subplot(1,3,1);
    hold on; grid on;
    plot(X_real(1,:), 'r');
    plot(X_apri(1,:), 'b--');
    xlabel('Time'); ylabel('X values'); title('X values vs time');
    legend('Real X', 'Predicted X');
subplot(1,3,2);
    hold on; grid on;
    plot(X_real(2,:), 'r');
    plot(X_apri(2,:), 'b--');
    xlabel('Time'); ylabel('Y values'); title('Y values vs time');
    legend('Real Y', 'Predicted Y');
```

```
subplot(1,3,3);
    hold on; grid on;
    plot(X_real(1,:), X_real(2,:), 'r');
    plot(X_apri(1,:), X_apri(2,:), 'b--');
    xlabel('X values'); ylabel('Y values'); title('Y values vs X
values');
    legend('Real Position', 'Predicted Position');

fig2 = figure;
subplot(1,3,1);
    hold on; grid on;
    plot(X_real(1,:), 'r');
    plot(X_optimal(1,:), 'b--');
    plot(Z(1,:), 'g--');
    xlabel('Time'); ylabel('X values'); title('X values vs time');
    legend('Real X', 'Optimal X', 'Measured X');
subplot(1,3,2);
    hold on; grid on;
    plot(X_real(2,:), 'r');
    plot(X_optimal(2,:), 'b--');
    plot(Z(2,:), 'g--');
    xlabel('Time'); ylabel('Y values'); title('Y values vs time');
    legend('Real Y', 'Optimal Y', 'Measured Y');
subplot(1,3,3);
    hold on; grid on;
    plot(X_real(1,:), X_real(2,:), 'r');
    plot(X_optimal(1,:), X_optimal(2,:), 'b--');
    plot(Z(1,:), Z(2,:), 'g--');
    xlabel('X values'); ylabel('Y values'); title('Y values vs X
values');
    legend('Real Position', 'Optimal Position', 'Measured
Position');

fig3 = figure;
subplot(2,2,1);
    plot(squeeze(P_apri(1,1,:))); title('P_a_p_r_i(1,1)');
subplot(2,2,2);
    plot(squeeze(P_apri(1,2,:))); title('P_a_p_r_i(1,2)');
subplot(2,2,3);
    plot(squeeze(P_apri(2,1,:))); title('P_a_p_r_i(2,1)');
subplot(2,2,4);
    plot(squeeze(P_apri(2,2,:))); title('P_a_p_r_i(2,2)');
```

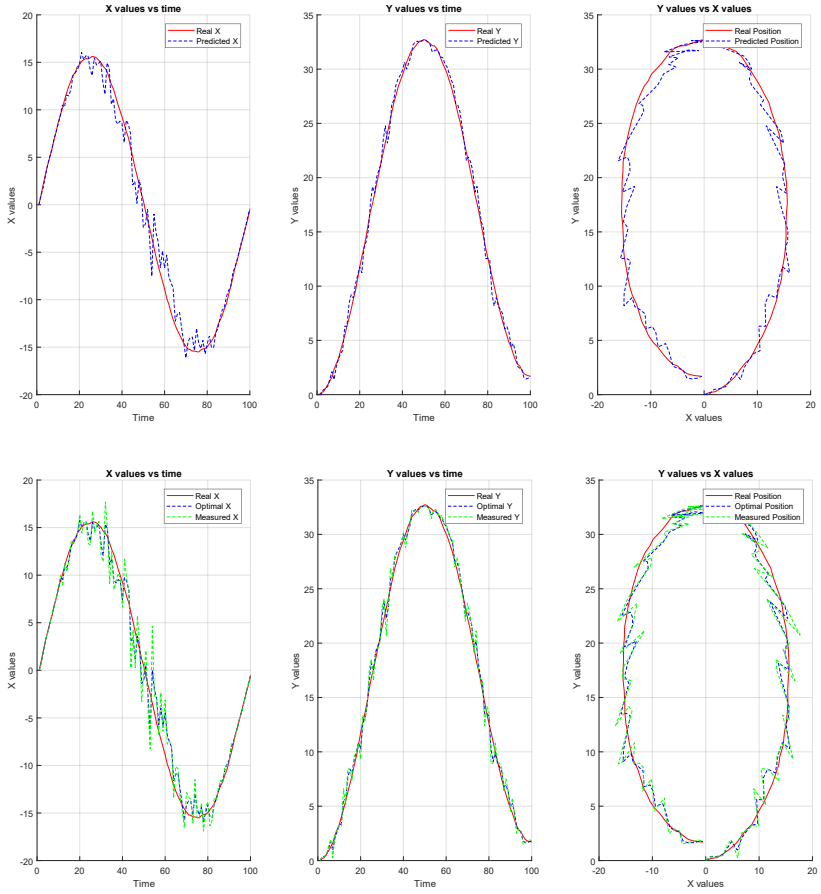
```
fig4 = figure;
subplot(2,2,1);
    plot(squeeze(P(1,1,:))); title('P(1,1)');
subplot(2,2,2);
    plot(squeeze(P(1,2,:))); title('P(1,2)');
subplot(2,2,3);
    plot(squeeze(P(2,1,:))); title('P(2,1)');
subplot(2,2,4);
    plot(squeeze(P(2,2,:))); title('P(2,2)');

fig5 = figure;
subplot(2,2,1);
    plot(squeeze(K(1,1,:))); title('K(1,1)');
subplot(2,2,2);
    plot(squeeze(K(1,2,:))); title('K(1,2)');
subplot(2,2,3);
    plot(squeeze(K(2,1,:))); title('K(2,1)');
subplot(2,2,4);
    plot(squeeze(K(2,2,:))); title('K(2,2)');

function X_real_new = moveRobot(X_real, U)
X_real_new(1) = X_real(1) + U(1);
X_real_new(2) = X_real(2) + U(2);
end

function U = calculateInput(k)
theta = k*2*pi/100;
U(1) = cos(theta);
U(2) = sin(theta);
end

function Z = takeMeasurement(X_real, R)
d = sqrt(X_real(1)^2 + X_real(2)^2);
theta = atan2(X_real(2), X_real(1));
real_val = [d; theta];
meas_val = real_val + sqrt(R) * randn(size(real_val));
Z(1) = meas_val(1) * cos(meas_val(2));
Z(2) = meas_val(1) * sin(meas_val(2));
end

function w = addNoise(X, Q)
w = sqrt(Q) * randn(size(X));
end
```

Results and Discussion

Some of the parameters of the simulations have been fixed throughout all of the cases and some of them have been modified to examine their effects in particular. The values that are not fixed are the process noise covariance matrix Q and measurement noise covariance matrix R. Control input to the system is an open loop cosine wave for x and sine wave for y. Also, the starting position, starting optimal state and the starting a posteriori covariance matrix have all been kept at the value explained in the previous part throughout the experiments.

Case 1

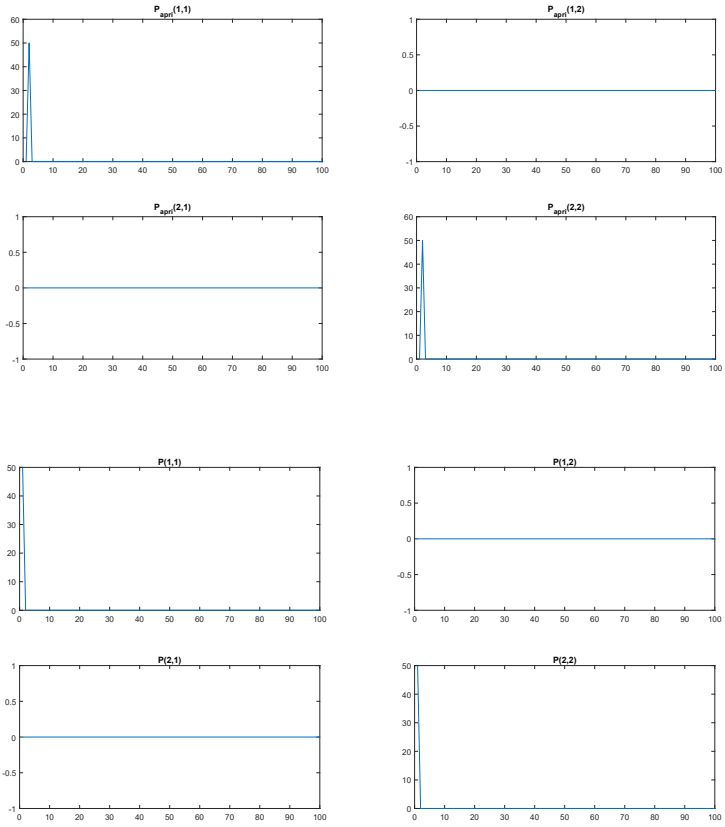
$$Q = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix} \qquad R = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}$$

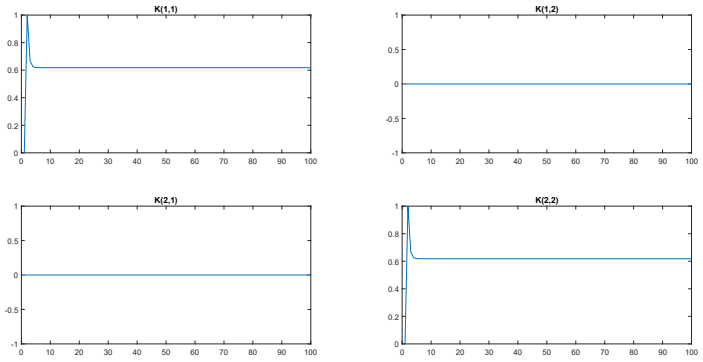


As it can be seen from the graphs on the left, the optimal state has an equal tendency to believe the system model and the measurements. This is caused by both process noise covariance matrix Q and measurement noise covariance matrix R having the same values. This also causes the noise in both the system and the measurements to be very similar, justifying the belief preferences of the system.

There is also a considerable amount of drift caused by the high noise in the system. It can be seen that the robot does not complete the circle.

On the right are the graphs for the a priori error covariance matrix values over time and a posteriori error covariance matrix values over time. Since the covariance matrix values of both the system and the measurements are the same, both error covariance matrices converge to a value close to them. The a priori error covariance matrix diagonal values converge to 0.0162, while a posteriori error covariance matrix diagonal values converge to 0.0062. It is clear that a posteriori values are lower since it is the fusion of two independent sources, while the a priori values are based on system model alone.

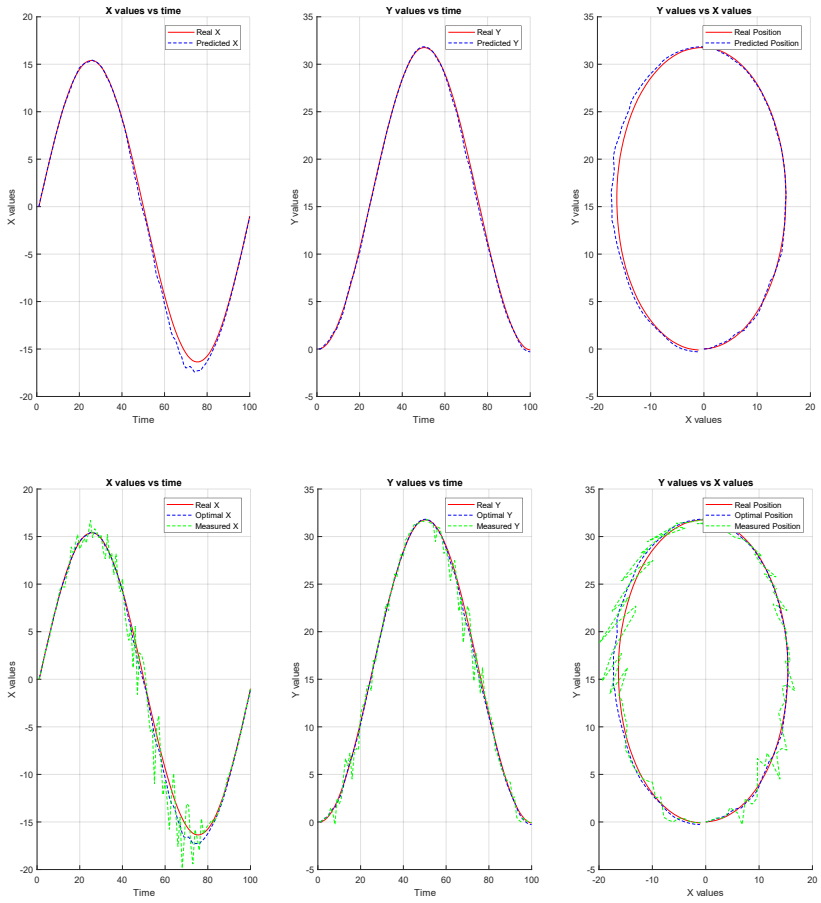




Similar observations are also made when the Kalman gain matrix is considered. The Kalman gain matrix converges to a value of 0.618 at the diagonal elements. It is clear that it should converge to a value around 0.5 since the belief for both the system model and the measurements is very similar.

Case 2

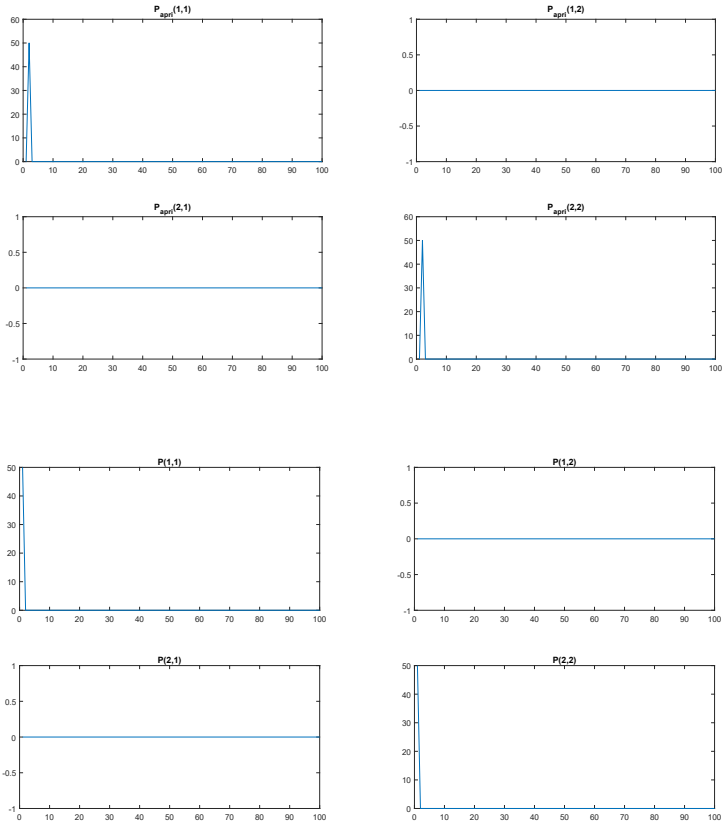
$$Q = \begin{bmatrix} 0.0001 & 0 \\ 0 & 0.0001 \end{bmatrix} \qquad R = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}$$

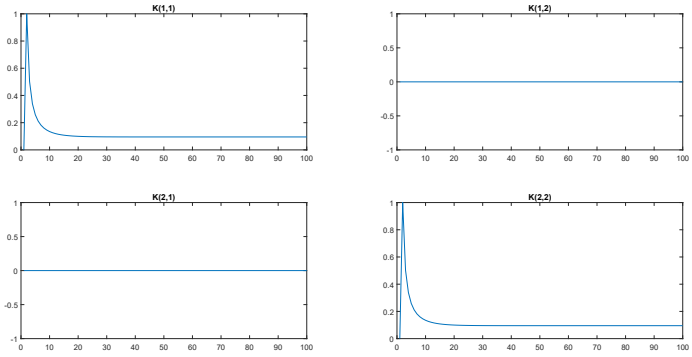


As it can be seen from the graphs on the left, the optimal state has a higher tendency to believe the system model compared to the measurements. This is caused by process noise covariance matrix Q having much lower values than the measurement noise covariance matrix R. This also causes the noise in the measurements to be higher, justifying the belief preferences of the system.

There is also very little drift caused by the low noise in the system. It can be seen that the robot almost completes a circle.

On the right are the graphs for the a priori error covariance matrix values over time and a posteriori error covariance matrix values over time. Since the covariance matrix values of the system is lower than the measurements, both error covariance matrices converge to a value close to the system values. The a priori error covariance matrix diagonal values converge to 0.0011, while a posteriori error covariance matrix diagonal values converge to 0.001. A posteriori values and a priori values are close since the fusion of sources is highly dominated by the system model.

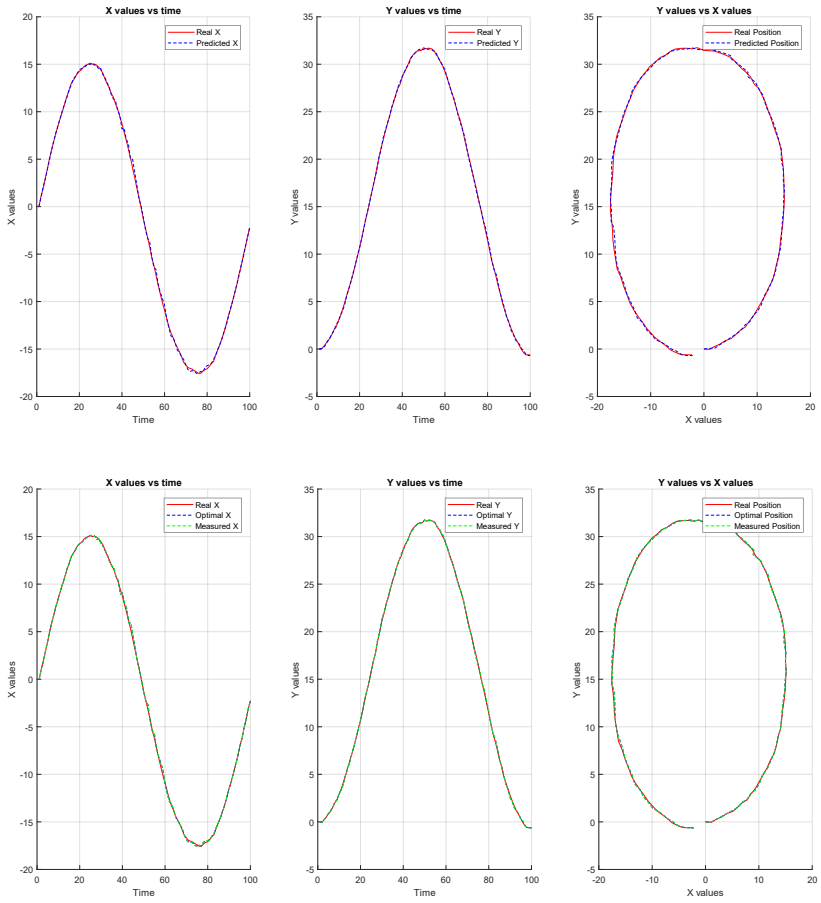




Similar observations are also made when the Kalman gain matrix is considered. The Kalman gain matrix converges to a value of 0.095 at the diagonal elements. It is clear that it should converge to a value close to 0 since the belief for the system model is much higher and there is almost no need for innovation term.

Case 3

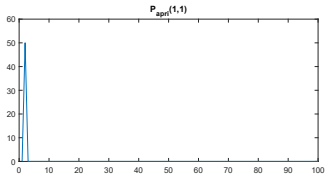
$$Q = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix} \qquad R = \begin{bmatrix} 0.0001 & 0 \\ 0 & 0.0001 \end{bmatrix}$$

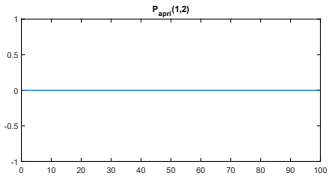


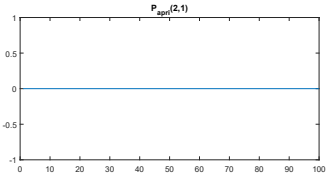
As it can be seen from the graphs on the left, the optimal state has a higher tendency to believe the measurements compared to the system model. This is caused by process noise covariance matrix Q having much higher values than the measurement noise covariance matrix R. This also causes the noise in the measurements to be lower, justifying the belief preferences of the system.

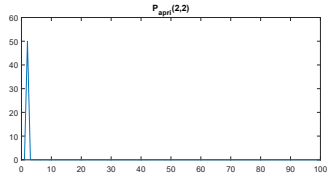
There is also a considerable amount of drift caused by the high noise in the system. It can be seen that the robot does not complete the circle.

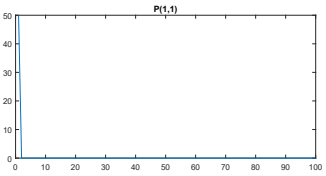
On the right are the graphs for the a priori error covariance matrix values over time and a posteriori error covariance matrix values over time. Since the covariance matrix values of the system are higher than the measurements, a priori error covariance matrices converge to a value close to the system values (0.01), while a posteriori error covariance matrix values converge to values close to the measurement values (0.0001). A posteriori values are much smaller since the model is highly unreliable while the measurements are reliable causing the optimal state to become precise.

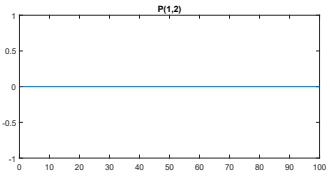


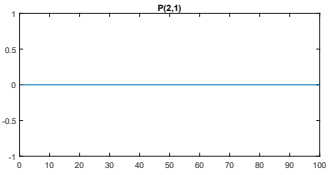


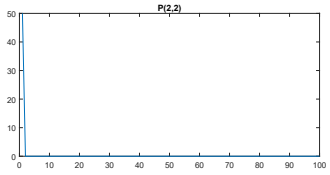


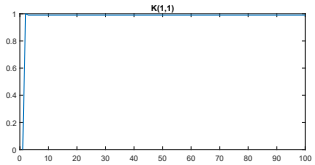


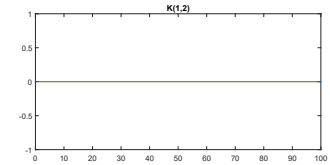


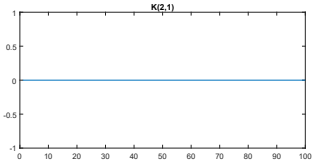


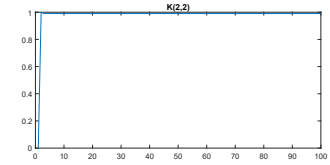












Similar observations are also made when the Kalman gain matrix is considered. The Kalman gain matrix converges to a value of 0.99 at the diagonal elements. It is clear that it should converge to a value close to 1 since the belief for the measurements is much higher and innovation term should be very dominant.

Extended Kalman Filter

Problem Definition

The final task of the homework was to implement an extended Kalman Filter for a non-holonomic point robot. This is very similar to the Kalman Filter done in the previous part. The only difference is that the system model is now non-linear.

Development and Algorithm

The same main idea as in the Kalman Filter still applies such that the goal is to predict the optimal state using two main steps. In the first step, the state is estimated using the previous optimal state and in the second step it is corrected using the measurements in order to obtain the current optimal state.

In the first step current state is estimated using the process model. Current estimated state $\hat{x}^-(k+1)$ of the system is found by substituting the previous optimal state $\hat{x}(k)$ and current input to the system $u(k+1)$ into the non-linear system model.

$$\hat{x}^-(k+1) = f(\hat{x}(k), u(k+1))$$

In the equation given above we don't have the A and B identity matrices that we had in the holonomic case since the $f(\hat{x}(k), u(k+1))$ is the non-linear process model. The function and its Jacobian matrix A will be further explained on the next page. Current covariance matrix associated with the estimated state (a priori covariance) $P^-(k+1)$ depends on the previous covariance matrix associated with the optimal state (a posteriori covariance) $P(k)$, Jacobian matrix A and process noise covariance matrix Q. The equation is given as

$$P^-(k+1) = AP(k)A^T + Q$$

Then, there is an intermediate step which calculates the current Kalman Gain matrix $K(k+1)$ using the a priori covariance matrix $P^-(k+1)$, state to measurement transformation matrix H and measurement noise covariance matrix R. The H in this case is a 2x3 matrix describing the transformation from states to measurements. The matrix is very similar to the holonomic case, with the only difference being the added 3rd column consisting of zeros, since the robot orientation is not related to neither of the measurements taken. The equation for Kalman Gain calculation is as follows

$$K(k+1) = P^-(k+1)H^T(HP^-(k+1)H^T + R)^{-1}$$

The second step is the corrector step. Here, the measurements are used to correct the predicted state and obtain the optimal state. Current optimal state $\hat{x}(k+1)$ is calculated by using the current predicted state $\hat{x}^-(k+1)$, current Kalman Gain matrix $K(k+1)$, current measurements $z(k+1)$ and state to measurement transformation matrix H. Following is the equation

$$\hat{x}(k+1) = \hat{x}^-(k+1) + K(k+1)(z(k+1) - H\hat{x}^-(k+1))$$

Finally, the current a posteriori covariance matrix $P(k+1)$ is calculated using the current Kalman Gain matrix $K(k+1)$, state to measurement transformation matrix H and current a priori covariance matrix $P^-(k+1)$. The equation can be seen below

$$P(k+1) = (I - K(k+1)H)P^-(k+1)$$

The previously mentioned non-linear system model $f(\hat{x}(k), u(k+1))$ and its Jacobian matrix A will be explained here. Since the robot in this task is a non-holonomic robot, it can be modeled with a non-linear system model shown below

$$f(\hat{x}(k), u(k+1)) = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} x + v\cos\theta \\ y + v\sin\theta \\ \theta + \omega \end{bmatrix}$$

Where x, y and θ are the optimal states of the system at the previous time step and v and ω are the control inputs at the current time step. Since this system is non-linear, it is necessary to take the Jacobian so that it is possible to have an A matrix in the extended Kalman filter equations given above. It can be found by taking the following partial derivatives

$$A = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial \theta} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial \theta} \\ \frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial y} & \frac{\partial f_3}{\partial \theta} \end{bmatrix}$$

When the partial derivatives are properly calculated, the A matrix can be found as

$$A = \begin{bmatrix} 1 & 0 & -v\sin\theta \\ 0 & 1 & v\cos\theta \\ 0 & 0 & 1 \end{bmatrix}$$

Since this matrix A is depends on the value of θ and v, this matrix has to be updated at each time step using the previous optimal state and current input.

Implementation

As in the simple filter and Kalman filter, some of the values have to be initialized at the beginning so that the algorithm can run. These values are the optimal state and a posteriori covariance matrix. First optimal state has been initialized to (0,0) since that is the starting point of the robot and a posteriori covariance matrix has been initialized to a 50*I (identity matrix).

The code written in Matlab to implement the Extended Kalman Filter and plot the results can be seen on the following pages.

ExtendedKalmanFilter.m

```
clear all; close all; clc;
%Initial values
Q = 0.01 * eye(3);
R = 0.01 * eye(2);
X_real(:,1) = [0 0 0];
X_optimal(:,1) = [takeMeasurement(X_real(:,1), R), 0];
P(:,:,1) = 50 * eye(3);
%Matrix values
H = [1 0 0; 0 1 0];
k=1;
while k<100
    U(:,k+1) = calculateInput(k);
    X_real(:,k+1) = moveRobot(X_real(:,k), U(:,k+1));
    X_real(:,k+1) = X_real(:,k+1) + addNoise(X_real(:,k), Q);
    Z(:,k+1) = takeMeasurement(X_real(:,k+1), R);
    A(:,:,k+1) = calculateA(X_optimal(:,k), U(:,k+1));
    X_apri(:,k+1) = moveRobot(X_optimal(:,k), U(:,k+1));
    P_apri(:,:,k+1) = A(:,:,k+1) * P(:,:,k) * transpose(A(:,:,k+1))
+ Q;
    K(:,:,k+1) = P_apri(:,:,k+1) * transpose(H) * inv( H *
P_apri(:,:,k+1) * transpose(H) + R );
    X_optimal(:,k+1) = X_apri(:,k+1) + K(:,:,k+1) * ( Z(:,k+1) - H *
X_apri(:,k+1) );
    P(:,:,k+1) = ( eye(3) - K(:,:,k+1) * H ) * P_apri(:,:,k+1);
    k = k + 1;
end
fig1 = figure;
subplot(4,4,[1 2 5 6]);
    hold on; grid on;
    plot(X_real(1,:), 'r');
    plot(X_apri(1,:), 'b--');
    xlabel('Time'); ylabel('X values'); title('X values vs time');
    legend('Real X', 'Predicted X');
subplot(4,4,[9 10 13 14]);
    hold on; grid on;
    plot(X_real(2,:), 'r');
    plot(X_apri(2,:), 'b--');
    xlabel('Time'); ylabel('Y values'); title('Y values vs time');
    legend('Real Y', 'Predicted Y');
subplot(4,4,[3 4]);
    hold on; grid on;
    plot(X_real(3,:), 'r');
    plot(X_apri(3,:), 'b--');
    xlabel('Time'); ylabel('Theta values'); title('Theta values vs
time');
    legend('Real Theta', 'Predicted Theta');
```

```
subplot(4,4,[7 8 11 12 15 16]);
    hold on; grid on;
    plot(X_real(1,:), X_real(2,:), 'r');
    plot(X_apri(1,:), X_apri(2,:), 'b--');
    xlabel('X values'); ylabel('Y values'); title('Y values vs X
values');
    legend('Real Position', 'Predicted Position');
fig2 = figure;
subplot(4,4,[1 2 5 6]);
    hold on; grid on;
    plot(X_real(1,:), 'r');
    plot(X_optimal(1,:), 'b--');
    plot(Z(1,:), 'g--');
    xlabel('Time'); ylabel('X values'); title('X values vs time');
    legend('Real X', 'Optimal X', 'Measured X');
subplot(4,4,[9 10 13 14]);
    hold on; grid on;
    plot(X_real(2,:), 'r');
    plot(X_optimal(2,:), 'b--');
    plot(Z(2,:), 'g--');
    xlabel('Time'); ylabel('Y values'); title('Y values vs time');
    legend('Real Y', 'Optimal Y', 'Measured Y');
subplot(4,4,[3 4]);
    hold on; grid on;
    plot(X_real(3,:), 'r');
    plot(X_optimal(3,:), 'b--');
    xlabel('Time'); ylabel('Theta values'); title('Theta values vs
time');
    legend('Real Theta', 'Optimal Theta');
subplot(4,4,[7 8 11 12 15 16]);
    hold on; grid on;
    plot(X_real(1,:), X_real(2,:), 'r');
    plot(X_optimal(1,:), X_optimal(2,:), 'b--');
    plot(Z(1,:), Z(2,:), 'g--');
    xlabel('X values'); ylabel('Y values'); title('Y values vs X
values');
    legend('Real Position', 'Optimal Position', 'Measured
Position');
fig3 = figure;
subplot(3,3,1);
    plot(squeeze(P_apri(1,1,:))); title('P_a_p_r_i(1,1)');
subplot(3,3,2);
    plot(squeeze(P_apri(1,2,:))); title('P_a_p_r_i(1,2)');
subplot(3,3,3);
    plot(squeeze(P_apri(1,3,:))); title('P_a_p_r_i(1,3)');
```

```
subplot(3,3,4);
    plot(squeeze(P_apri(2,1,:))); title('P_a_p_r_i(2,1)');
subplot(3,3,5);
    plot(squeeze(P_apri(2,2,:))); title('P_a_p_r_i(2,2)');
subplot(3,3,6);
    plot(squeeze(P_apri(2,3,:))); title('P_a_p_r_i(2,3)');
subplot(3,3,7);
    plot(squeeze(P_apri(3,1,:))); title('P_a_p_r_i(3,1)');
subplot(3,3,8);
    plot(squeeze(P_apri(3,2,:))); title('P_a_p_r_i(3,2)');
subplot(3,3,9);
    plot(squeeze(P_apri(3,3,:))); title('P_a_p_r_i(3,3)');
fig4 = figure;
subplot(3,3,1);
    plot(squeeze(P(1,1,:))); title('P(1,1)');
subplot(3,3,2);
    plot(squeeze(P(1,2,:))); title('P(1,2)');
subplot(3,3,3);
    plot(squeeze(P(1,3,:))); title('P(1,3)');
subplot(3,3,4);
    plot(squeeze(P(2,1,:))); title('P(2,1)');
subplot(3,3,5);
    plot(squeeze(P(2,2,:))); title('P(2,2)');
subplot(3,3,6);
    plot(squeeze(P(2,3,:))); title('P(2,3)');
subplot(3,3,7);
    plot(squeeze(P(3,1,:))); title('P(3,1)');
subplot(3,3,8);
    plot(squeeze(P(3,2,:))); title('P(3,2)');
subplot(3,3,9);
    plot(squeeze(P(3,3,:))); title('P(3,3)');
fig5 = figure;
subplot(3,2,1);
    plot(squeeze(K(1,1,:))); title('K(1,1)');
subplot(3,2,2);
    plot(squeeze(K(1,2,:))); title('K(1,2)');
subplot(3,2,3);
    plot(squeeze(K(2,1,:))); title('K(2,1)');
subplot(3,2,4);
    plot(squeeze(K(2,2,:))); title('K(2,2)');
subplot(3,2,5);
    plot(squeeze(K(3,1,:))); title('K(3,1)');
subplot(3,2,6);
    plot(squeeze(K(3,2,:))); title('K(3,2)');
```

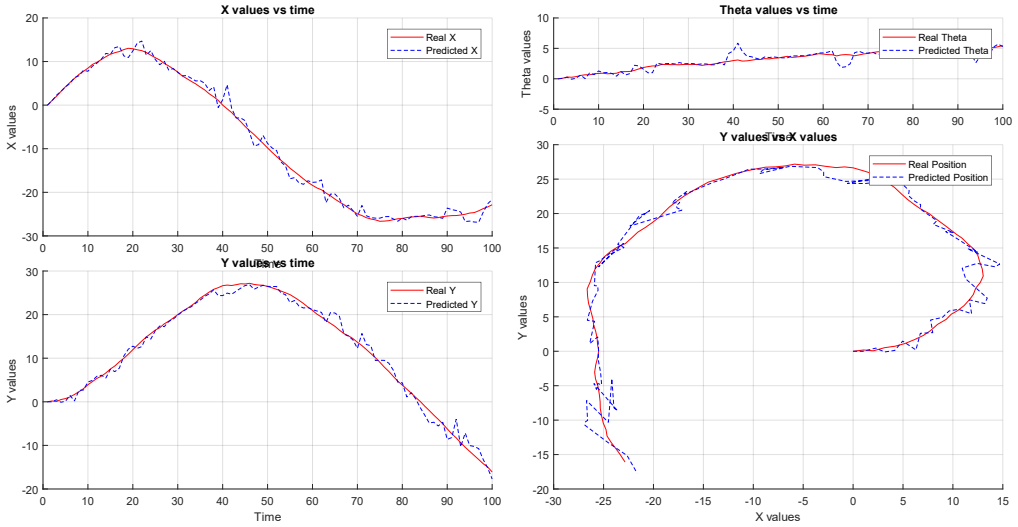
```
set(fig1, 'Position',  [50, 100, 1500, 700]);
set(fig2, 'Position',  [50, 100, 1500, 700]);
set(fig3, 'Position',  [50, 100, 1500, 700]);
set(fig4, 'Position',  [50, 100, 1500, 700]);
set(fig5, 'Position',  [50, 100, 1500, 700]);

%saveas(fig1, 'EKF_3_1.svg');
%saveas(fig2, 'EKF_3_2.svg');
%saveas(fig3, 'EKF_3_3.svg');
%saveas(fig4, 'EKF_3_4.svg');
%saveas(fig5, 'EKF_3_5.svg');

function X_real_new = moveRobot(X_real, U)
X_real_new(1) = X_real(1) + U(1) * cos(X_real(3));
X_real_new(2) = X_real(2) + U(1) * sin(X_real(3));
X_real_new(3) = X_real(3) + U(2);
end

function U = calculateInput(k)
ang = k*2*pi/100;
xdir = cos(ang);
ydir = sin(ang);
U(1) = sqrt(xdir^2 + ydir^2);
U(2) = 2*pi/100;
end

function Z = takeMeasurement(X_real, Q)
d = sqrt(X_real(1)^2 + X_real(2)^2);
theta = atan2(X_real(2), X_real(1));
real_val = [d; theta];
meas_val = real_val + sqrt(Q) * randn(size(real_val));
Z(1) = meas_val(1) * cos(meas_val(2));
Z(2) = meas_val(1) * sin(meas_val(2));
end

function w = addNoise(X, Q)
w = sqrt(Q) * randn(size(X));
end

function A = calculateA(X, U)
A = [1 0 -U(1)*sin(X(3));...
    0 1 U(1)*cos(X(3));...
    0 0 1];
end
```

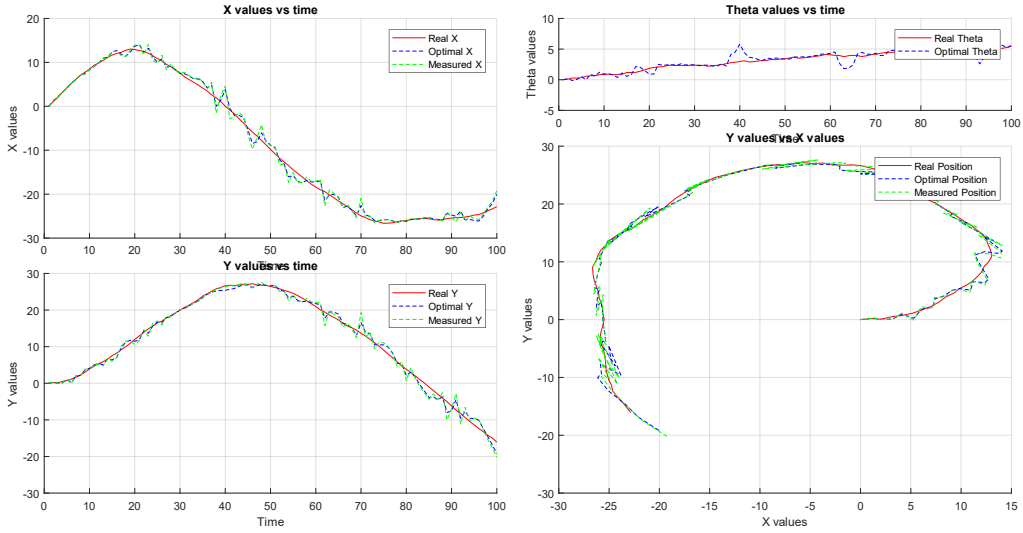
Results and Discussion

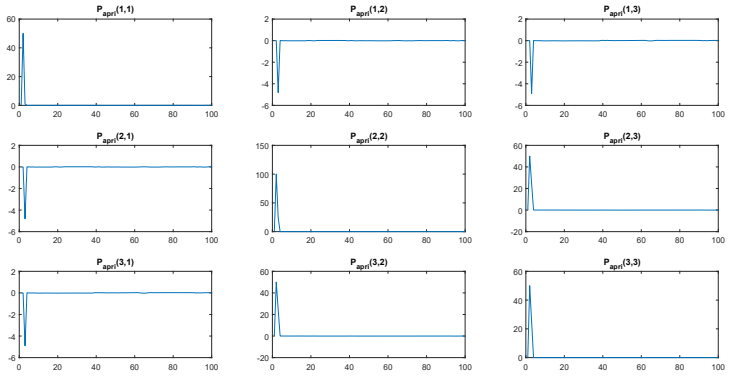
Just like in the Kalman Filter case, only process noise covariance matrix Q and measurement noise covariance matrix R have been changed between the cases.

Case 1

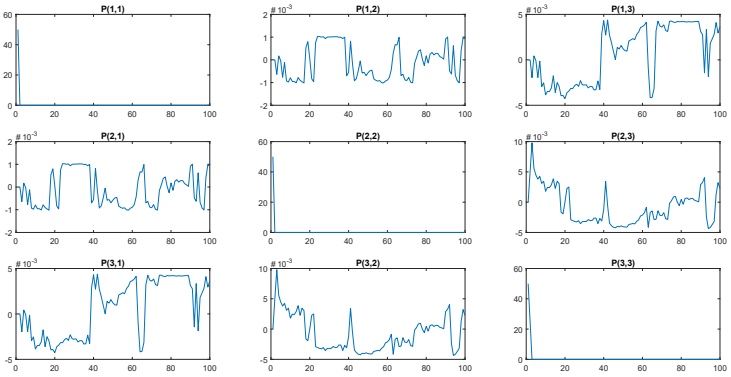
$$Q = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix} \qquad R = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}$$

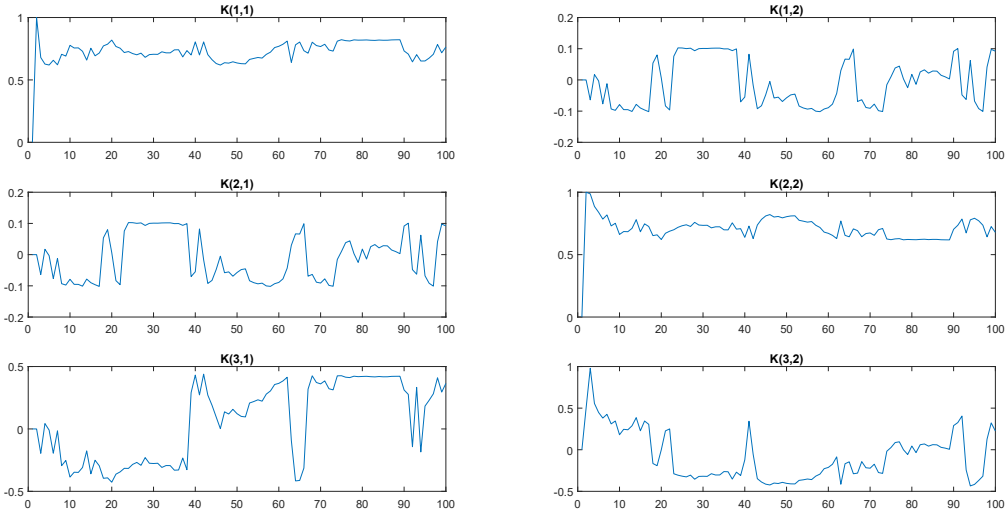






In the first case the same values have been given to the process noise covariance matrix Q and to the measurement noise covariance matrix R just like in the Kalman Filter case. Since most conclusions about which information source is believed more are very similar to the Kalman Filter case, they will not be discussed here. However, the different aspects from the Kalman Filter will be discussed. First, it can be seen that there is more noise in both x and y directions caused by the non-linearity of the system. This is the case because Kalman Filter does not promise optimal results for non-linear systems. Another result is that all the values of both the a priori error covariance matrix and a posteriori error covariance matrix converge to 0. Finally, Kalman Gain matrix values do not converge to a value like in the Kalman Filter. K (1,1) and K (2,2) oscillate highly around 0.7, while the other entries oscillate around 0.

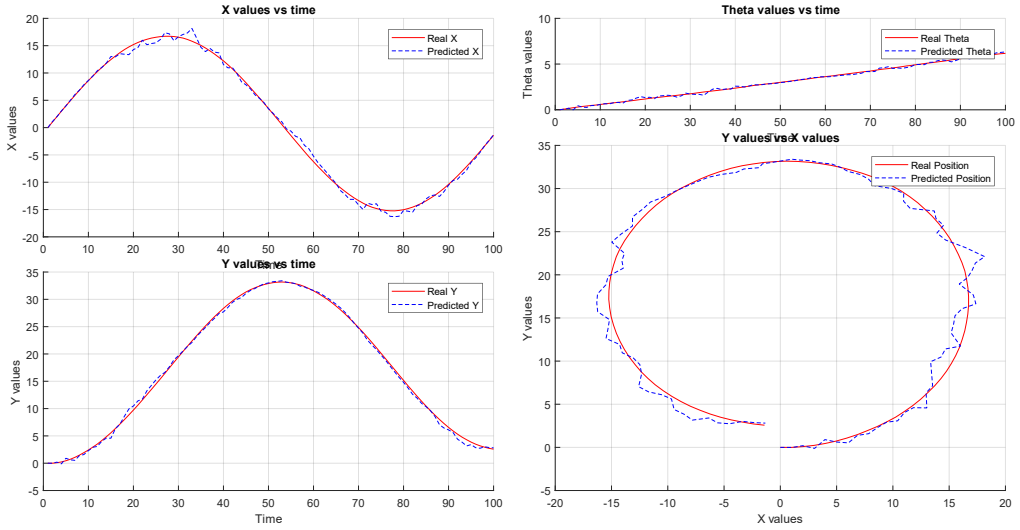


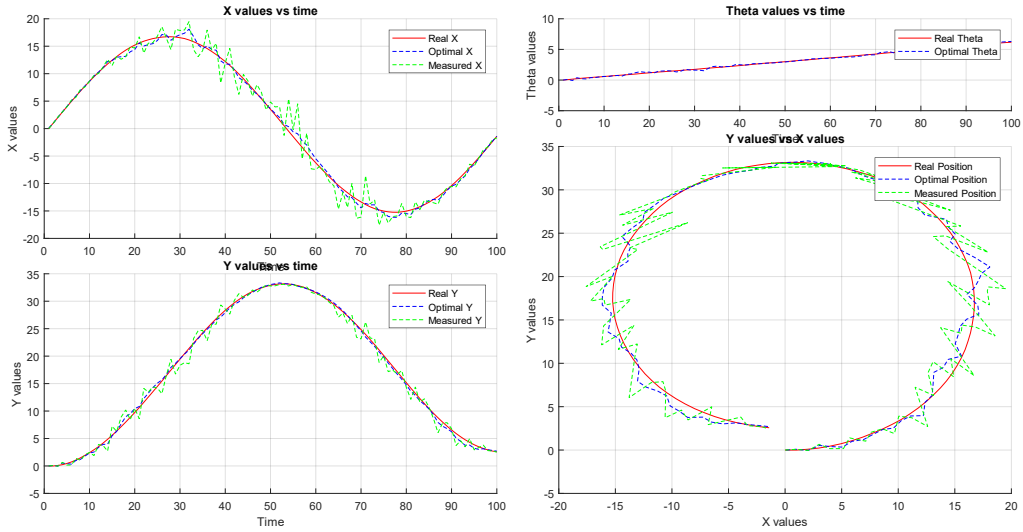


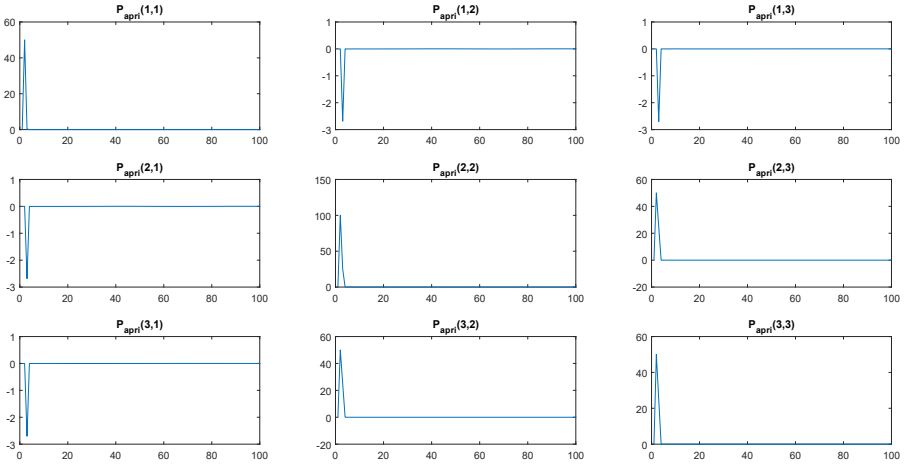
Very similar remarks can be made for both case 2 and 3, so in order to avoid repetition, only the results will be shown.

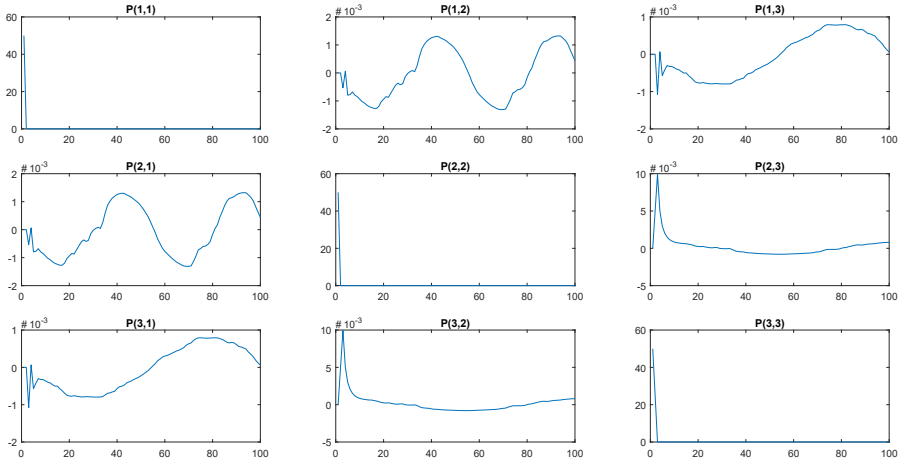
Case 2

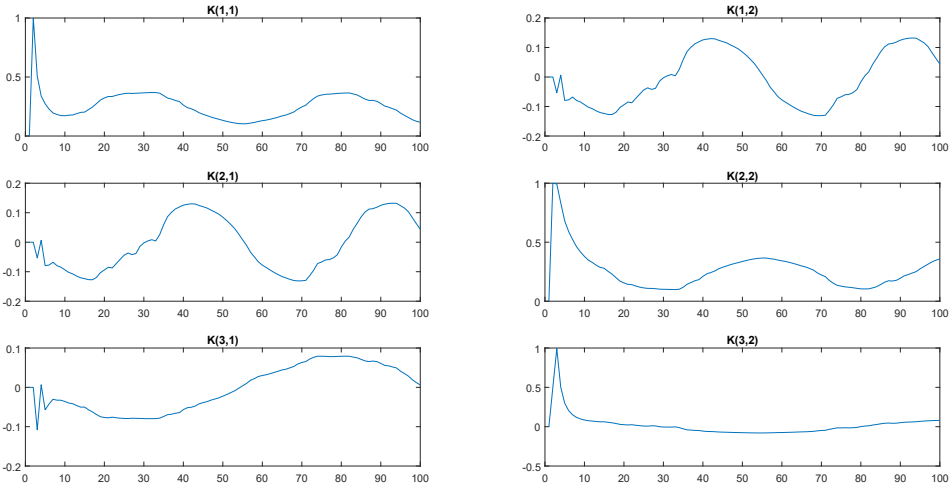
$$Q = \begin{bmatrix} 0.0001 & 0 & 0 \\ 0 & 0.0001 & 0 \\ 0 & 0 & 0.0001 \end{bmatrix} \qquad R = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}$$





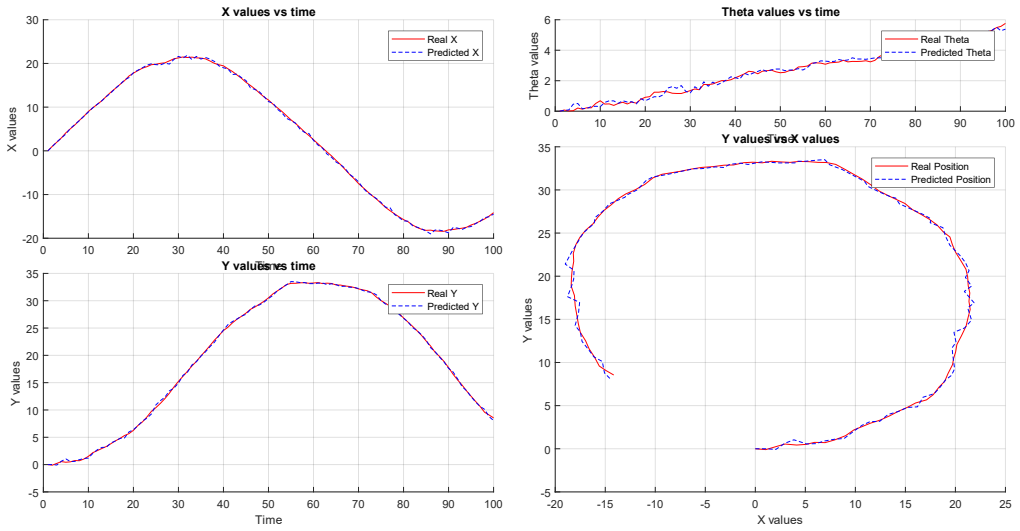

P_apri(1,1)
P_apri(1,2)
P_apri(1,3)
P_apri(2,1)
P_apri(2,2)
P_apri(2,3)
P_apri(3,1)
P_apri(3,2)
P_apri(3,3)

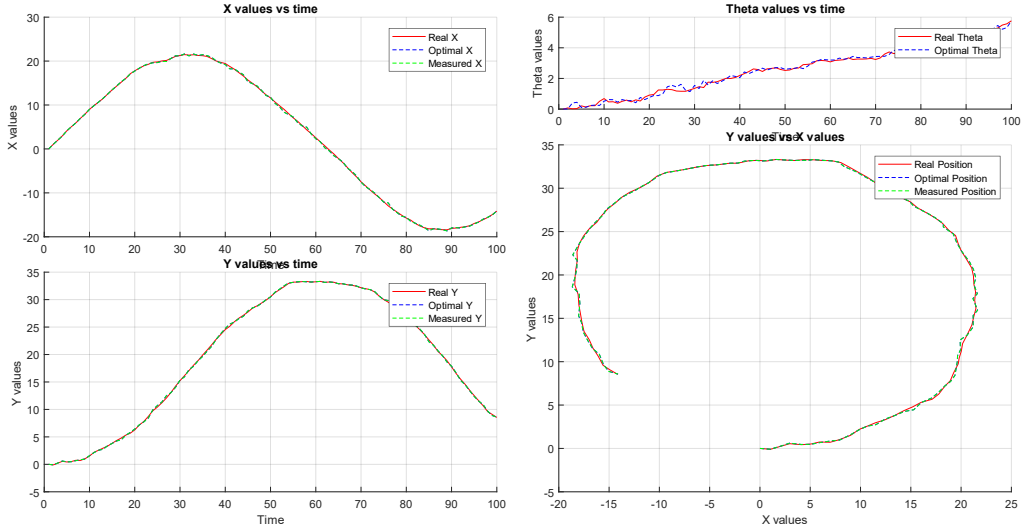

P(1,1)
P(1,2)
P(1,3)
P(2,1)
P(2,2)
P(2,3)
P(3,1)
P(3,2)
P(3,3)
# 10^-3

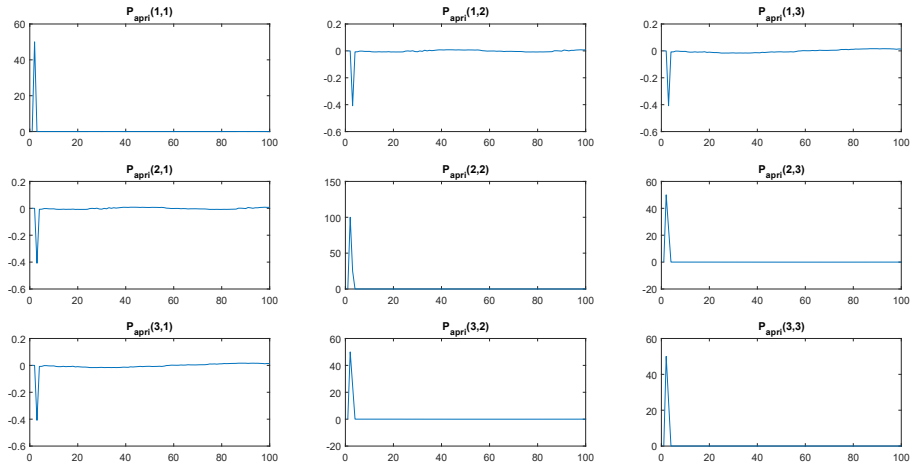

K(1,1)
K(1,2)
K(2,1)
K(2,2)
K(3,1)
K(3,2)

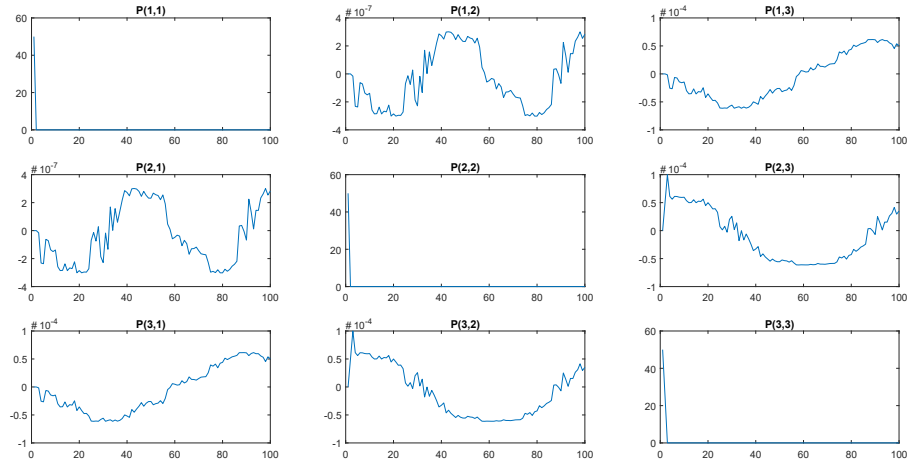
Case 3

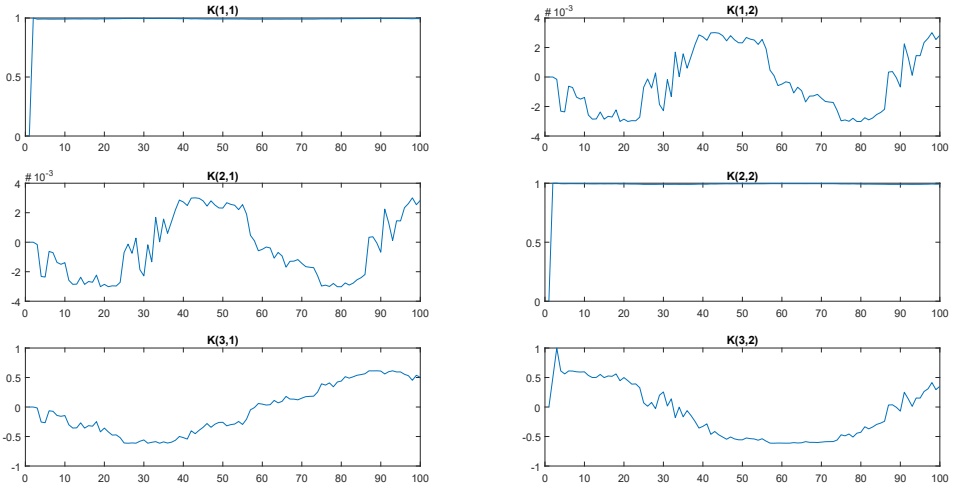
$$Q = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix} \qquad R = \begin{bmatrix} 0.0001 & 0 \\ 0 & 0.0001 \end{bmatrix}$$






P_apri(1,1)
P_apri(1,2)
P_apri(1,3)
P_apri(2,1)
P_apri(2,2)
P_apri(2,3)
P_apri(3,1)
P_apri(3,2)
P_apri(3,3)


P(1,1)
P(1,2)
P(1,3)
P(2,1)
P(2,2)
P(2,3)
P(3,1)
P(3,2)
P(3,3)


K(1,1)
K(1,2)
K(2,1)
K(2,2)
K(3,1)
K(3,2)