
# MECHANICAL ENGINEERING FACULTY

## Sensor Systems

## Semestral Work

## Drone Attitude Estimation with Kalman Filters

**Student:Caner Emrecan**

**Sofiene Ayadi**

Matrix A (State Transition Matrix):

A is a square matrix that connects the system's present state to its upcoming state.

It outlines the state's temporal evolution.

Given that the system being represented assumes a fixed position, A is an identity matrix.

Matrix H (Measurement Matrix):

H is a matrix that connects the state to readings made by the "inductive position sensors".

The status is mapped to the anticipated measures.

Since the measurements exactly match the state variables, H is an identity matrix.

Matrix Q (Process Noise Covariance):

Q stands for the covariance matrix of the process noise, which is responsible for the ambiguity and errors in the dynamics of the system.

The state transition's uncertainty is quantified.

To simulate low process noise, Q is a diagonal matrix with small values

Matrix R (Measurement Noise Covariance):

The measurement noise covariance matrix, or R, encapsulates the degree of uncertainty in the sensor measurements.

It rates the measurement's precision and dependability.

R is a diagonal matrix with small values to simulate low measurement noise.

Matrix P (Error Covariance):

The error covariance matrix, or P, reflects the degree of uncertainty in the estimated state.

Based on the measurements and the prior state estimate, it expresses the degree of confidence in the estimated state.

P represents the error covariance matrix, which characterizes the uncertainty in the estimated state.

These matrices are fundamental components of the Kalman filter algorithm and play a crucial role in estimating the true state of a system based on noisy measurements. By tuning these matrices, you can adjust the filter's performance and adapt it to different system characteristics and noise levels.

**MATLAB CODE**

```
A = eye(3);
H = eye(3);


Q = eye(3) * 0.1;
R = eye(3) * 1000;


P = eye(3);


dronePositionDB = table('Size', [0 4], 'VariableTypes', {'datetime', 'double',
'double', 'double'}, ...
    'VariableNames', {'TimeStamp', 'X', 'Y', 'Z'});
```

```
fileID = fopen('3Ddata.txt','r');
formatSpec = '%*s %f %f %f';
sizeA = [3 Inf];
A_pos = fscanf(fileID,formatSpec,sizeA);


fileID = fopen('3Ddata.txt','r');
formatSpec = '%c';
sizeA = [1 Inf];
A_time = fscanf(fileID,formatSpec,sizeA);


colon = 0;
i = 1;
j = 1;


while i < length(A_time)
    if (A_time(i) == ':')
        colon = colon + 1;
    end
    if colon == 2
        colon = 0;
        time(j) = str2double(A_time(i+1:i+6));
        j = j + 1;
    end
    i = i + 1;
end


for i = 1:length(time)-1
    dt(i) = time(i+1)-time(i);
end


dt(dt<0) = dt(dt<0)+60;


len = length(A_pos);
x0 = A_pos(1,1);
y0 = A_pos(2,1);
z0 = A_pos(3,1);
x_hat=[x0;y0;z0];
x_pos = A_pos(1,2:len);
y_pos = A_pos(2,2:len);
z_pos = A_pos(3,2:len);
```

```
truePositionsX = x_pos;
truePositionsY = y_pos;
truePositionsZ = z_pos;


measuredPositionsX = truePositionsX + rand*0.5;
measuredPositionsY = truePositionsY + rand *0.1;
measuredPositionsZ = truePositionsZ + rand*0.8 ;


for i = 1:length(measuredPositionsZ)

    x_m = measuredPositionsX(i);
    y_m = measuredPositionsY(i);
    z_m = measuredPositionsZ(i);



    x_hat_minus = A * x_hat;
    P_minus = A * P * A' + Q;



    K = P_minus * H' * inv(H * P_minus * H' + R);
    x_hat = x_hat_minus + K * ([x_m; y_m; z_m] - H * x_hat_minus);
    P = (eye(3) - K * H) * P_minus;



    newRow = table(datetime('now'), x_hat(1), x_hat(2), x_hat(3),
'VariableNames', dronePositionDB.Properties.VariableNames);
    dronePositionDB = [dronePositionDB; newRow];
end


time = double(time);
truePositionsX = double(truePositionsX);
truePositionsY = double(truePositionsY);
truePositionsZ = double(truePositionsZ);
measuredPositionsX = double(measuredPositionsX);
measuredPositionsY = double(measuredPositionsY);
measuredPositionsZ = double(measuredPositionsZ);


[timeGrid, trueGridX] = meshgrid(time, truePositionsX);
[timeGrid, trueGridY] = meshgrid(time, truePositionsY);
[timeGrid, trueGridZ] = meshgrid(time, truePositionsZ);


figure;
```

```
plot(dronePositionDB.TimeStamp, double(dronePositionDB.X), 'r', 'LineWidth',
1.5);
hold on;
plot(dronePositionDB.TimeStamp, measuredPositionsX, 'g', 'LineWidth', 1.5);
xlabel('Time');
ylabel('Position X');
title('Measured Positions vs Filtered Positions');
legend('X FILTERED', 'X MEASURED');
grid on;
hold off;


figure;
plot(dronePositionDB.TimeStamp, double(dronePositionDB.Y), 'r', 'LineWidth',
1.5);
hold on;
plot(dronePositionDB.TimeStamp, measuredPositionsY, 'g', 'LineWidth', 1.5);
xlabel('Time');
ylabel('Position Y');
title('Measured Positions vs Filtered Positions');
legend('Y FILTERED', 'Y MEASURED');
grid on;
hold off;


figure;
plot(dronePositionDB.TimeStamp, double(dronePositionDB.Z), 'r', 'LineWidth',
1.5);
hold on;
plot(dronePositionDB.TimeStamp, measuredPositionsZ, 'g', 'LineWidth', 1.5);
xlabel('Time');
ylabel('Position Z');
title('Measured Positions vs Filtered Positions');
legend('Z FILTERED', 'Z MEASURED');
grid on;
hold off


figure;
plot3(measuredPositionsX,measuredPositionsY,measuredPositionsZ,'r');
hold on;
plot3(dronePositionDB.X,dronePositionDB.Y,dronePositionDB.Z,'b');


legend('Measured Position', 'Filtered Position');
xlabel('X Position (meters)');
ylabel('Y Position (meters)');
zlabel('Z Position (meters)');
grid on;
```

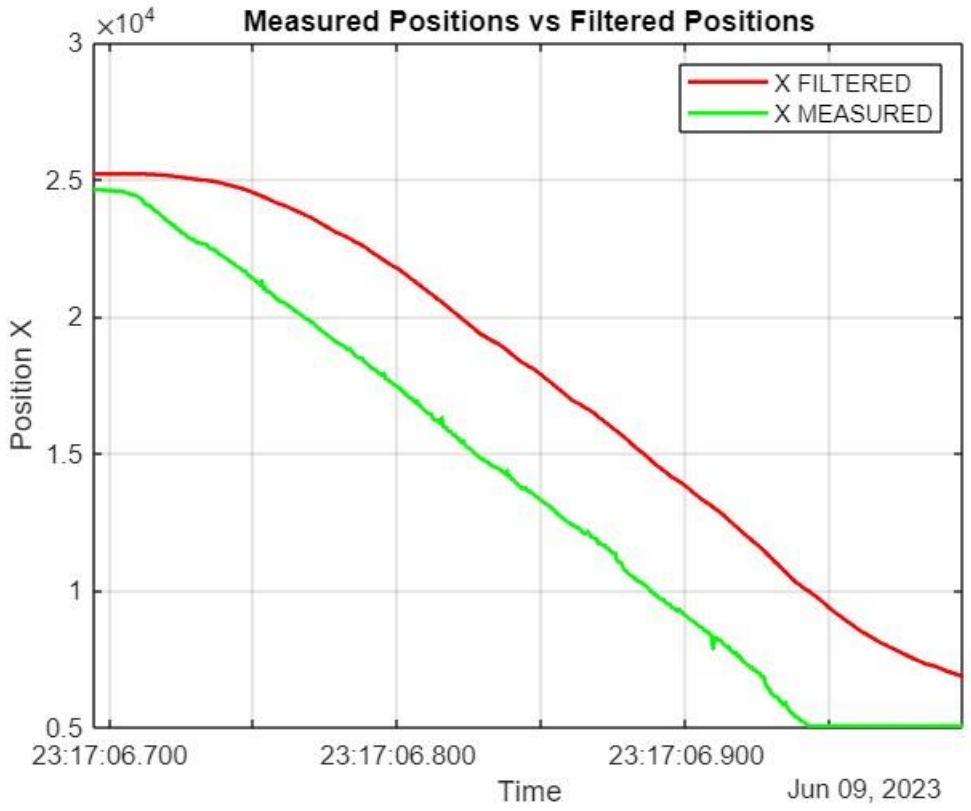


Figure1:Meausred Positions and Filtered Positions of X

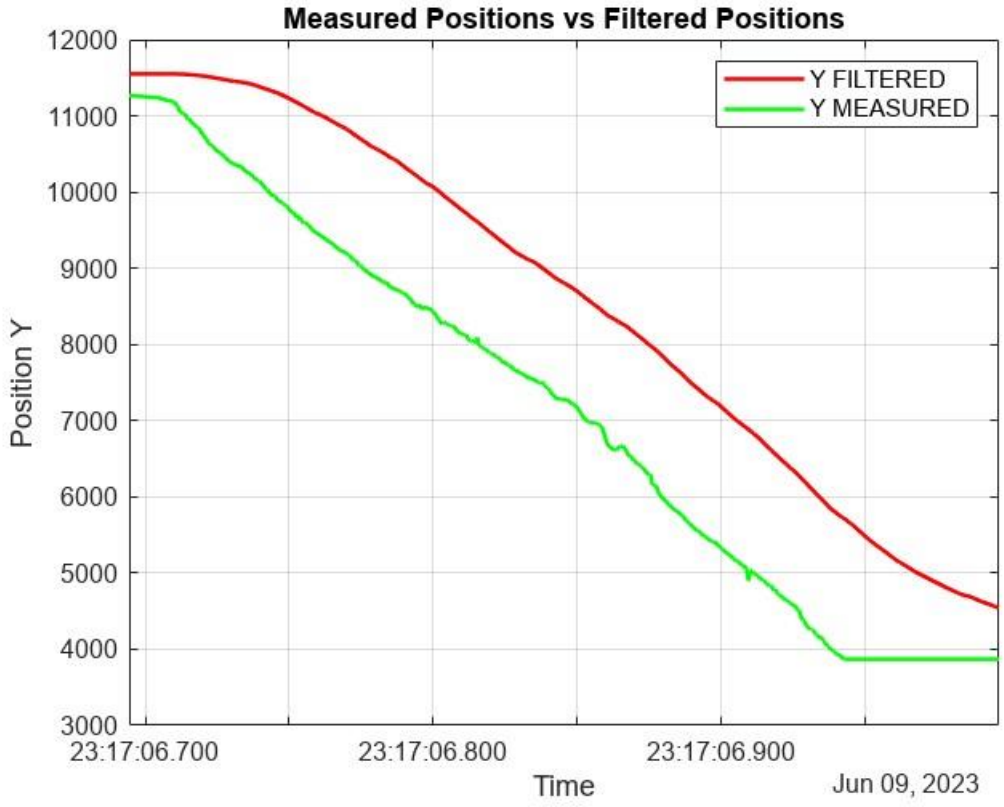

Figure2:Meausred Positions and Filtered Positions of Y

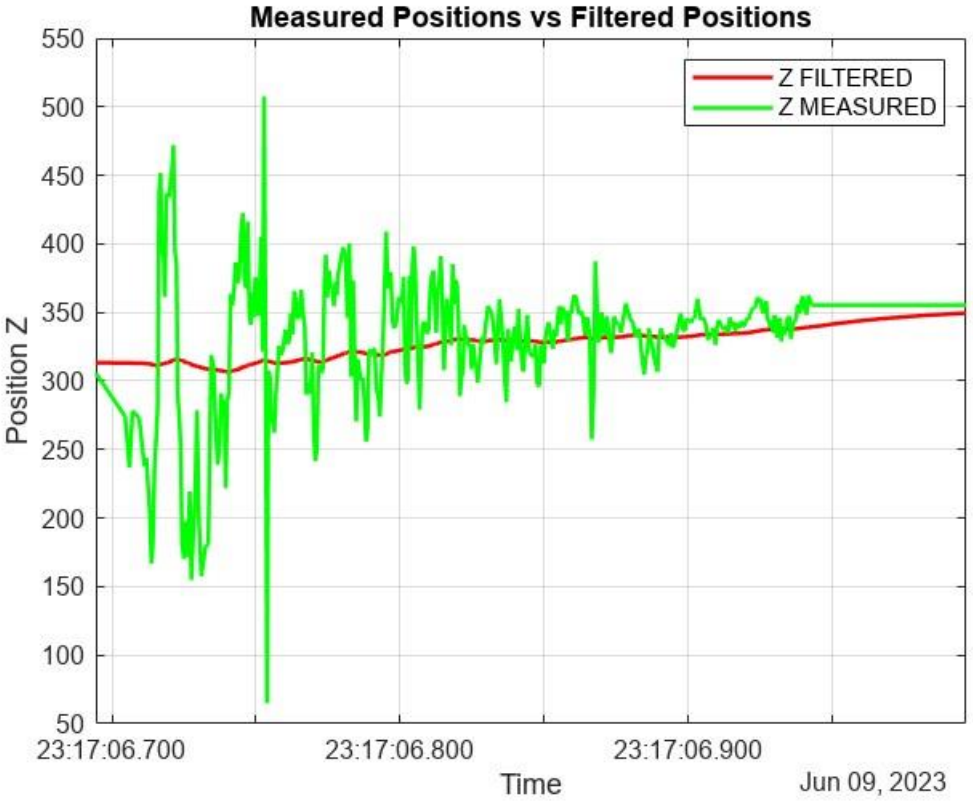


Figure3:Meausred Positions and Filtered Positions of Z

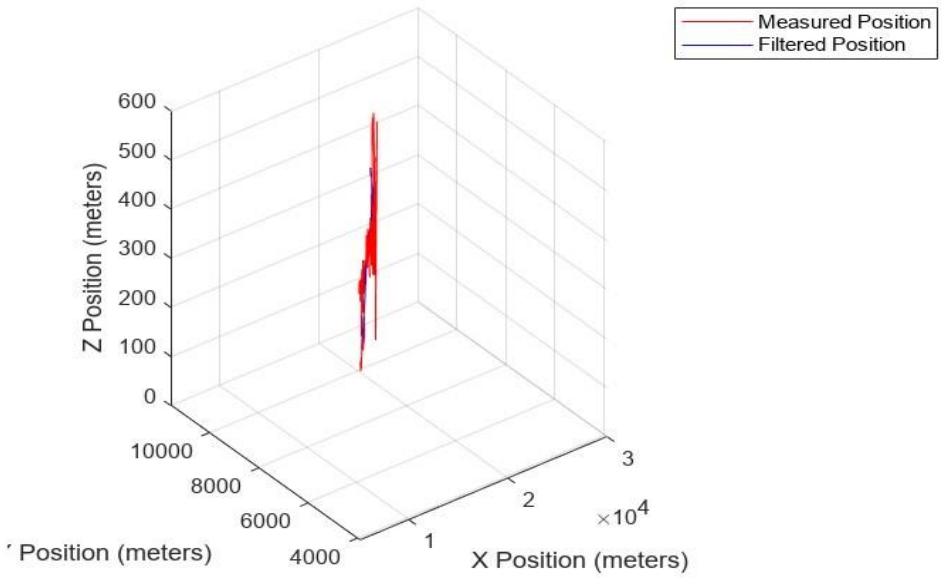


Figure4:Meausred 3D Positions and Filtered Positions of all axes