Juan D Franco and David Goldhirsch Project Team 9
EECE 212- Linear Algebra and Engineering Programming Prof. Zhang

<u>Final Project Report:</u>

I. Dynamic systems with disturbances

a. Purpose:

Our task was to use a state space model to characterize the dynamics of a system. For part one, we were to utilize the simple state model from previous labs and adapt it to more general scenarios. In addition, we were to introduce a sense of randomness to account for extraneous variables. Thus, the final equation was:

$$\mathbf{q[n+1] = Aq[n] + Bx[n]}$$

The components of the equation are as follows: q[n] is the state element, A is the transition matrix, B is the input matrix and x[n] is random vectors from the Gaussian distribution.

b. How Our Function Works

To create the equation above first, we created the following user inputs. The user will input the number of state elements (N), the time between states (T), the initial positions and velocities for the x and y direction and the randomness vectors for x and y (σx,σy). The q matrix, the position vectors and the random vectors are all initialized with the zeros functions; this is done to create the correct number of elements for each variable. We set the first element of the position vector to be the inputted initial states. We do the same for the q matrix. We set up the q matrix such that element one is the x position, position two is the x velocity, third position is the y position and fourth is the y velocity. Next, we implemented a state transition matrix such that it is a constant velocity model. Then, we created a random variable with 4 rows, one for each position and velocity state that are normally distributed using randn. We set the first random values to be part of the random vector matrix. We created B such that when multiplied by x, it will only affect the velocity. Finally, we have calculated all the values, so we placed them into a for-loop such that the next state depends on the previous state. In summary, we first created randomness by creating a set of random vectors that multiply by a B matrix, so they only affect velocity. Then we multiplied this answer by the constant velocity state transition matrix with the previous q.

c. Code:

<u>**The Gen_state function:**</u>

```
function [q] = gen_state(N,T,Px,Py,Vx,Vy,sigx,sigy)

    px = zeros(1,N+1); %Make array for values of x
    py = zeros(1,N+1); %Make array for values of y
    q = zeros(4,N+1); %Make matrix for values of q
    x_n = zeros(4,N+1); %Make matrix for values of x_n
    px(:,1) = Px; %Initialize first value of array for x
    py(:,1) = Py; %Initialize first value of array for y

    q(:,1) = [px(1);Vx;py(1);Vy]; %Initialize first values of q matrix

    A = [1 T 0 0;0 1 0 0;0 0 1 T;0 0 0 1]; %State Transition matrix

    X = randn([4 (N+1)]); %Random values
    x_n(:,1) = [X(1,1);X(2,1);X(3,1);X(4,1)]; %Initialize first values
of x_n matrix

    B = [0 0 0 0;0 sigx 0 0;0 0 0 0;0 0 0 sigy]; %Matrix to affect the
x and y velocities

    for n = 2:N+1
        q(:,n) = A*q(:,n-1) + B*x_n(:,n-1); %Equation for time-varying
state vector and noise to the velocity
        x_n(:,n) = [X(1,n);X(2,n);X(3,n);X(4,n)]; %Goes to next value
of x_n
    end

end
```

<u>Plotting the Gen State Function for Figure 1:</u>

```
      %%
      N = 60; %Amount of time/points
      T = 1;
      Px = 0; %Initial value of x
      Py = 0; %Initial value of y
      Vx = 100; %Initial velocity of x
      Vy = 100; %Initial velocity of y
      sigx = 0;
      sigy = 0;

      [q] = gen_state(N,T,Px,Py,Vx,Vy,sigx,sigy);

      %%
      figure
      plot(q(1,:), q(3,:), '-r*')
      hold on
      xlabel('x position')
      ylabel('y position')
      str = sprintf('2D Aircraft Trajectory with Sigma-x = %d and
      Sigma-y = %d', sigx, sigy);
      title(str)
      hold off
```

<u>Plotting the Gen State Function for Figure 2:</u>

```
      %%
      N = 60; %Amount of time/points
      T = 1;
      Px = 0; %Initial value of x
      Py = 0; %Initial value of y
      Vx = 100; %Initial velocity of x
      Vy = 100; %Initial velocity of y
      sigx = 5;
```

```
sigy = 10;

[q] = gen_state(N,T,Px,Py,Vx,Vy,sigx,sigy);

%%
figure
plot(q(1,:), q(3,:), '-r*')
hold on
xlabel('x position')
ylabel('y position')
str = sprintf('2D Aircraft Trajectory with Sigma-x = %d and
Sigma-y = %d', sigx, sigy);
title(str)
hold off
```

d. *How We Plotted Our Functions*

To create Figure 1, first we chose the desired inputs. We desired 61 states, from 0 to 60 seconds with a time difference of 1 second. Our initial position was (0,0) and the velocity was (100,100). For Figure 1, there was no randomness so sigx and sigy were 0's. For Figure 2 and 3, sigx was made 5 and sigy was 10. After creating the figure, we plotted the position over time by using the first row as the x position and the third row as the y position. Finally, the title was created such that it would display the sigma values, and the plot would be complete when run was pressed.

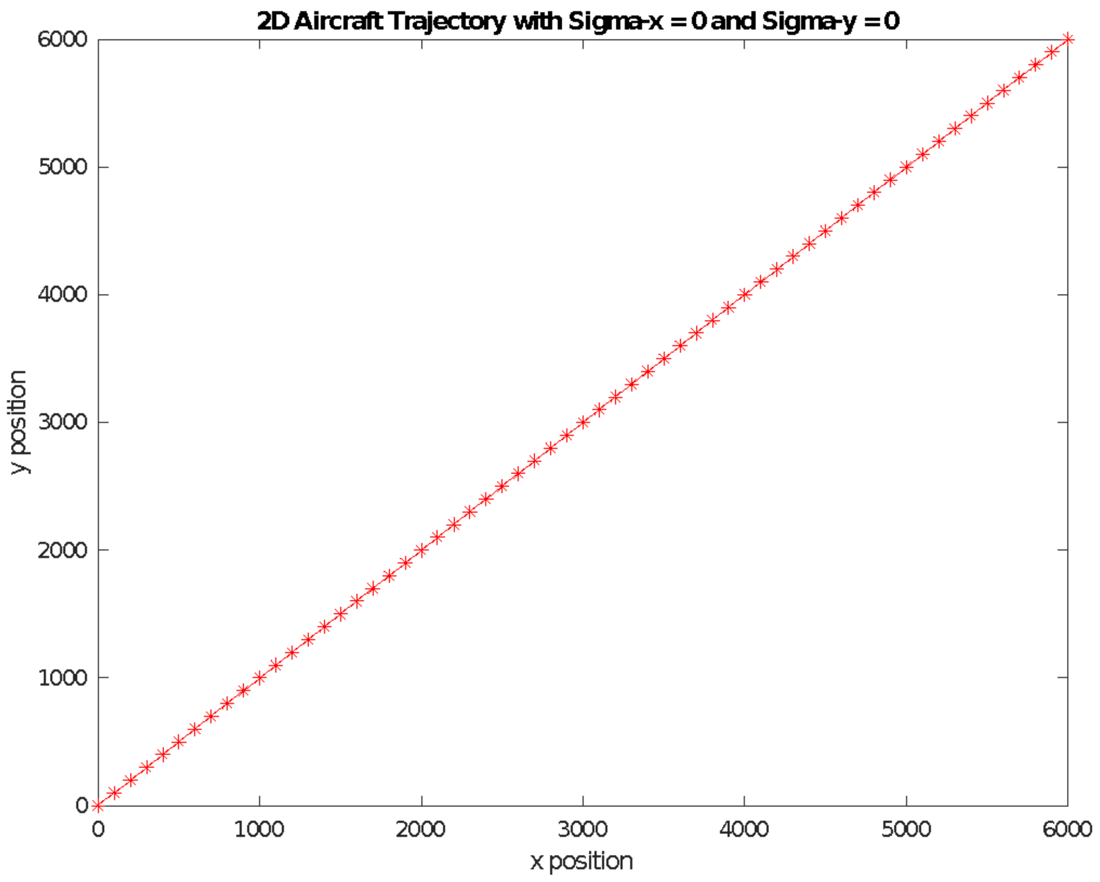

Figure 1: Graphical representation of a Constant Velocity Model with no randomness

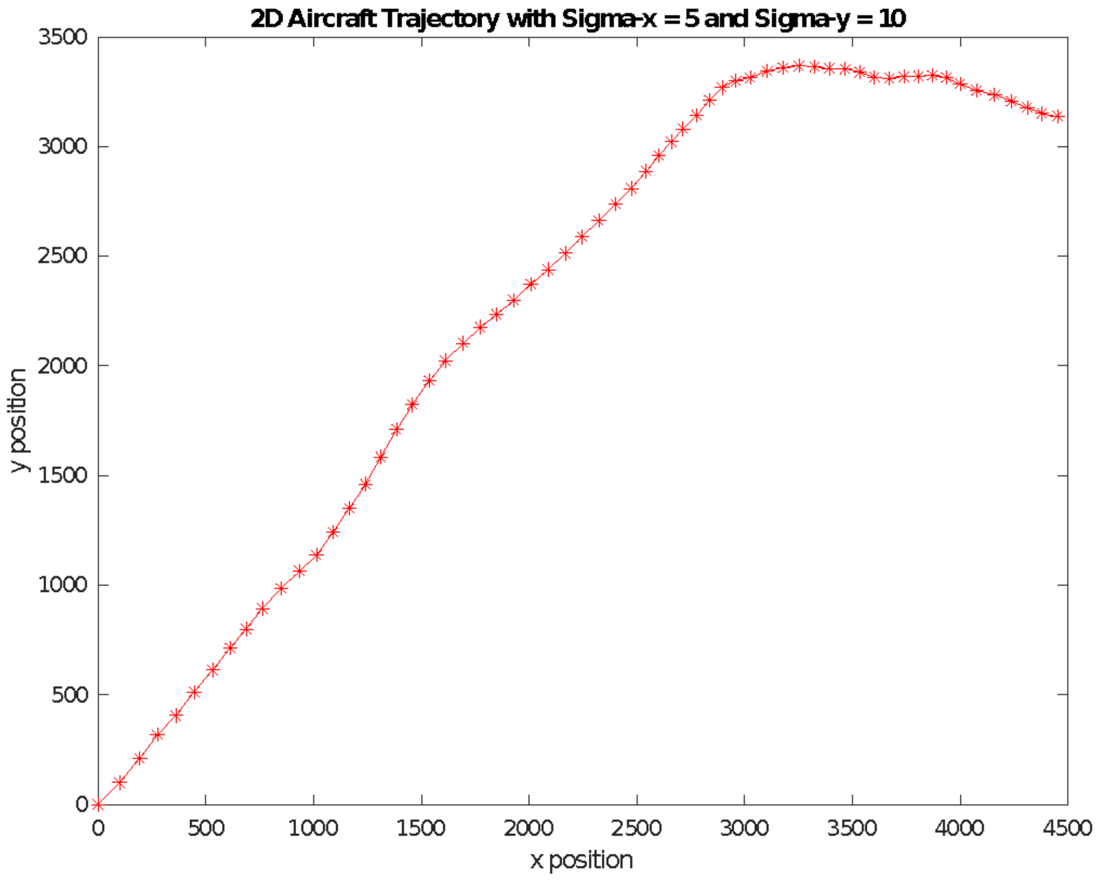

Figure 2: Graphical representation of a Constant Velocity Model with sigx=5 and sigy=10

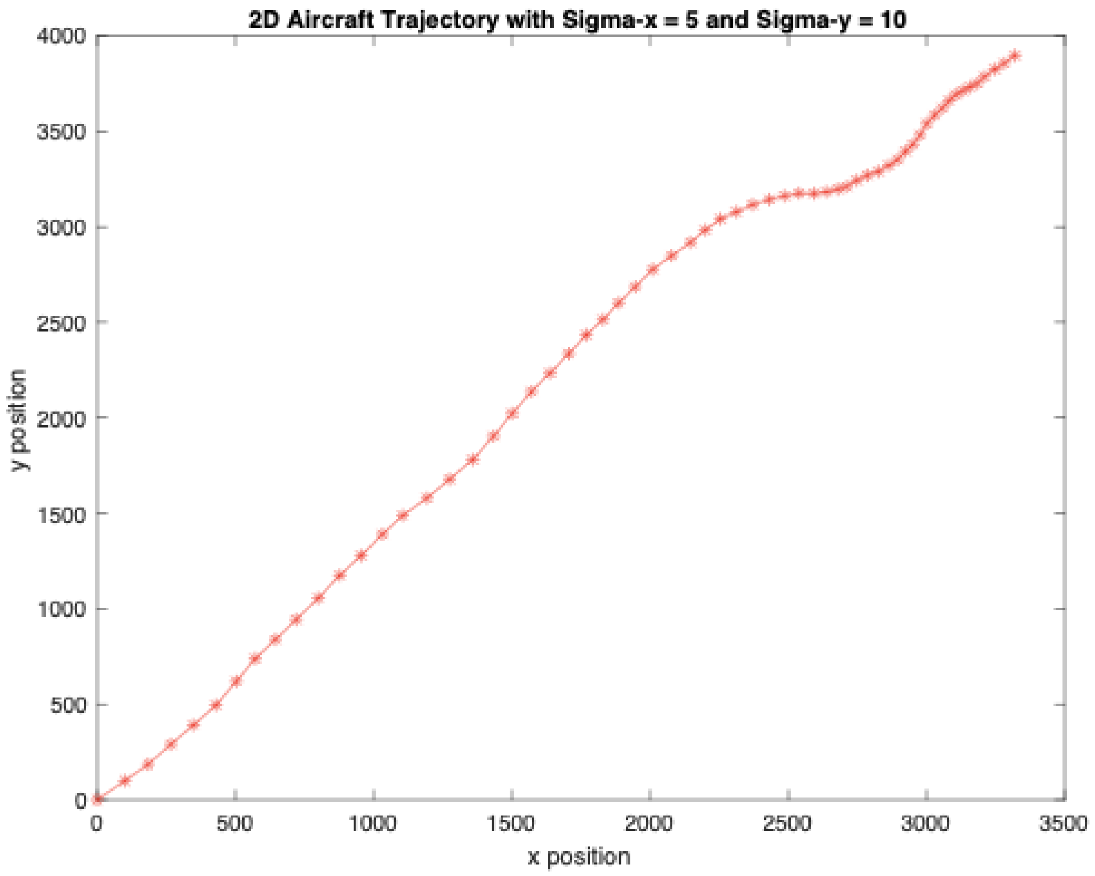

Figure 3: Graphical representation of a Constant Velocity Model with sigx=5 and sigy=10

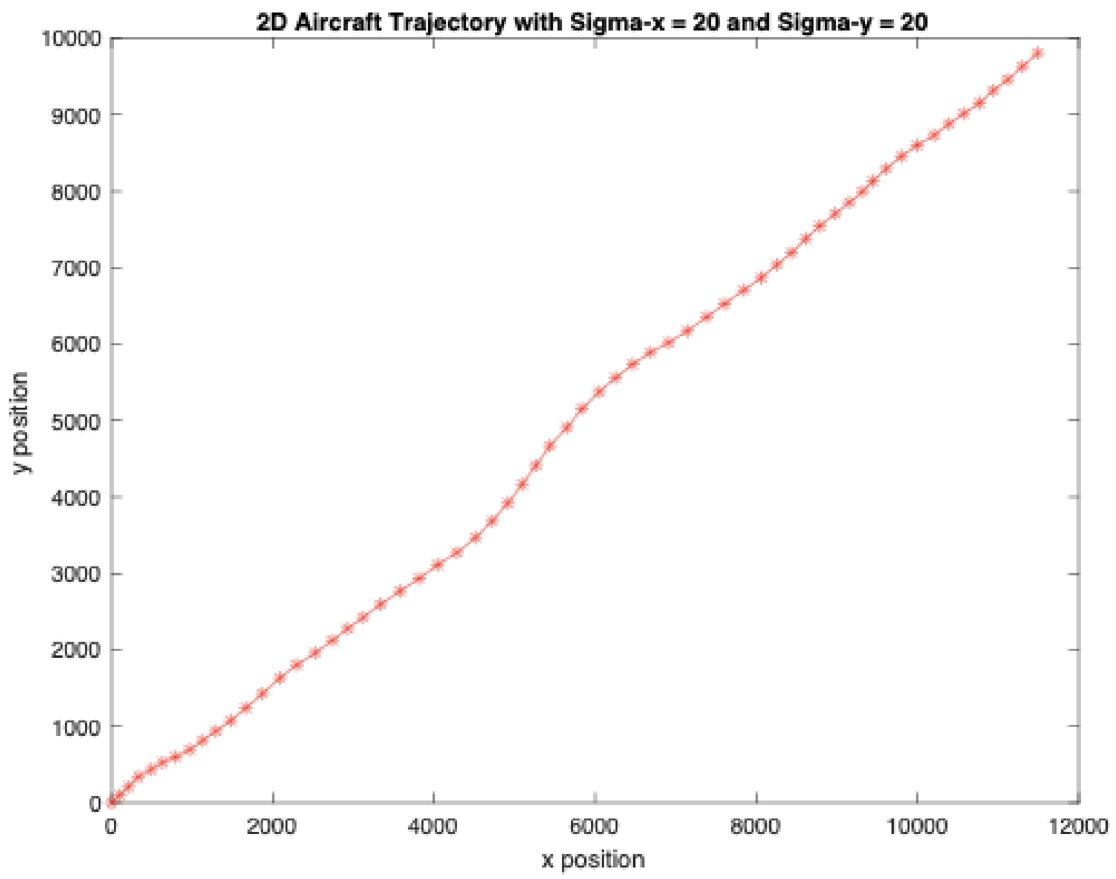

Figure 4: Graphical representation of a Constant Velocity Model with sigx=20 and sigy=20

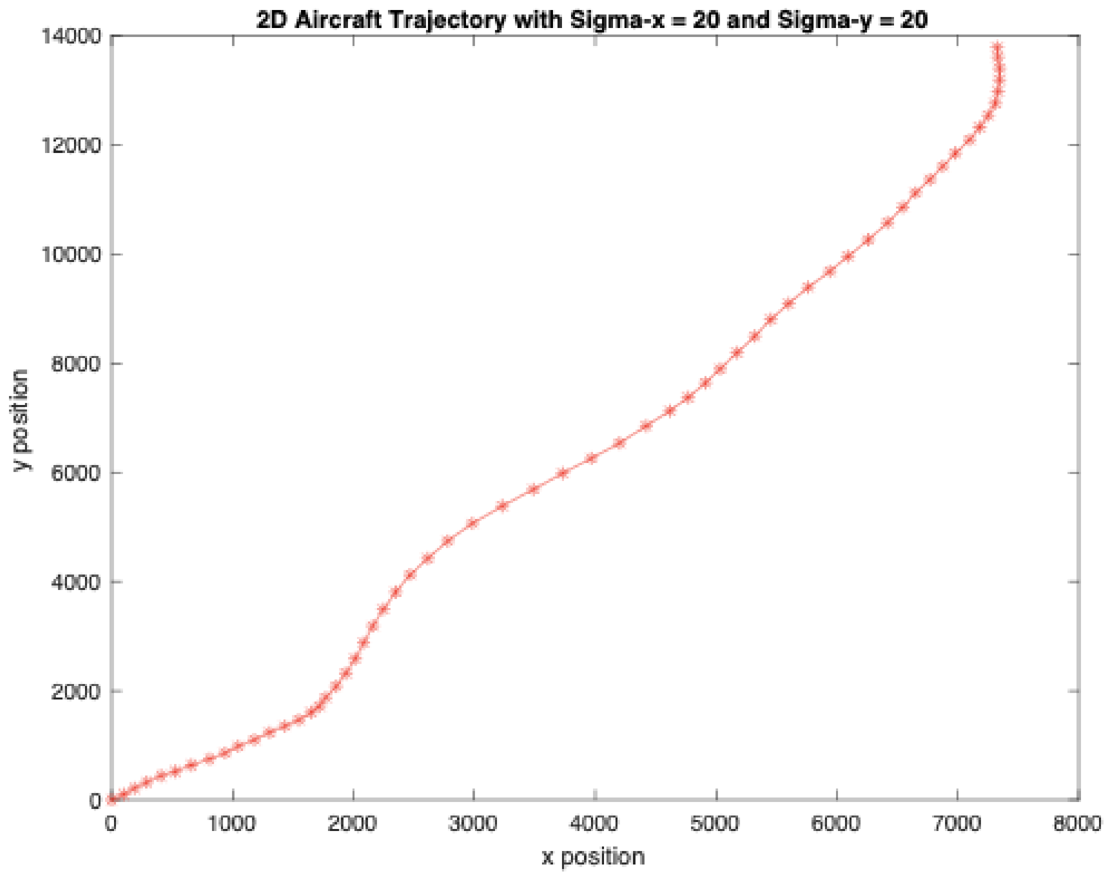

Figure 5: Graphical representation of a Constant Velocity Model with sigx=20 and sigy=20

e. *Verification of the Function*

To verify our results, we relied on both our graphs and our own calculations. We checked that our lines were truly random by using the same sigma values. Since the graphs were different each time, we knew our states were being multiplied by the random vectors. In addition, we compared how changing the sigma values affected the shape of the line. Our line became more random and less straight as we raised sigmas, so we were quite sure of our answers. Finally, we calculated the values by hand, using randomly generated normally distributed values and when plotted our graphs looked very similar (since they were random, it couldn't be exact).

II. Using Sensors to Measure States

a. *Purpose*

We seek to use sensors to measure our systems, but sensors inherently produce errors. Thus, we will model a system that characterizes the error. This equation will incorporate a measuring model with a noise factor:

$$\mathbf{y[n] = Cq[n]+Dw[n]}$$

y[n] will be the sensor measurements, C is the sensor matrix, q is the current state, D is the diagonal matrix that holds the standard deviations of noise, and w is a set of Guassian distributed random vectors with a standard deviation of 1.

b. *How Our Function Works*

First, we set the y and w to have p (number of sensors) elements with the zeros function. Next, we use randn to randomize normally for W. We set the input values equal to the first state. Then, we diagonalize d (the standard deviations for noise), so each element multiplies with only one random value. Finally, we implement a for loop to calculate the sensor readings an N number of times.This function will add a sense of noise to each q state to simulate a sensor measurement environment.

c. *Code*

The Measure_state function:

```
function [y] = measure_state(p,N,C,d,q)
    % p = number of sensors
    % N = number of whatever
    % C = pxN sensor matrix that forms linear combinations of the state
elements
    % d = Noise vector

    % Returns:
    % y = sensor measurements

    y = zeros(p,1); %Make matrix for values of y

    w = zeros(p,1); %Make matrix for values of w

    W = randn([4 (N+1)]); %Random values
    w(:,1) = [W(1,1);W(2,1);W(3,1);W(4,1)]; %Initialize first values of
w matrix

    D = diag(d); %Identity matrix with d values diagonally placed
    y(:,1) = C*q(:,1) + D*w(:,1); %Initialize first values of y matrix


    for n = 2:N+1
        w = [W(1,n);W(2,n);W(3,n);W(4,n)]; %Goes to next value of w
        y(:,n) = C*q(:,n) + D*w; %Equation for time-varying state
vector and noise to the position and velocity
    end

end
```

Plotting the Measure_State_Function:

```
%%
N = 60; %Amount of time/points
T = 1;
Px = 0; %Initial value of x
Py = 0; %Initial value of y
Vx = 100; %Initial velocity of x
Vy = 100; %Initial velocity of y
sigx = 1;
sigy = 10;

[q] = gen_state(N,T,Px,Py,Vx,Vy,sigx,sigy);

%%
p = 4;
N = 60; %Amount of time/points
C = eye(p); %Identity matrix
d = [25;10;25;10]; %Noise for the position and velocity of x and y

[y] = measure_state(p,N,C,d,q);

%%
figure
plot(q(1,:),q(3,:), '-r*','DisplayName','Real Position')
hold on
plot(y(1,:), y(3,:), '-bo','DisplayName','Measured Position')
title({('Sensor Measurement of a 2D Aircraft Trajectory')
['Sigma-x = ' num2str(sigx) ' and Sigma-y = ' num2str(sigy)]
['D = [' num2str(d(1)) '; ' num2str(d(2)) '; ' num2str(d(3)) '; '
num2str(d(4)) ']']})
title(str)
xlabel('x position')
ylabel('y position')
hold off
legend('Location','southeast')

xdifference = q(1,:)-y(1,:);
xvdifference = q(2,:)-y(2,:);
ydifference = q(3,:)-y(3,:);
yvdifference = q(4,:)-y(4,:);

%%
figure
subplot(2,1,1)
plot(xdifference,'-r*','DisplayName','X-Values')
hold on
plot(ydifference,'-bs','DisplayName','Y-Values')
title('Difference Between Real and Measured Values for Position')
xlabel('Time')
```

```
xlim([0 60])
ylabel('Position (m)')
hold off
legend

subplot(2,1,2)
plot(xvdifference,'-m^','DisplayName','Xv-Values')
hold on
plot(yvdifference,'-gp','DisplayName','Yv-Values')
title('Difference Between Real and Measured Values for Velocity')
xlabel('Time')
xlim([0 60])
ylabel('Velocity (m/s)')
hold off
legend
```

d. How We Plotted Our Functions

To create Figure 6, first we chose the desired inputs. Our gen_state inputs are the same. For the Measured_state, we set our number of sensors to 4 (one for x position, y position, x velocity and y velocity). We set our C to the identity matrix and d is [25;10;25;10]. C is the identity matrix because we wish to use those exact standard deviations for the states from d. The d values were chosen because our directions ask for a 25 m standard deviation for position and a 10 m/s standard deviation for velocity. For the first figure of this function, we just plot the lines for the measured states and the actual states. There should be some variation since the noise is above 0. Our next plot is the difference between the measured values for the position and for the velocity. To do this, we merely subtract the corresponding row of the actual state matrix from the measured state matrix and we plot the difference. For the other graphs, we changed the C, d and sigma values to evaluate the new plots.

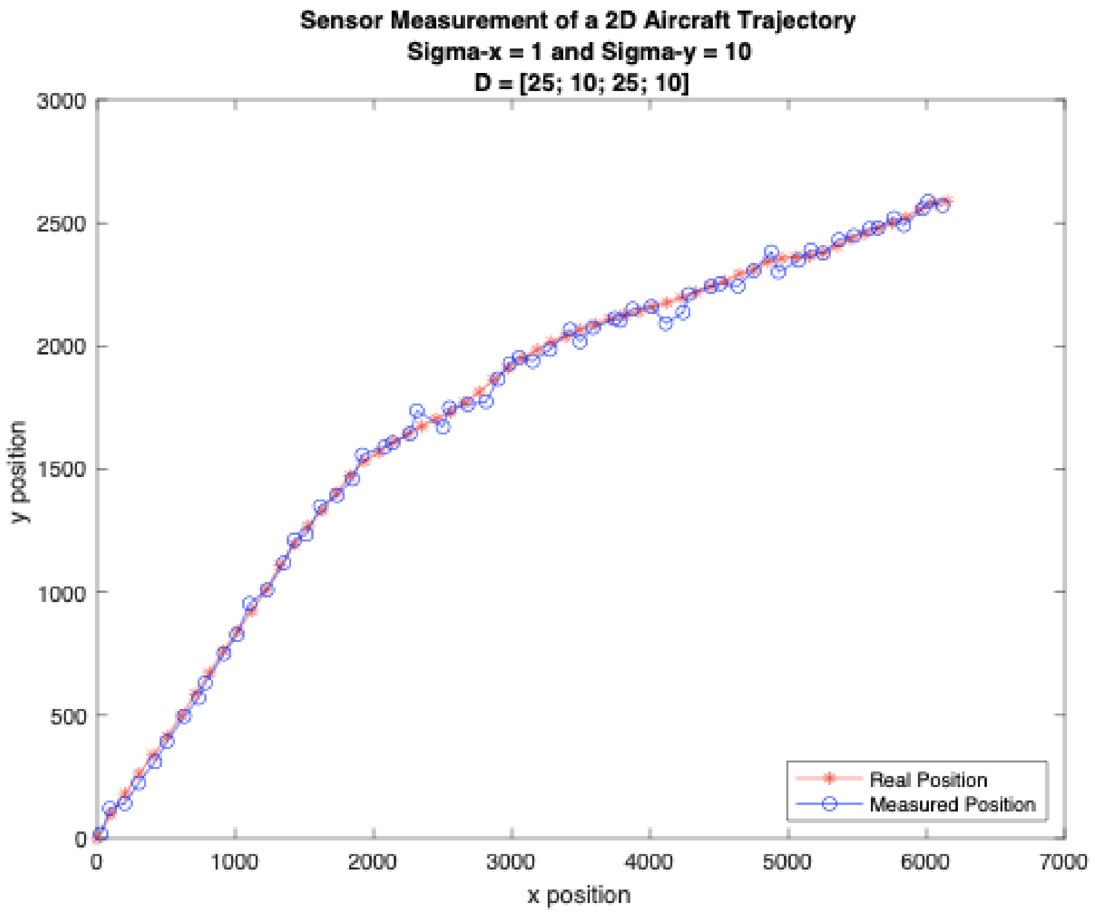

Figure 6: Graphical representation of the real position and the measured position using the measured state model we devised with sigma x =1 and sigma y =10 and the noise standard deviations was 25 for the x position, 10 for the x velocity, 25 for the y position and 10 for the y velocity

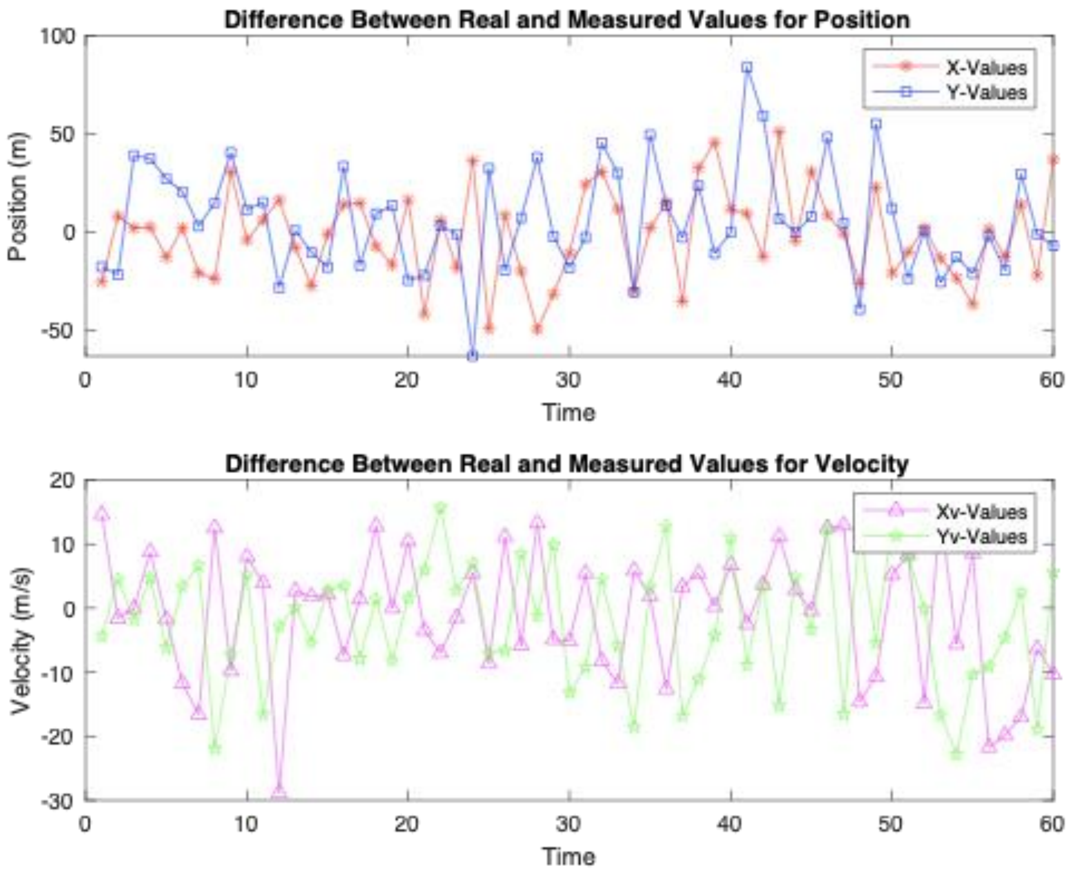

Figure 7: Graphical representation of the difference between the real and measured positions for Figure 6. The position is on the upper graph and the velocity is shown on the lower graph.

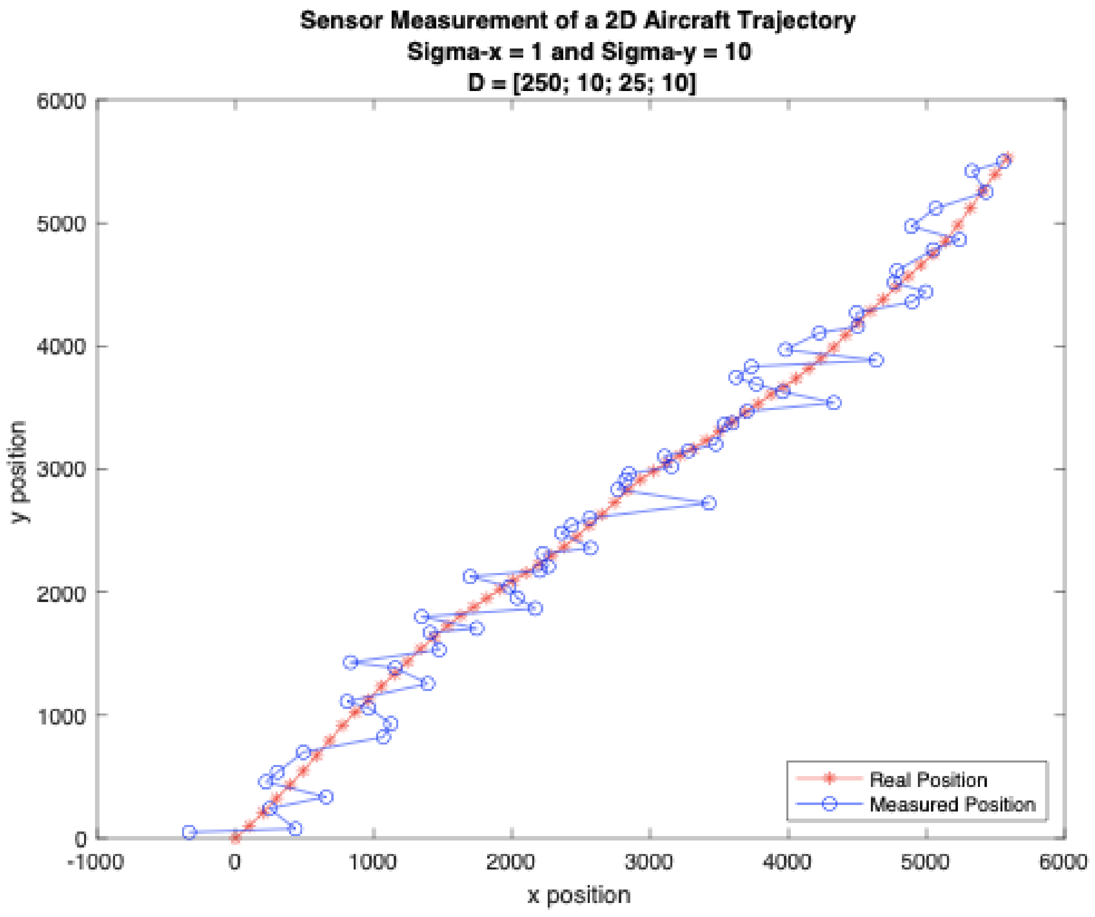

Figure 8: Graphical representation of the real position and the measured position using the measured state model we devised with sigma x =1 and sigma y =10, and the noise standard deviations was 250 for the x position, 10 for the x velocity, 25 for the y position and 10 for the y velocity

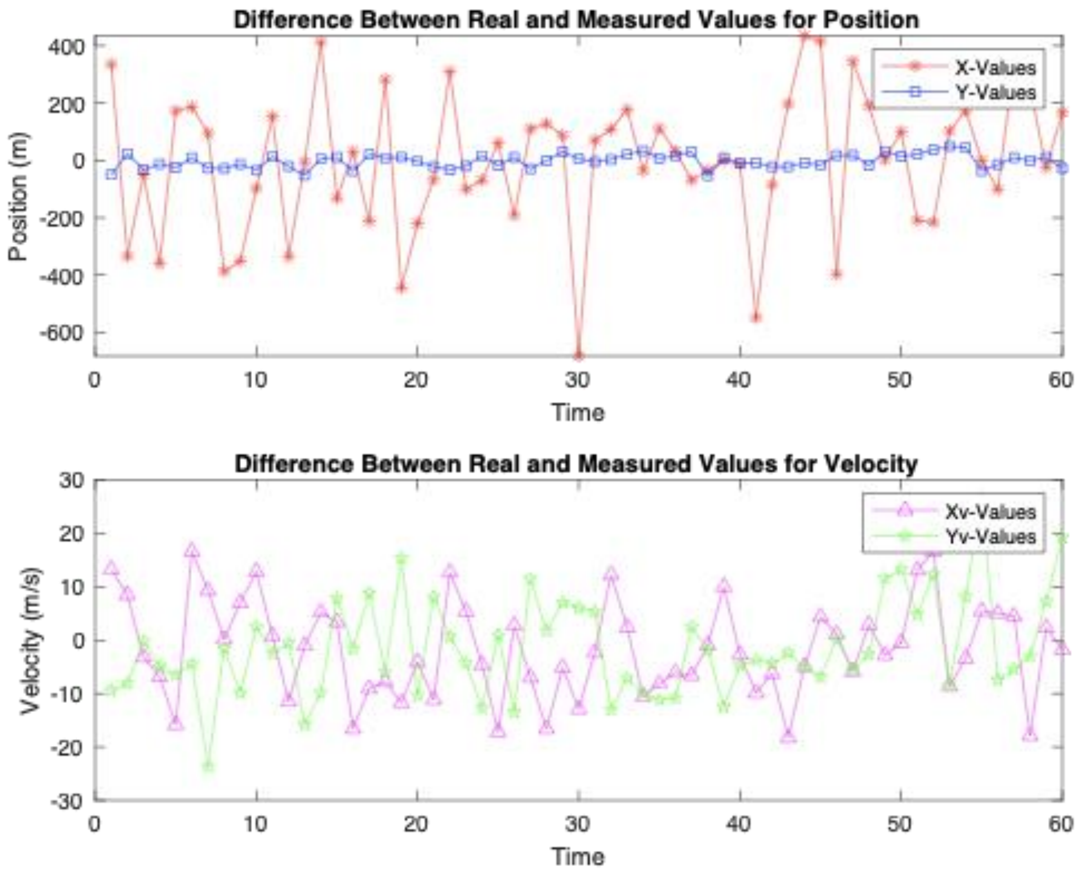

Figure 9: Graphical representation of the difference between the real and measured positions for Figure 8. The position is on the upper graph and the velocity is shown on the lower graph.

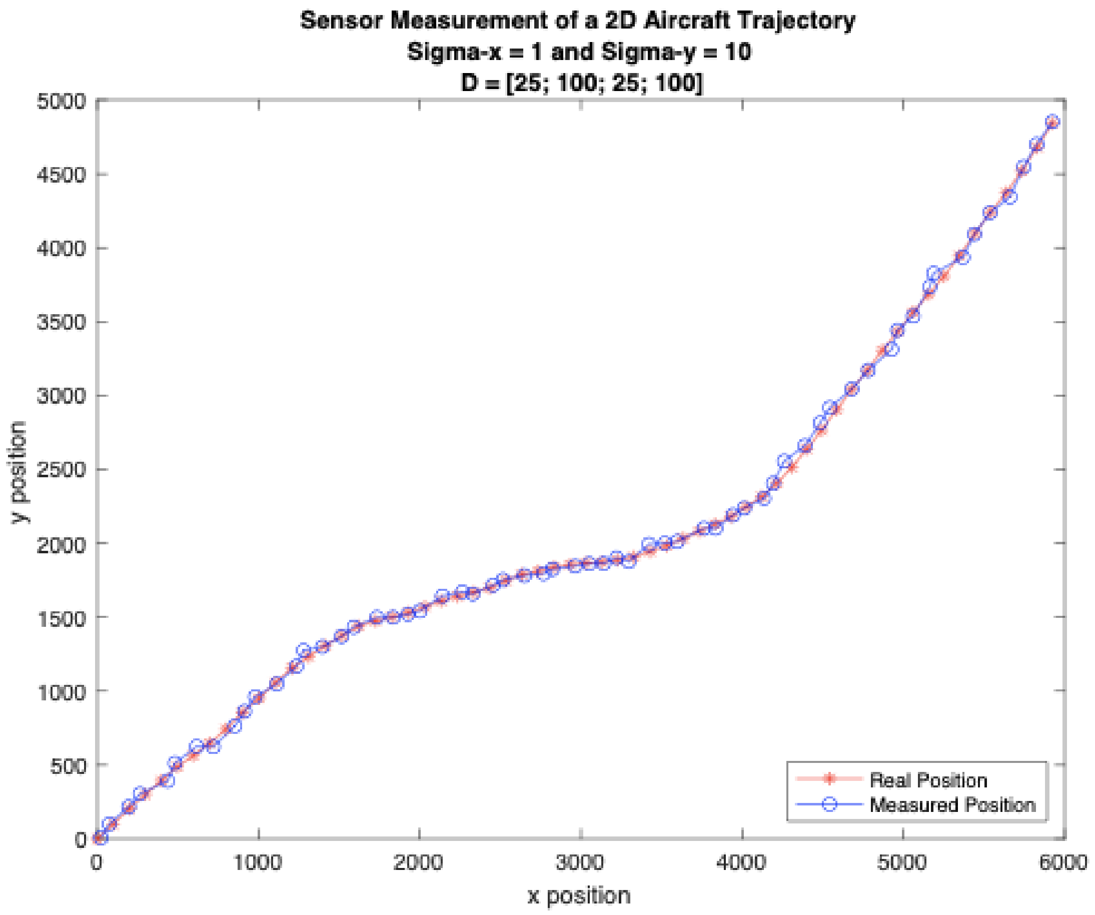

Figure 10: Graphical representation of the real position and the measured position using the measured state model we devised with sigma x =5 and sigma y =10 and the noise standard deviations was 25 for the x position, 100 for the x velocity, 25 for the y position and 100 for the y velocity

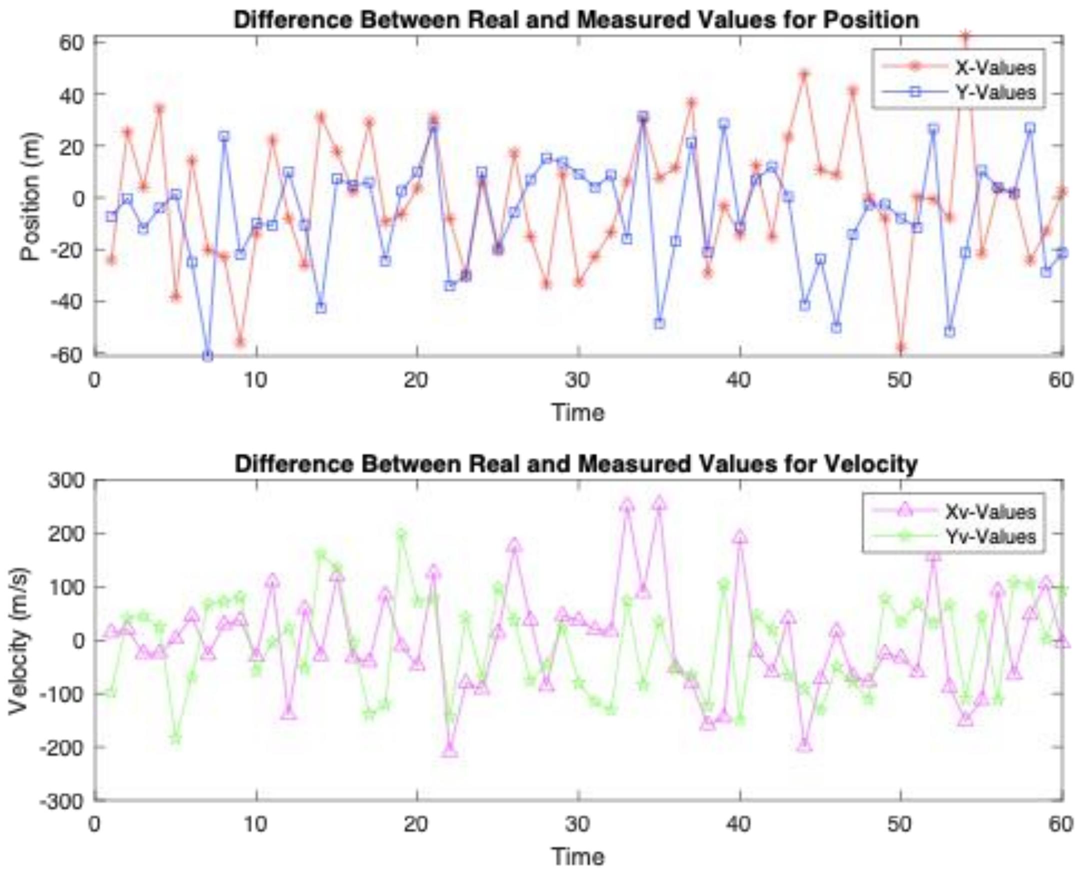

Figure 11: Graphical representation of the difference between the real and measured positions for Figure 10. The position is on the upper graph and the velocity is shown on the lower graph.

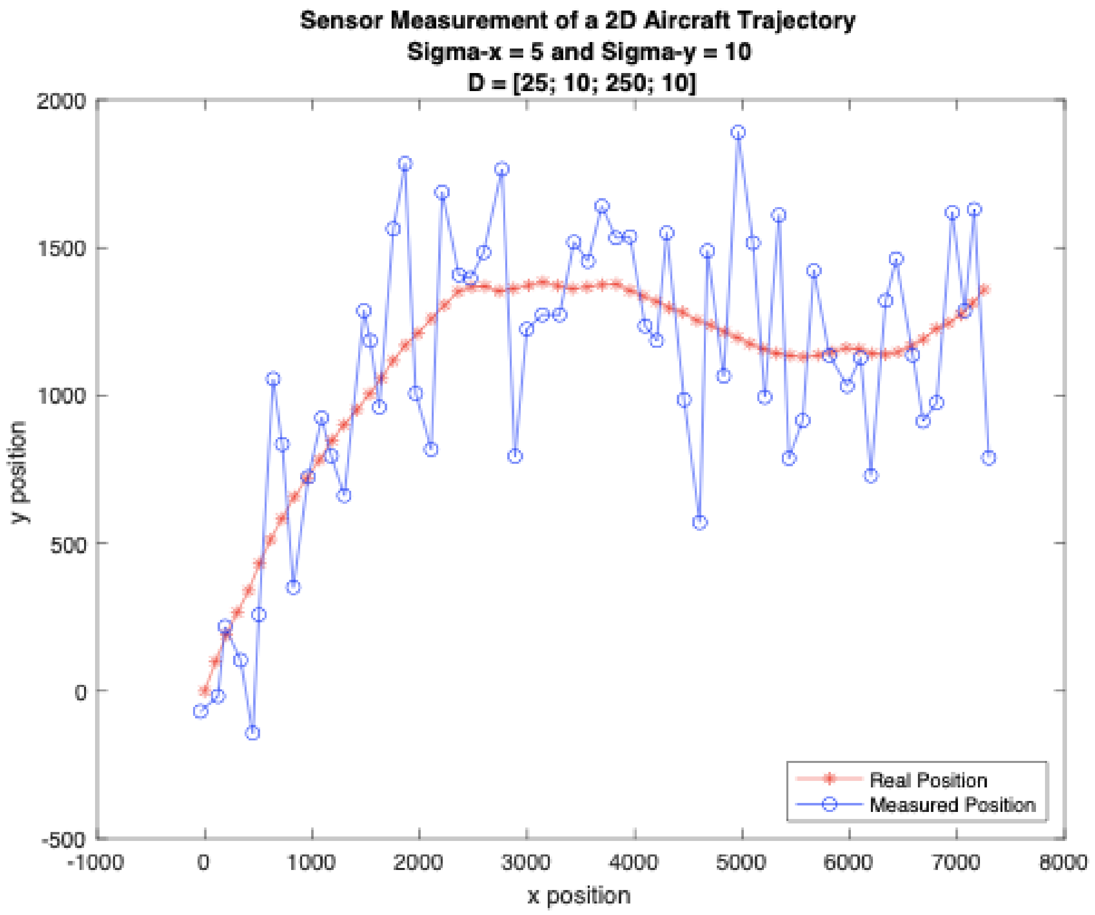

Figure 12: Graphical representation of the real position and the measured position using the measured state model we devised with sigma x =5 and sigma y =10, and the noise standard deviations was 25 for the x position, 10 for the x velocity, 250 for the y position and 10 for the y velocity

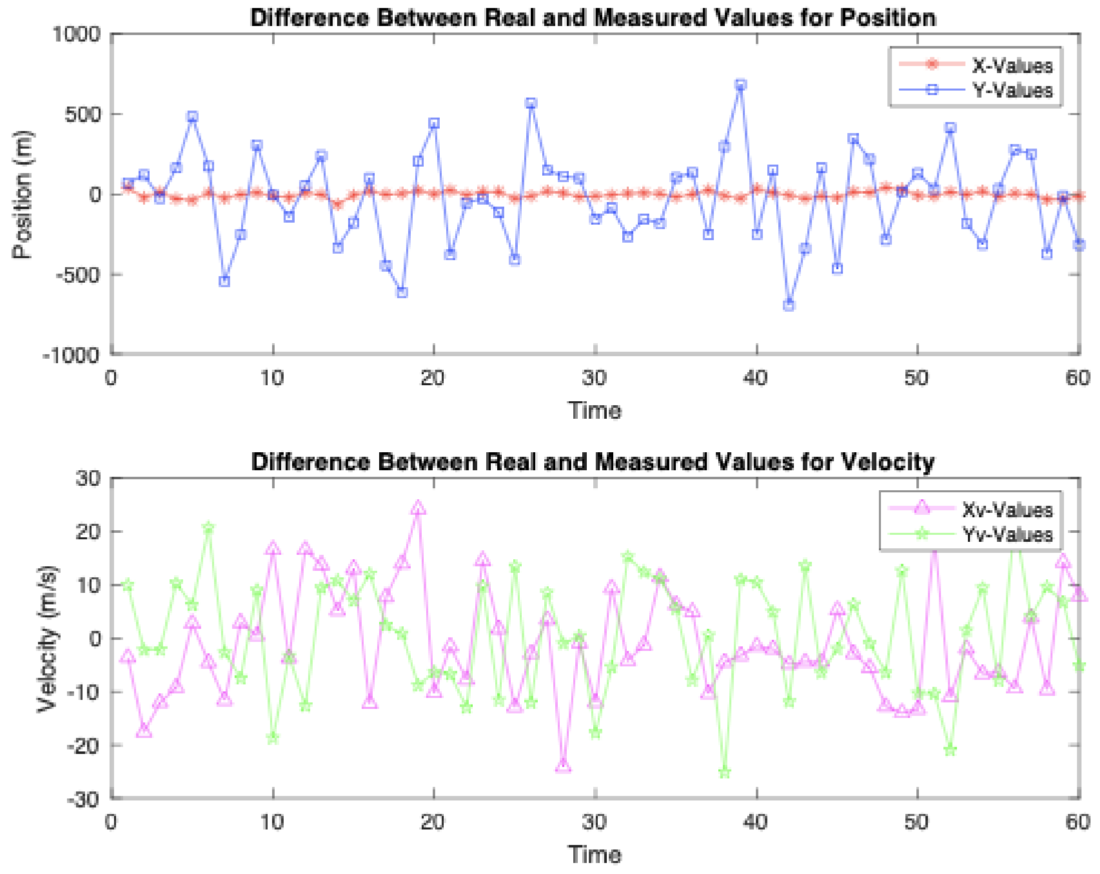

Figure 13: Graphical representation of the difference between the real and measured positions for Figure 12. The position is on the upper graph and the velocity is shown on the lower graph.

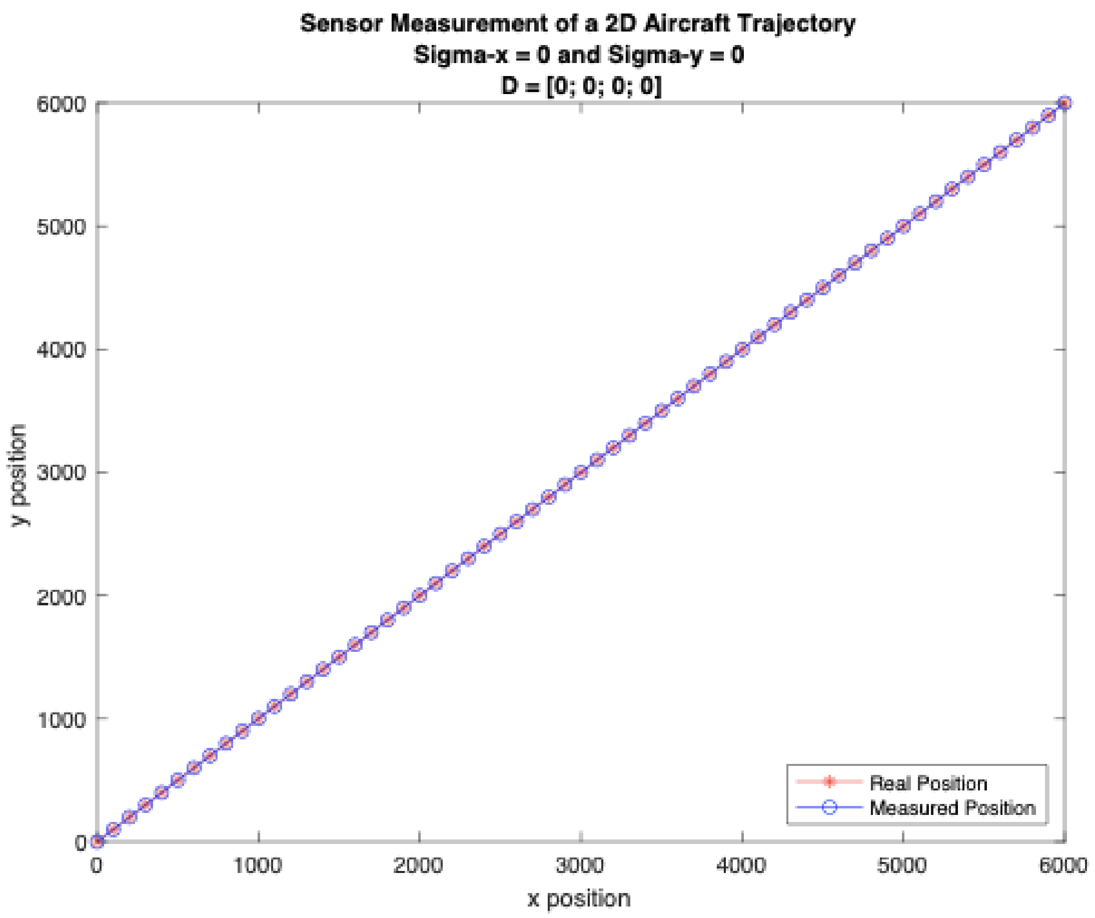

Figure 14: Graphical representation of the real position and the measured position using the measured state model we devised with sigma x =0 and sigma y =0 and no noise at all

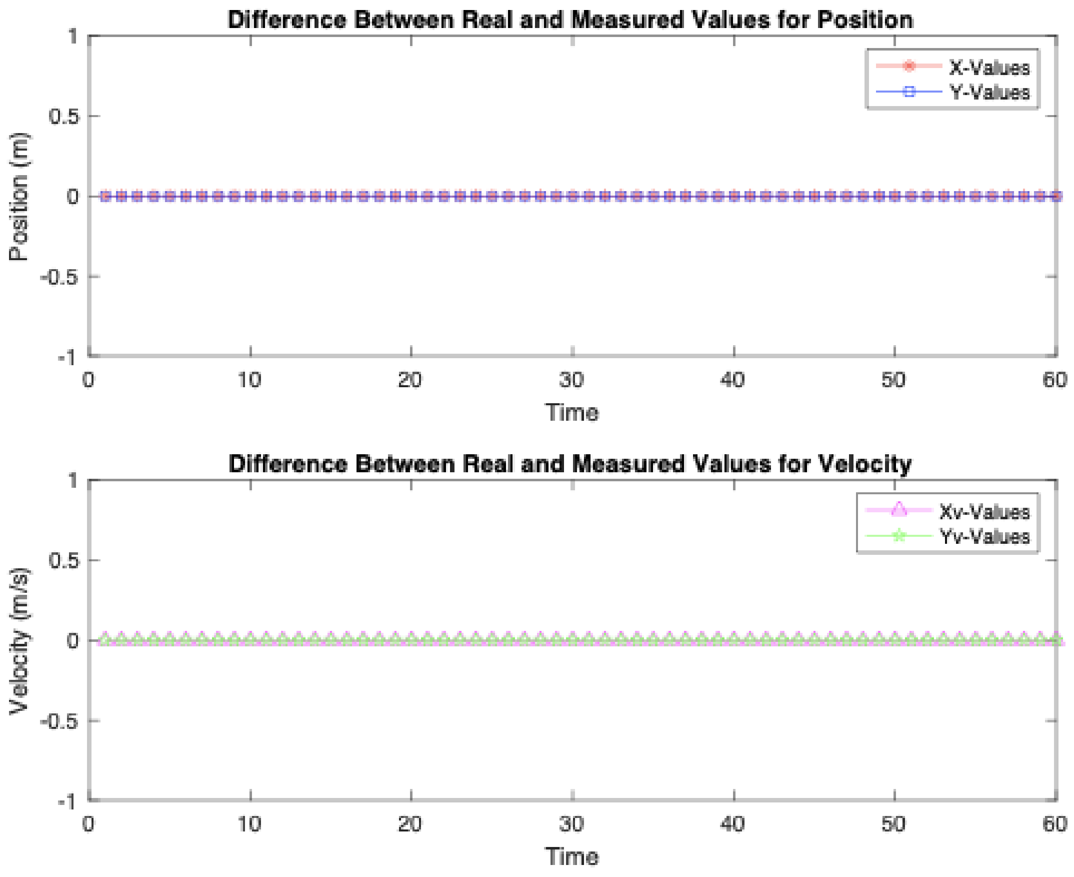

Figure 15: Graphical representation of the difference between the real and measured positions for Figure 14. The position is on the upper graph and the velocity is shown on the lower graph.

e. *Verification of the Function*

To verify our results, we utilized the graphic elements and our previous linear algebra knowledge. To explore if our noise vector was indeed working, we ensured that as we raised the C and d values, the difference between the actual state and the measured state was rising. We did this for hundreds of graphs and we could confirm that our function was doing as we expected. We did attempt a few calculations, but for this system only when the noise was low could we really compare our graphs. As we raised the noise, the graphs became too erratic to compare.

III. Exploiting the Dynamic Model to Improve Sensor Measurements

a. *Purpose*

Our task is to take the inaccuracies from the measured states and filter them to provide an improved estimate for the actual value. We will utilize the Recursive Least Squares Method to find the estimated states. To find this, we will first do these steps:

1. **Kalman Gain:** $K(n) = P(n-1)\mathrm{AT}(AP(n-1)\mathrm{AT} + R\ )\text{-}1$
2. **Covariance matrix:** $P(n) = R\ \text{-}1\ - K(n)AR\text{-}1\ P(n-1)$

3. **Estimated states: $\hat{q}(n) = \hat{q}(n-1) + K(n)(y(n) - A\hat{q}(n-1))$**

b. How Our Function Works

First, we set q hat to be a matrix with the minimum size of y rows and the maximum size of y plus 1 column. These dimensions were chosen because to properly do matrix multiplication, it is essential the dimensions match. Then, P according to the instructions is an identity matrix. We make K have a number of elements equal to the minimum size of y. We set the first values of qhat to be 0 for position and 100 m/s for velocity. Next inside of a for loop, we perform the Kalman Gain with the formula above. Then, we input the K values into the Covariance Matrix. Finally, we use the P calculated from the Covariance Matrix for the Estimated State Model. We do this a k number of times and we produce the Estimated States.

c. Code

RLS_estimation function:

```
function [q_hat] = RLS_estimation(A,y,R)

    % A is a matrix
    % R is a matrix which has forgetting factors
    % y is measurements from the mesure_state
    % Y is p * (k + 1)

    %Returns:
    % q_hat is output as the columns of a matrix

    k = max(size(y)); %Set k to be the number of columns
    q_hat = zeros(min(size(y)),k+1); %Make matrix for values of q_hat
    P(:,:,1) = eye(max(size(A))); %initialize the first covariance
matrix
    K = zeros(min(size(y))); %Make matrix for values of K
    q_hat(:,1) = [0;100;0;100]; %Initialize first value of q_hat


    for n = 2:k+1
        K(:,:,n) = P(:,:,n-1)*A.'*(A*P(:,:,n-1)*A.'+R)^-1; %Kalman Gain
        P(:,:,n) = inv(R) - K(:,:,n)*A*(R^-1) *P(:,:,n-1); %Covariance
Matrix
        q_hat(:,n) = q_hat(:,n-1) + K(:,:,n)*(y(:,n-1)-A*q_hat(:,n-1));
%Estimated States
    end
end
```

Plotting the Estimated States:

```
%%
N = 60; %Amount of time/points
T = 1;
Px = 0; %Initial value of x
```

```
Py = 0; %Initial value of y
Vx = 100; %Initial velocity of x
Vy = 100; %Initial velocity of y
sigx = 5;
sigy = 5;

[q] = gen_state(N,T,Px,Py,Vx,Vy,sigx,sigy);

%%
p = 4;
N = 60; %Amount of time/points
C = eye(p); %Identity matrix
d = [25;10;25;10]; %Noise for the position and velocity of x and y

[y] = measure_state(p,N,C,d,q);

%%
A = [1 T 0 0;0 1 0 0;0 0 1 T;0 0 0 1]; %State Transition matrix
R = eye(p)*0.98; %Identity matrix with forgetting factor

[q_hat] = RLS_estimation(A,y,R);
q_hat(:,1)=[];

%%
hold on
plot(q(1,:),q(3,:),'-r*','DisplayName','Real Position')
plot(y(1,:), y(3,:), '-bo','DisplayName','Measured Position')
plot(q_hat(1,:),q_hat(3,:), '-c^','DisplayName','Estimated Position')
str = sprintf('2D Aircraft Trajectory with Sigma-x = %d and Sigma-y =
%d',sigx, sigy);
title(str)
xlabel('x position')
ylabel('y position')
hold off
legend('Location','southeast')
```

d. *How We Plotted Our Functions*

To create Figure 16, first we set the inputs. Our gen_state inputs are the same, except for a 5 m/s standard deviation (sigx and sigy =5). For the Measured_state, we used the same values as well. For the RLS function, we choose the constant velocity state transition matrix we used before for A. Then, we decided to use a forgetting factor of .98. This is because the forgetting factor, lambda, is supposed to be 0 to 1.0. The higher lambda is, the less sensitive the estimation will be to noise.1 Finally, we plotted the real positions, the measured positions and the estimated positions.

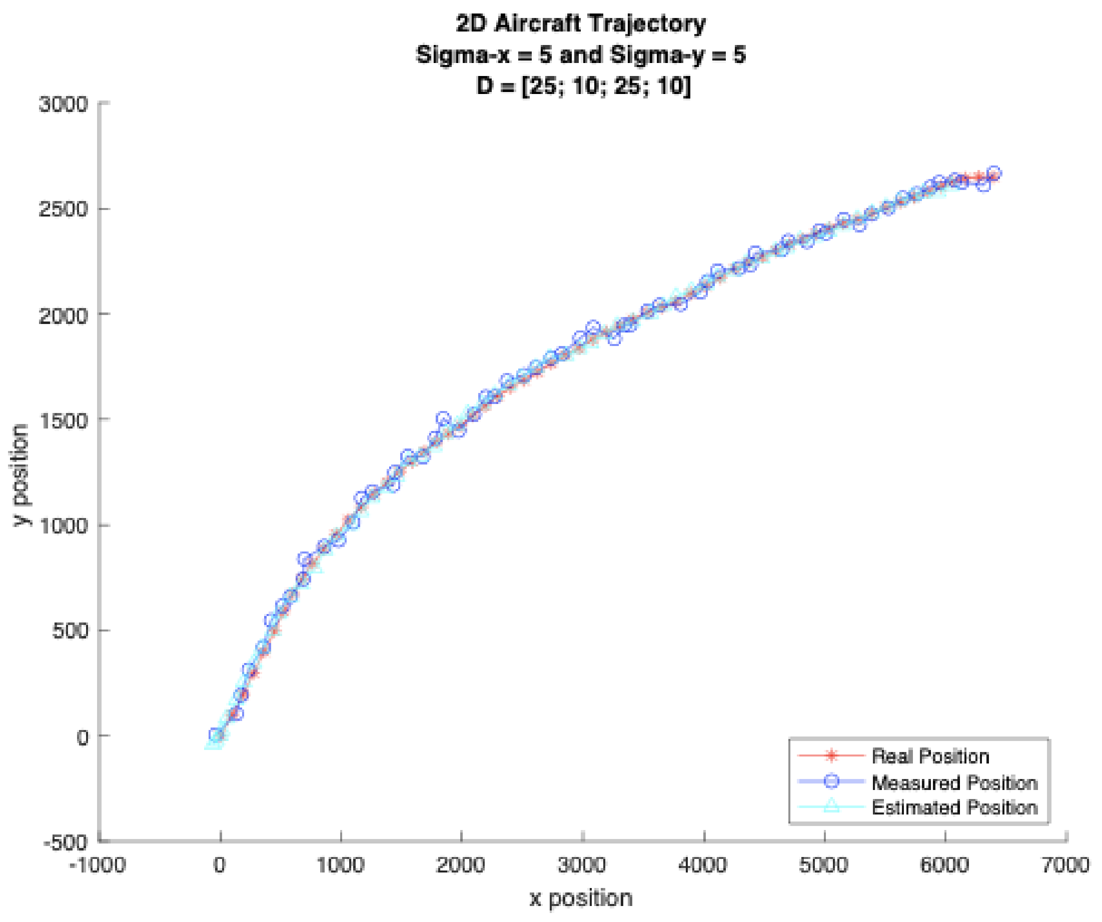

Figure 16: Graphical representation of the real position, the measured position and the estimated position using the estimated state model we devised with sigma x =5 and sigma y =5 and the noise standard deviations was 25 for the x position, 10 for the x velocity, 25 for the y position and 10 for the y velocity

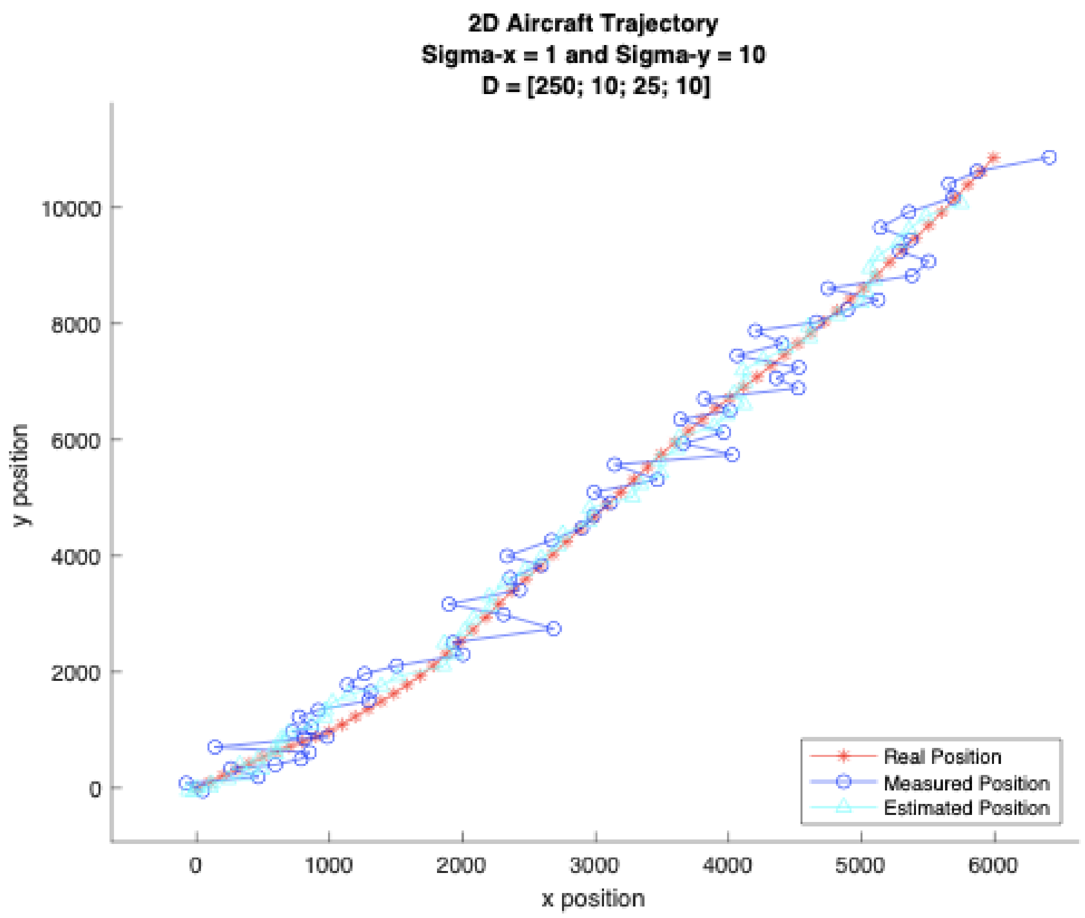

Figure 17: Graphical representation of the real position, the measured position and the estimated position using the estimated state model we devised with sigma x =1 and sigma y =10 and the noise standard deviations was 250 for the x position, 10 for the x velocity, 25 for the y position and 10 for the y velocity

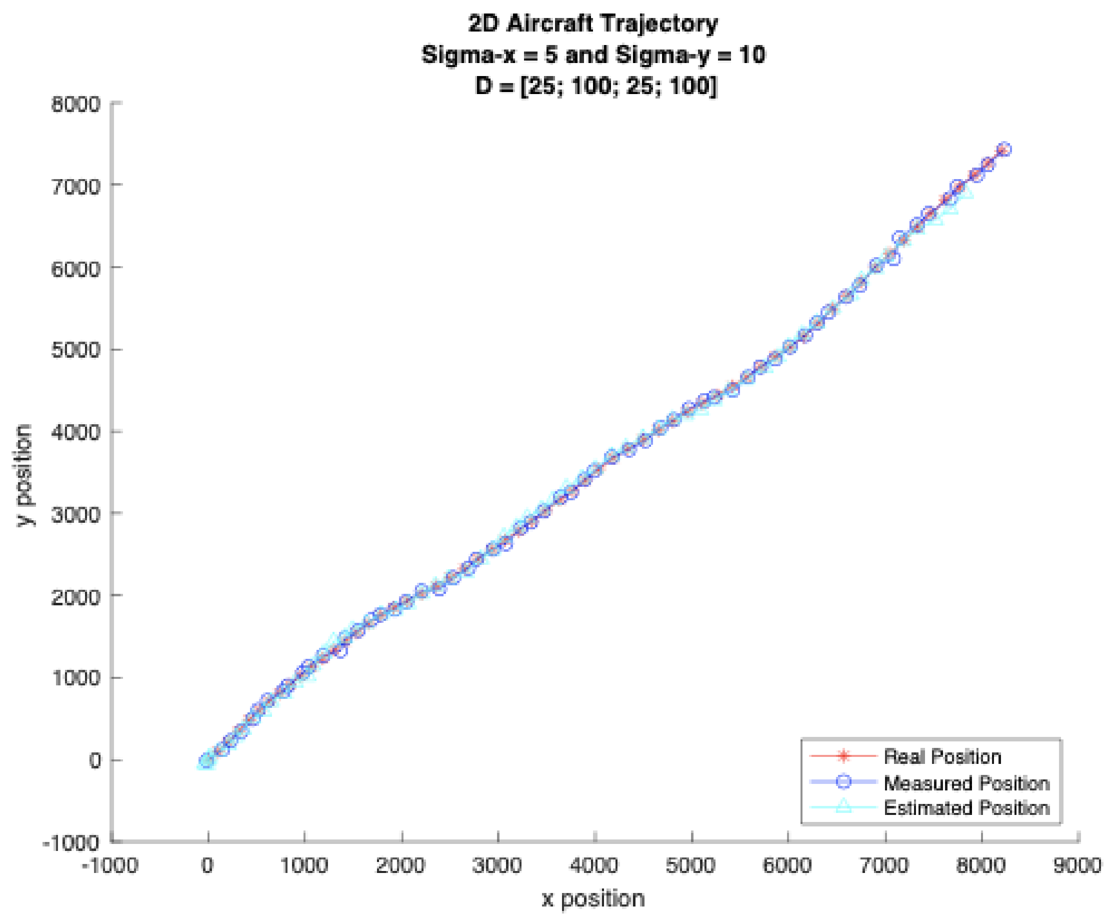

Figure 18: Graphical representation of the real position, the measured position and the estimated position using the estimated state model we devised with sigma x =5 and sigma y =10 and the noise standard deviations was 25 for the x position, 100 for the x velocity, 25 for the y position and 100 for the y velocity

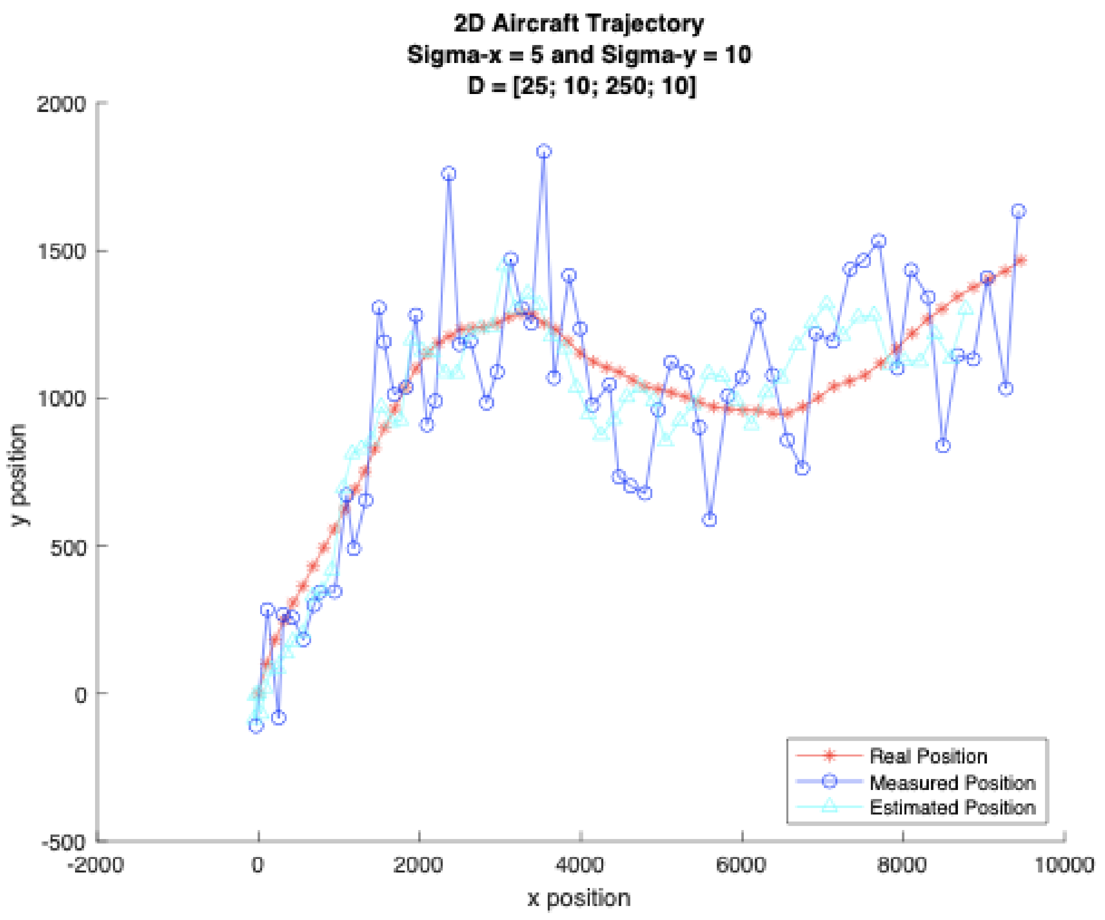

Figure 19: Graphical representation of the real position, the measured position and the estimated position using the estimated state model we devised with sigma x =5 and sigma y =10 and the noise standard deviations was 25 for the x position, 10 for the x velocity, 250 for the y position and 10 for the y velocity

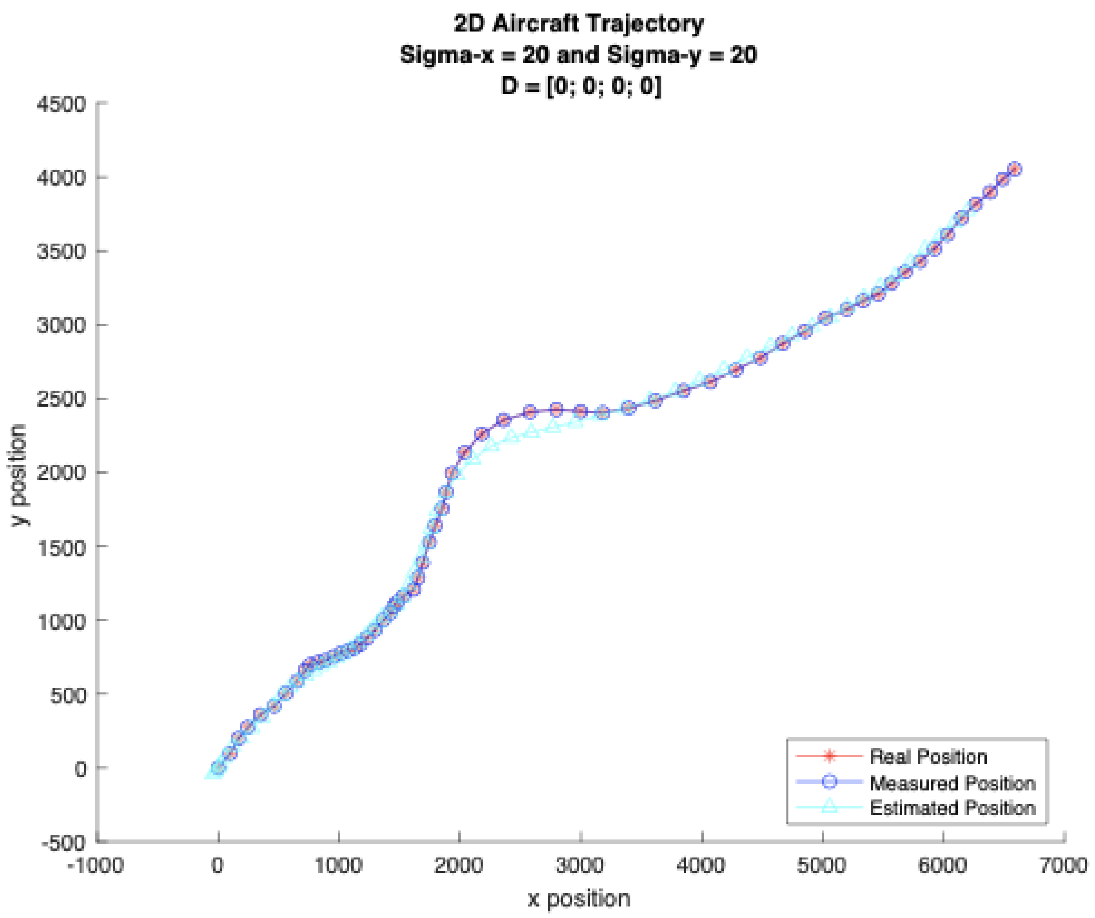

Figure 20: Graphical representation of the real position, the measured position and the estimated position using the estimated state model we devised with sigma x =20 and sigma y =20 and no noise at all

e. Verification of the Function

To verify our results, we compare our results over hundreds of graphs and they confirmed our results. We were able to calculate only a few estimation states, but they improved upon the measured states as well.

IV. Discussion

This project explored how dynamic systems can best be understood. First, we examined a simple difference equation with a discrete time system that we applied some randomness to with sigma x and y. In this case, the current state is directly affected by the previous state. Then, for part 2, we used the same difference equation, but we looked at how sensors will produce some noise and how the results changed as a result. The main difference besides the addition of sensors was the use of diagonalization. This was used to create a decoupled system. Finally, we explored a better method of obtaining data from sensors that reduced the amount of noise. The RLS method for estimation produced lines that were closer to the data set with less deviation than the direct sensor measurements. This is because the recursive least squares filter utilizes a projection of the measurements to construct a more straight line and it will reduce the overall noise. As we can see with increased noise from Figure 17, even as the measured values are far from the real position, the estimate is much closer. The RLS method reduces the spikes in the measured data and will serve to be a better method than using the data directly.

V. References

1. Emannual C. Ifeacor, Barrie W. Jervis. Digital signal processing: a practical approach, second edition. Indianapolis: Pearson Education Limited, 2002, p. 718