In this assignment we reconstructed the $\phi, \theta, u, v, w, z, p, q, r$ measurements of the UAV using a set of Kalman Filters. These measurements are important because they are the "flight" states of the UAV. ie, they are critical for the UAV to stay in stable flight. The X,Y position and yaw angle are not important for keeping the UAV in the air, so we don't care about reconstructing those three states. Once we have the 9 flight critical states, we can design a feedback controller to keep the drone airborne. For this assignment we will be be implementing a Wiener Filter on the UAV and checking to make sure the outputs of the filter make sense.

1 Assumptions

We reconstruct some of ours signals using the acceleration measurements from the IMU. For the angle signals, ϕ, θ, we used ratios of the acceleration measurements to determine the angle. This reconstruction assumes that there are no outside forces on the drone. For example, we reconstruct ϕ with

$$\phi = \frac{a_y}{a_z} \tag{1}$$

which at $\phi = 0$, we have $a_y = 0$. However, if there is an outside force on the UAV in the y direction, such as wind, the UAV will believe that it has rotated. This assumption comes from the fact that we are ignoring aerodynamics and other outside forces in our dynamics model. Additionally, our angle reconstructions are actually linearizations of trig functions,

$$\tan(\phi) = \frac{a_y}{a_z}. \tag{2}$$

but we assume the UAV will be close to trim, so small angle approximations hold.

Because we are using a Wiener Filter, we are assuming steady-state or near steady-state because the Wiener Filter has constant gain. For the actual UAV, we will switch to a time-varying Kalman Filter.

We will not be using a Kalman or Wiener Filter on the p, q, r measurements, because we cannot reconstruct them outside of the actual measurements from the sensors. Therefore, we must assume that they are accurate.

For process and measurement noise, w, v, we assume that the sources of noise are independent and white. We estimate the covariances of each source of noise, Q, R respectively, and assume the cross-covariance is 0 because they are independent. For single integrators, this gives us

$$Q = \sigma_w^2 \tag{3}$$
$$R = \sigma_v^2 \tag{4}$$

and for the double integrator,

$$Q = \begin{pmatrix} 0 & 0 \\ 0 & \sigma_w^2 \end{pmatrix} \tag{5}$$

$$R = \sigma_v^2 \tag{6}$$

where σ is the standard deviation of the noise.

2 Building the Filter

The first step is designing the Wiener Filter. This requires creating a differential system with sensor readings as inputs. I decided to make single integrator for ϕ, θ, u, v and a double integrator for z, w. This is because for the first 4 signals, we can get both the actual signal and a derivative from themselves and the acceleration measurements. u, v are measured and are also the integral of their respective accelerations, so we create the single integrator system,

$$\dot{u} = (0)u + (1)a_x \tag{7}$$

$$y = (1)u + (0)a_x \tag{8}$$

for the two angles, we can reconstruct the angles using accelerations, and use measured angular velocities as the inputs

$$\phi = \frac{a_y}{a_z} \tag{9}$$

$$\theta = \frac{a_x}{\sqrt{a_y^2 + a_z^2}} \tag{10}$$

and create the same single integrator system in equations 3,4 for the filter. For z, w we can use the double integrator given in an example in the class notes,

$$\begin{pmatrix} z \\ w \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} z \\ w \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} a_z \tag{11}$$

$$y = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} z \\ w \end{pmatrix} + (0)a_z \tag{12}$$

Next, we have to account for process noise and measurement noise. For this, I used a Simulink model of the dynamics with white noise and recorded the data. Then, I took the covariances of the recorded signals to get Q, R. These values are not incredibly important, because we can adjust them later to get the desired behavior from the filters.

Next, we want do discretize the dynamics of our single and double integrators, because the filters are discrete. In general, the discrete dynamics of a system are

$$A_\Delta = e^{A\Delta T} \tag{13}$$

$$B_\Delta = \int_0^{\Delta T} e^{At}\, dt B \tag{14}$$

where ΔT is the sampling time of the UAV, 1/200 seconds. We can quickly calculate these using the fact that integrators are in Jordan Canonical Form,

$$A_{\Delta single} = (1) \tag{15}$$
$$B_{\Delta single} = (\Delta T) \tag{16}$$
$$A_{\Delta double} = \begin{pmatrix} 1 & \Delta T \\ 0 & 1 \end{pmatrix} \tag{17}$$
$$B_{\Delta double} = \begin{pmatrix} \Delta T^2 \\ \Delta T \end{pmatrix} \tag{18}$$

Up to this point, the process for the Wiener and Kalman Filters are the same. To create the Kalman Filter, we simply plug in the values for our discrete dynamics matrices, our output matrices and noise covariances into a Kalman Filter Simulink block. For the Wiener Filter, we must calculate the static gain of the observer, L. We use Matlab's 'lqe' function to do this. We then build the dynamic difference system of the observer

$$\mathcal{S}X_e = A_\Delta X_e + B_\Delta u + L(y - y_e) \tag{19}$$
$$y_e = CX_e + Du \tag{20}$$

in Simulink with a discrete integrator. The code to calculate the matrices for the single and double integrator Wiener and Kalman Filters is shown below, and the Simulink models for Kalman and Wiener Filters are shown in Figure 1.

Listing 1: Calculations for Wiener Filters

```
load('In_out_data')
syms t
Ts = 0.005;

% Single Integrator state-space model
A_single = [0];
B_single = [1];
C_single = [1];
D_single = [0];

% Double Integrator state-space model
A_double = [0 1; 0 0];
B_double = [0; 1];
C_double = [1 0];
D_double = [0];

% Calculating Discrete Dynamics
Ad_single = expm(A_single*t);
Bd_single = int(Ad_single)*B_single;
Ad_double = expm(A_double*t);
Bd_double = int(Ad_double)*B_double;

% Evalute Discrete Dynamics for t = Ts (sampling time)
A_s_single = eval(subs(Ad_single,Ts));
B_s_single = eval(subs(Bd_single,Ts));
A_s_double = eval(subs(Ad_double,Ts));
B_s_double = eval(subs(Bd_double,Ts));

```

```
% Calculate Covariances
q_single = cov(out.In_Out_Data{1}.Values.Data); % input data
r_single = cov(out.In_Out_Data{2}.Values.Data); % output data

% don't worry about size of Q, the lqe function handle it
q_double = q_single;
r_double = r_single;

% Wiener Filter code
G_single = B_single; % noise input is same as process input
G_double = B_double;

% Gain for the Wiener filter
L_single = lqe(A_single, G_single, C_single, q_single, r_single, 0);
L_double = lqe(A_double, G_double, C_double, q_double, r_double, 0);
```

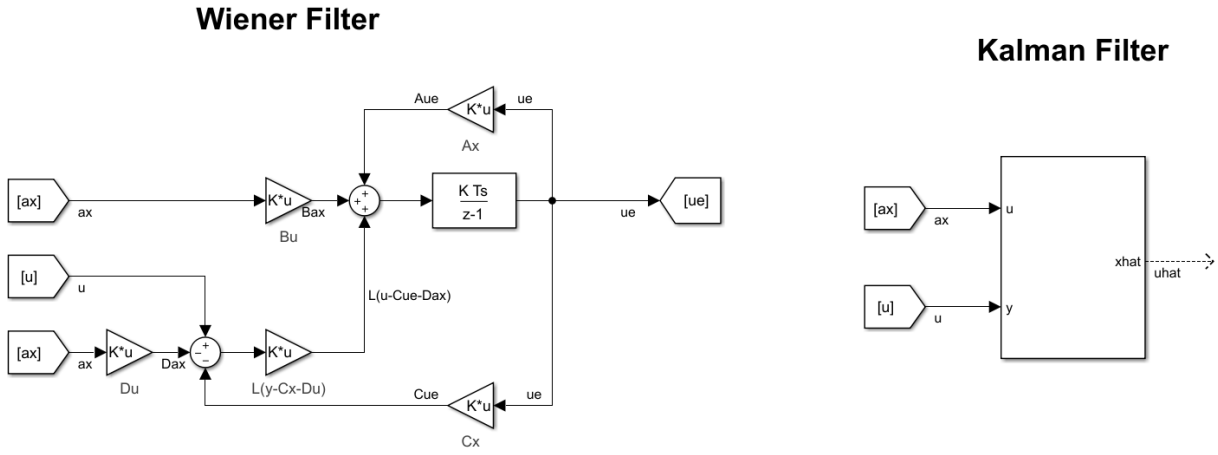

Figure 1: Kalman and Wiener Filters for u generalized by changing the inputs and states

We can adjust the values of Q, R for both the LQE observer and Kalman Filter block to get the best, least noisy, state estimates.

3 Analysis

To analize the effectiveness of the filter, I first simultated it purely in Simulink using the example from lecture as a base. I only needed to do the single integrator for this part, because the double integrator was given. The single integrator is shown in Figure 2, with it's output shown in Figure 3. I used a step input, which should produce a linear output. The output of the double integrator estimator, shown in Figure 4, should be quadratic for z and linear for w. Both filters match the actual values accurately.

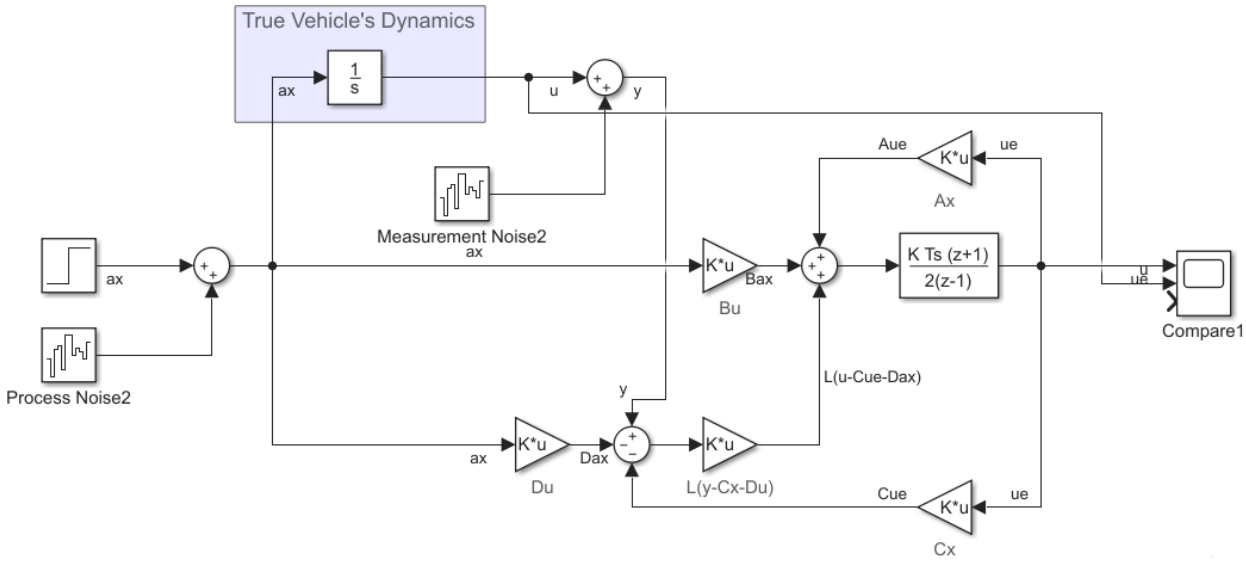

Figure 2: Simulated Single Integrator Wiener Filter

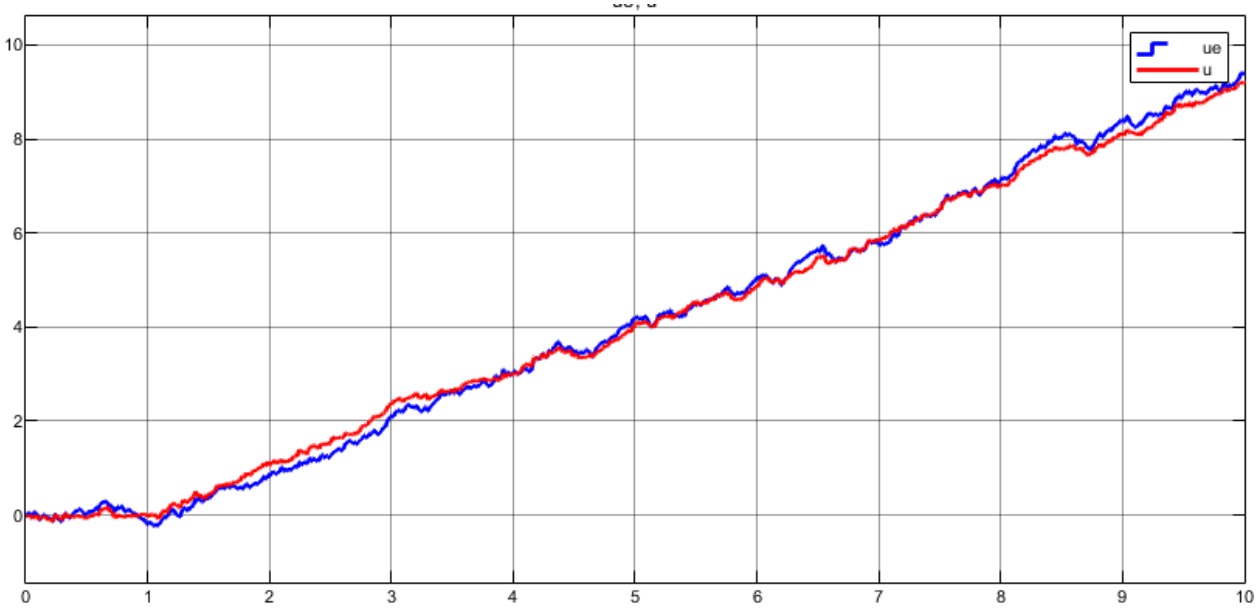

Figure 3: Simulated Single Integrator Wiener Filter Output

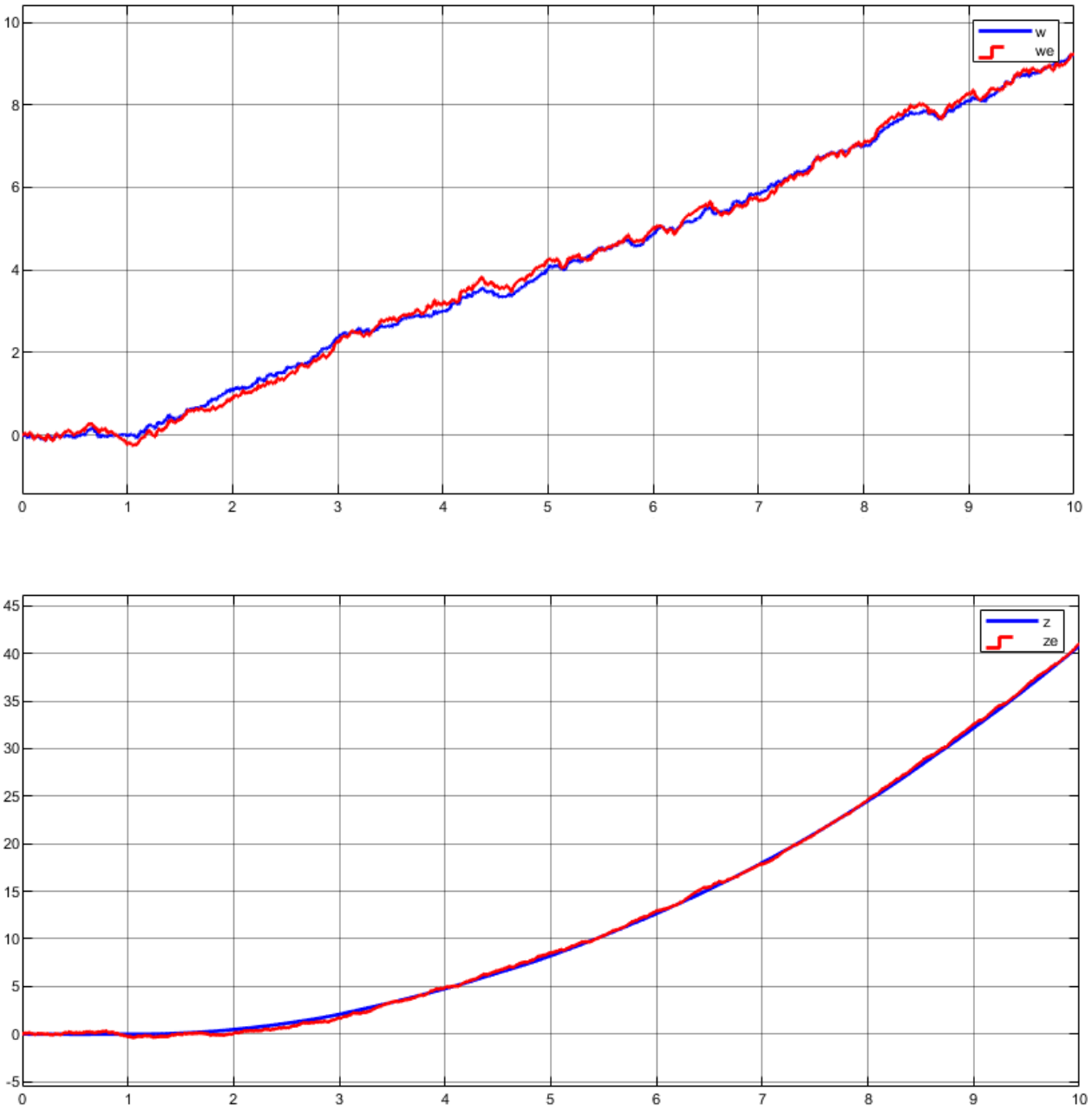

Figure 4: Simulated Double Integrator Wiener Filter Output

Next, we want to evaluate the performance of the Kalman Filter blocks. We do this the same way as for the Wiener filter with a step input. Initially, the output was very noisy, as shown in Figure 5.

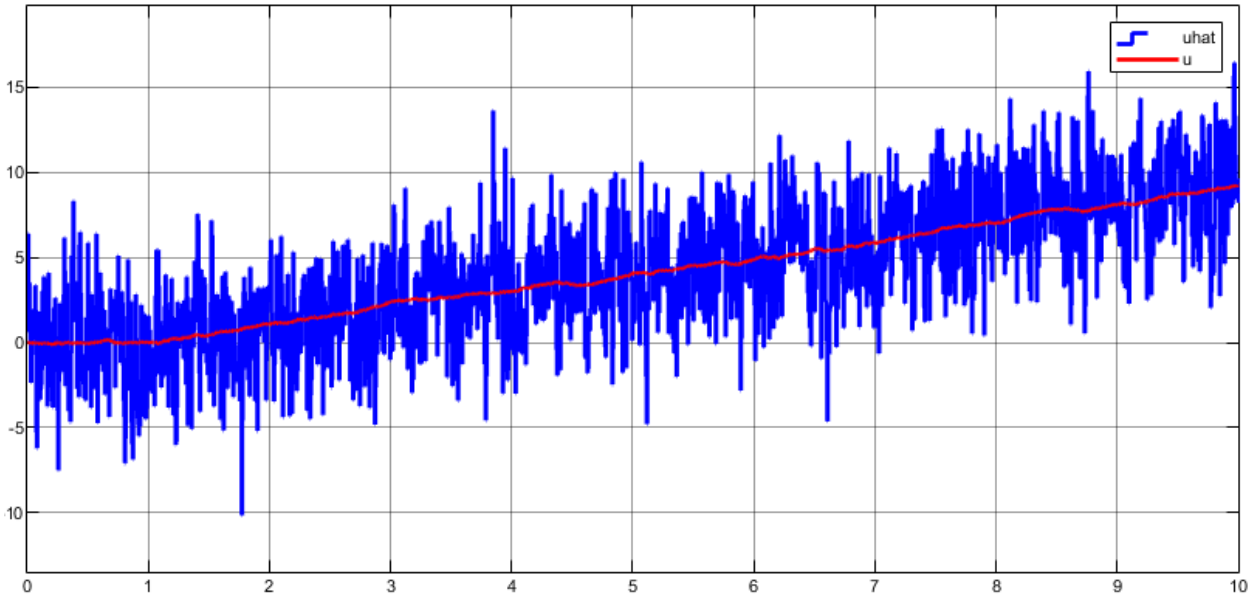

Figure 5: Simulated Single Integrator Kalman Filter Output, Small R

We then change the size of R, multiplying it by 10000 to simulate that we are much more confident in our integrator than the measurement. This reduced most of the noise, as shown in Figure 6. We did this as well for the double integrator.

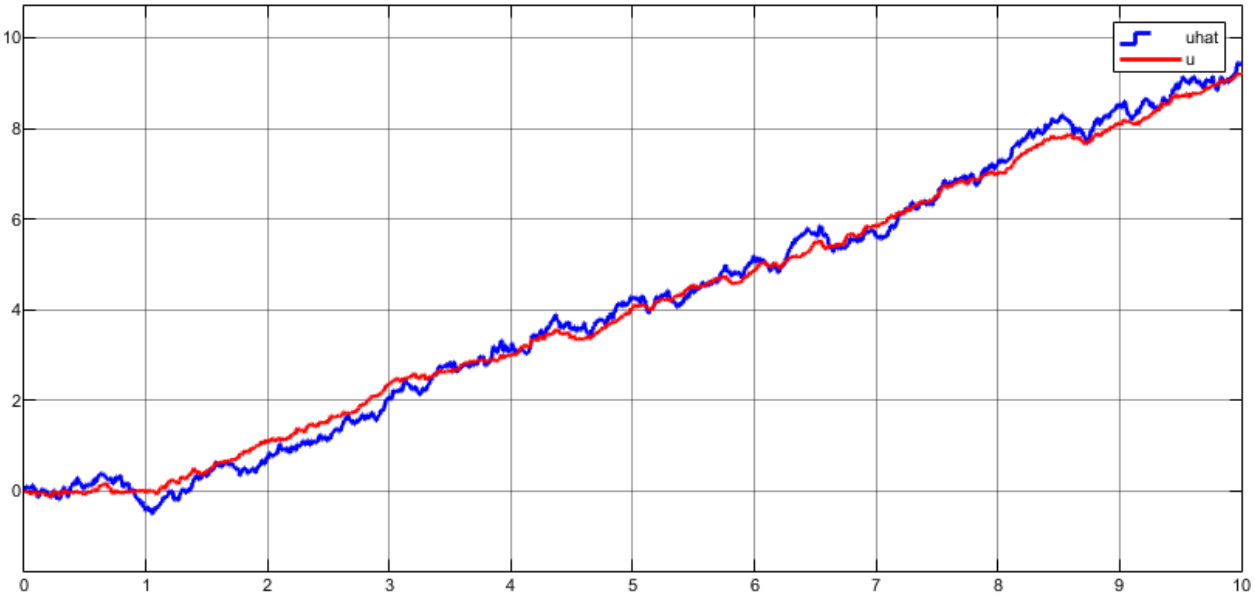

Figure 6: Simulated Single Integrator Kalman Filter Output, Big R

Next, I implemented the Wiener Filter onto the UAV. The Filter tracked the states fairly well, despite some noise in the measurements. An example is shown in Figure 7.

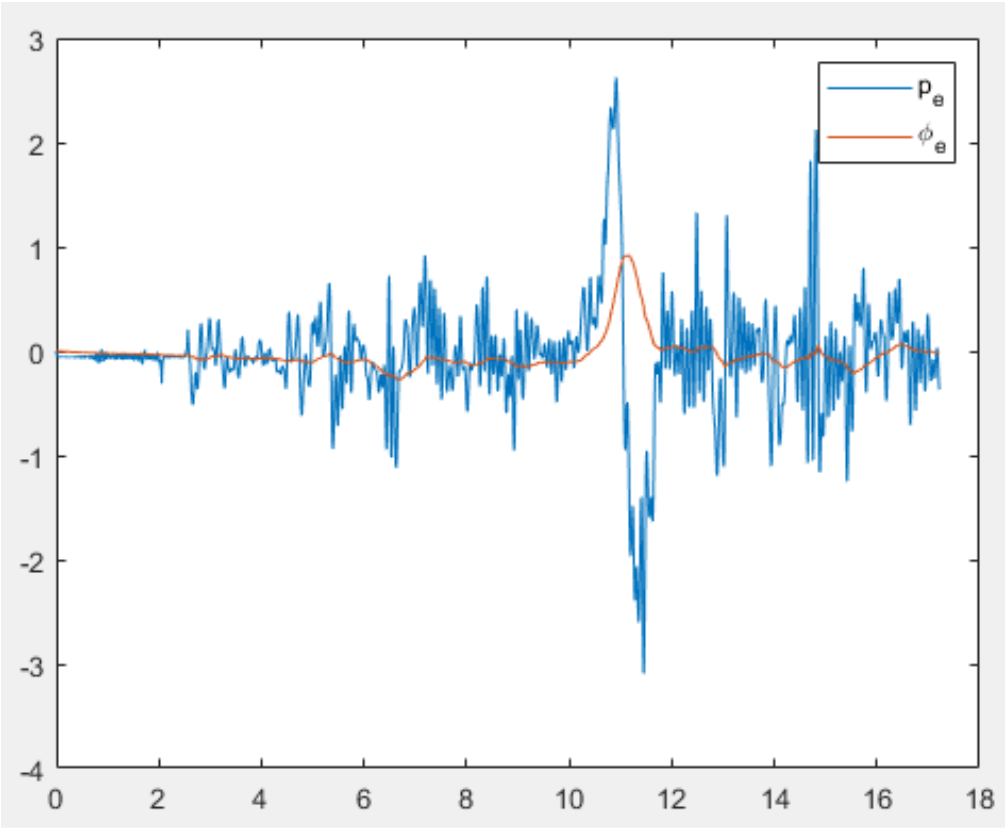

Figure 7: Actual Single Integrator Wiener

4 Conclusion

Though the Wiener Filter is not the best Filter for the dynamic UAV, we were able to successfully implement it on the UAV and confirm that it worked fairly well. Not all states were as clean as ϕ_e in Figure 7, but they all matched the behavior I input. The Kalman Filter will likely do a better job for reconstructing states for closed-loop feedback control once it has been calibrated.