ESE 448

Chris Bonney

3 - Linear and Nonlinear Models

February 13, 2021

This assignment has a few goals: first, is to create the beginning of a practical model of the system we can use to test our control laws before uploading them onto the drone; second, we want to test the viability of a linearized model compared to the nonlinear one; third, we want to observe the dynamics of the system; and last, we test the stability of the system.

1 Assumptions and constants

Many of our assumptions from the last assignment where we derived the equations of motion still hold for this assignment, including

1. Aerodynamic forces are negligible at the low air speed of the UAV. Since we are testing the UAV inside, wind speed is also assumed to be 0.
2. The UAV is symmetric about its axes for the purpose of calculating the mass moment of inertia matrix, J. The values of the moment of inertia about each axis, J_{xx}, J_{yy}, J_{zz}, were given in the assignment description.
3. The coefficient of thrust curve, $C_T(n)$, is linear relative to the propeller's rotational velocity, and the coefficient of power curve, $C_P(n)$, is constant relative to the propeller's speed. The values we used were found using a linear fit from propeller data collected within the range of 180 and 480 Hz propeller speed [1].
4. There are a number of values we assumed to be constants, or as calculated in the previous assignment in Matlab. The code for this, shown below, is abbreviated for the sake of reading, and the full code will be attached.

Listing 1: Constants and Assumptions

```
%UAV
UAV_mass = 0.068; %kg ; total mass of UAV
Jxx = 0.0000582857;
Jyy = 0.0000716914;
Jzz = 0.0001;
J = [... % kg/m2 ; Moment of Inertia Matrix of UAV
     Jxx   0   0 ;...
      0   Jyy  0 ;...
      0    0  Jzz;...
];

%PHYSICAL CONSTANTS
rho = 1.22495; % kg/m3 ; density of air, given in presentation
F_g = [0;0;UAV_mass*9.81]; %m/s2 ; acceleration due to gravity in NED frame

%PROPELLER
prop_dist = sqrt(2)*0.09; %meters ; distance from CM to prop
prop_mass = 0.001; %kg (1 gram) ; mass of prop
prop_dia = 0.066; %meters ; diameter of prop
prop_mass_moment = 1/12*(prop_dia^2)*prop_mass; %kg/m2 ; Moment of Inertia of prop about its CM
Jprop = prop_mass_moment + prop_mass*(prop_dist^2); %kg/m2 ; MoI of prop in UAV frame, from Parallel Axis
C_T = [0.069075;4.95e-05]; %C_T(n) = C_T(1) + C_T(2)*n for prop spinning at n Hz
C_P = 0.041;
thrust_eq = @(n) [(C_T(1)+C_T(2)*n).*(n.^2)*rho*(prop_dia^4)]; %Thrust in N, a function of motor input, Hz
```

```
power_eq  = @(n) [(C_P)*(n.^3)*rho*(prop_dia^5)]; %Power in W, a function of motor input, Hz

%TRIM STATE
trim_state = [0,0,0,0,0,0,0,0,0,0,0,0]; %All states at 0
trim_input = [293,293,293,293]; %Hz for all propellers
```

5. Like in the previous assignment, we will have two frames, the inertial "North, East, Down" frame (NED), and the body frame. The states of the system will be defined in the following order:

$$\textbf{states} = \begin{bmatrix} X_{NED} & Y_{NED} & Z_{NED} & u & v & w & \phi & \theta & \psi & p & q & r \end{bmatrix} \tag{1}$$

which will be written as 3-dimentional vectors with x, y, z components

$$\textbf{states} = \begin{bmatrix} NED & V & euler & \omega \end{bmatrix} \tag{2}$$

where NED is the NED frame positions, V is the body axis velocities, $euler$ is the euler angles required to produce the rotation using the rotation matrix $R = R(\phi)R(\theta)R(\psi)$. We will be using the same equations of motion as in assignment 2:

$$m\dot{V} + \omega \times mV = R(\phi, \theta, \psi)\vec{F}_G + \vec{F}_T \tag{3}$$

$$J\dot{\omega} + \omega \times J\omega = \vec{M}_T + \vec{M}_{gyro} \tag{4}$$

$$\dot{\phi} = p + \tan\theta(q\sin\phi + r\cos\phi) \tag{5}$$

$$\dot{\theta} = q\cos\phi - r\sin\phi \tag{6}$$

$$\dot{\psi} = \frac{q\sin\phi + r\cos\phi}{\cos\theta} \tag{7}$$

$$N\dot{E}D = R(\phi, \theta, \psi)^T V \tag{8}$$

with the addition of a set of outputs to the system:

$$h = -Z_{NED} \tag{9}$$

$$a_{x,y,z} = R(\phi, \theta, \psi)^T \frac{\vec{F_T}}{m} \tag{10}$$

with h as the height of the UAV and $a_{x,y,z}$ as the NED-frame accelerations of the UAV from thrust.

2 Derivations and Calculations

In this section, we will calculate the gyroscopic moment on the UAV, as well as the Jacobian matrices evaluated at trim state and inputs for the linear system, $A,\ B,\ C,\ D$.

2.1 Gyroscopic Moment

The gyroscopic moment, M_{gyro}, was not calculated in the previous assignment, so we will show it here.

The change in angular momentum of the UAV is desribed as

$$\frac{\delta}{\delta t} H + \omega \times H = M_T \tag{11}$$

where $\frac{\delta}{\delta t}$ is the fixed-frame derivative, and $\omega \times H$ is due to the Coriolis Effect. Since we are ignoring aerodynamics, the only "external" moment is due to thrust from the motors. If we naively evaluate the above derivative and replace H with $J\omega$, we get equation 4, without the gyroscopic moments:

$$J\dot{\omega} + \omega \times J\omega = \vec{M}_T \tag{12}$$

this evaluation, however, assumes the propellers do not generate moments or have significant impact on the UAV's angular momentum. We therefore must split H into two parts, H_{body} and H_{prop}:

$$H = J\omega + J_p \left(\sum_{i=1}^{4} \begin{bmatrix} 0 \\ 0 \\ 2\pi n_i \end{bmatrix} \right) \tag{13}$$

where we are assuming J_p is the same for all four propellers. Additionally, since we are treating the propellers as a rod, we assume symmetry, making the diagonals of the matrix J_p 0; and, since the propellers are only spinning in the z body-axis, we only care about the J_{zz} component of J_p, which has been calculated in Listing 1. Because of this, we treat J_p as a scalar that multiplies the angular velocity of each propeller.

We can then plug this back into equation 11 to get

$$J\dot{\omega} + J_p \left(\sum_{i=1}^{4} \begin{bmatrix} 0 \\ 0 \\ 2\pi \dot{n}_i \end{bmatrix} \right) + \omega \times J\omega + \omega \times J_p \left(\sum_{i=1}^{4} \begin{bmatrix} 0 \\ 0 \\ 2\pi n_i \end{bmatrix} \right) = \vec{M}_T \tag{14}$$

These new terms introduced by including the propellers are what we call the gyroscopic moment M_{gyro}. However, in this assignment, since we are applying only step inputs for n_i, we can assume $\dot{n}_i = 0$ to cancel out the second term, leaving us with

$$M_{gyro} = -\omega \times J_p \left(\sum_{i=1}^{4} \begin{bmatrix} 0 \\ 0 \\ 2\pi n_i \end{bmatrix} \right) \tag{15}$$

A final note, is that n_i is directional, so propellers 2 and 4 will have negative values of n for the purpose of angular momentum.

2.2 Calculating Jacobians

To create a linear version of our nonlinear model, we take the derivative of all our states' time-derivatives in terms of the states and inputs. If we let our system of equations in 3-8 be $\vec{f}(\mathbf{states}, \mathbf{inputs})$, we get

$$J_{\mathbf{states}} = \frac{\partial f}{\partial \mathbf{states}} \tag{16}$$
$$J_{\mathbf{inputs}} = \frac{\partial f}{\partial \mathbf{inputs}} \tag{17}$$

which will result in a 12×12 matrix and a 12×4 matrix, respectively, for our 12 states and 4 inputs.

These can be calculated by hand, but I used Matlab's symbolic toolbox to calculate them, as shown in Listing 2. I did it in the x, y, z component vectors described in equation 2:

Listing 2: Calculating Jacobians Symbolically

```
syms x y z u v w phi theta psi p q r n1 n2 n3 n4

states = [x,y,z,u,v,w,phi,theta,psi,p,q,r];
inputs = [n1,n2,n3,n4];
trim_state = [0,0,0,0,0,0,0,0,0,0,0,0];
%n trim = 293 Hz;
trim_input = [293,293,293,293];

phi_dot   = [p + tan(theta)*(q*sin(phi) + r*cos(phi))];
theta_dot = [q*cos(phi) - r*sin(phi)];
psi_dot   = [(q*sin(phi) + r*cos(phi))/cos(theta)];

angle_dot = [phi_dot; theta_dot; psi_dot];
J_angle   = jacobian(angle_dot, states);
J_angle_n = jacobian(angle_dot, inputs);

NED_dot = Unrotate_angles*[u; v; w];
J_NED   = jacobian(NED_dot, states);
J_NED_n = jacobian(NED_dot, inputs);

V_dot = (F_G + F_T)/UAV_mass - cross([p;q;r],[u;v;w]);
J_V   = jacobian(V_dot, states);
J_V_n = jacobian(V_dot, inputs);

M_gyro = cross([p;q;r], Jprop*2*pi*(n1-n2+n3-n4)*[0;0;1]);

% actual code sums all 4 propellers with adjusted +/- signs for terms
M_roll_pitch = thrust_eq(n1)*prop_dist*[ 1/sqrt(2);  1/sqrt(2); 0]) + ...
M_yaw        = power_eq(n1)/(n1*2*pi)*[0; 0; -1]) + ...

M_T = M_roll_pitch + M_yaw;

omega_dot = J\(M_T + M_gyro - cross([p;q;r],J*[p;q;r]));
J_omega   = jacobian(omega_dot, states);
J_omega_n = jacobian(omega_dot, inputs);

J_x =       [ J_NED ; J_V ; J_angle ; J_omega ];
J_x_n =     [ J_NED_n ; J_V_n ; J_angle_n ; J_omega_n ];
```

A note to make is in the the equation for the moment of roll and pitch is the $1/\sqrt{2}$ term. This exists because the propellers are on a diagonal from the x and y axes, so half the moment is in the x, and the other half is in the y.

We can do the same calculations to calculate the output jacobians, with our four outputs:

Listing 3: Output Jacobians

```
h = -z;
a_NED = Unrotate_angles*F_T/UAV_mass;

J_h   = jacobian(h, states);
J_h_n = jacobian(h, inputs);

J_a_NED   = jacobian(a_NED, states);
J_a_NED_n = jacobian(a_NED_n, inputs);

J_y =     [ J_h ; J_a_NED ];
J_y_n =   [ J_h_n ; J_a_NED_n ];
```

We then substitute our trim states and inputs in for our symbolic variables in the four calculated matrices to get A, B, C, D for our linear system, rounded to 3 decimal places. These matrices are now doubles instead of symbolic variables, and so can be used for our linear model in Simulink later.

Listing 4: Symbolic Substitution

```

A = double(round(subs(J_x,   [states, inputs], [trim_state, trim_input])*1000)/1000);
B = double(round(subs(J_x_n, [states, inputs], [trim_state, trim_input])*1000)/1000);
C = double(round(subs(J_y,   [states, inputs], [trim_state, trim_input])*1000)/1000);
D = double(round(subs(J_y_n, [states, inputs], [trim_state, trim_input])*1000)/1000);
```

The resulting matrices were:

$$A = \begin{bmatrix} 0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&-g&0&0&0&0\\ 0&0&0&0&0&0&g&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix} \quad B = \begin{bmatrix} 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0\\ -0.018&-0.018&-0.018&-0.018\\ 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0\\ 1.91&1.91&1.91&1.91\\ 1.553&1.553&1.553&1.553\\ -0.059&-0.059&-0.059&-0.059 \end{bmatrix} \tag{18}$$

$$C = \begin{bmatrix} 0&0&-1&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&-g&0&0&0&0\\ 0&0&0&0&0&0&g&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix} \quad D = \begin{bmatrix} 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0\\ -0.018&-0.018&-0.018&-0.018 \end{bmatrix} \tag{19}$$

Calculating the eigenvalues of A is almost trivial for the stability is almost trivial. Because the only non-zero terms of A are above the main diagonal, we can calculate the eigenvalues of the twelve states as just the values of the main diagonal of A, which is zero for all states, so all states are unstable. We can then use Matlab to calculate the Jordan Canonical form of A to get which states are coupled, we get equation 20 below, with λs along the main diagonal for visibility (since all the eigenvalues are 0, $\lambda = 0$), and the states next to it for comparison.

$$\text{Jordan}(A) = \begin{bmatrix} \lambda & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \lambda & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \lambda & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \lambda & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ u \\ v \\ w \\ \phi \\ \theta \\ \psi \\ p \\ q \\ r \end{pmatrix} \tag{20}$$

We can see that there are Jordan blocks for x, y, z, u ; v, w, ϕ, θ ; ψ, p ; and q, r, making them coupled for this state-space realization. The Jordan blocks, are dependent on which variables are next to each other in our state space representation, so if we change the order of states, we will get a different Jordan matrix. an example of this can be seen below, if we redefine our state, as in equation 21, to get the Jordan matrix in the same equation:

$$\text{jordan}(A^*) = \begin{bmatrix} \lambda & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \lambda & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \lambda & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \lambda & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda \end{bmatrix} \begin{pmatrix} x \\ u \\ \theta \\ q \\ y \\ v \\ \phi \\ p \\ z \\ w \\ \psi \\ r \end{pmatrix} \tag{21}$$

This arrangement of states gives us x, u, θ, q coupled; y, u, ϕ, p coupled; z, w coupled; and ψ, r coupled. This set of couplings is much more intuitive and is more likely to be visible in the dynamics of the system. z, w coupling is likely to visible with Thrust input; x, u, θ, q with Elevator; y, u, ϕ, p with Aileron; and ψ, r with Rudder.

3 Simulink Model

Now that we have all the components of our model, we can create it in Simulink. In simulink, solving differential equations requires integrating the time-derivative of the statem, shown in figure 1.

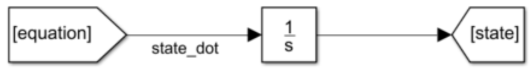

Figure 1: Solving Differential Equations in Simulink

3.1 Nonlinear Model

For the nonlinear system, we solve each of our equations of motion for the state variable's derivative, and create these equations in Simulink using blocks. I did this by compartmentalizing into the set of four x, y, z vectors defined in equation 2. This is shown in figure 2, with the equations for each state vector inside the four subsystem.

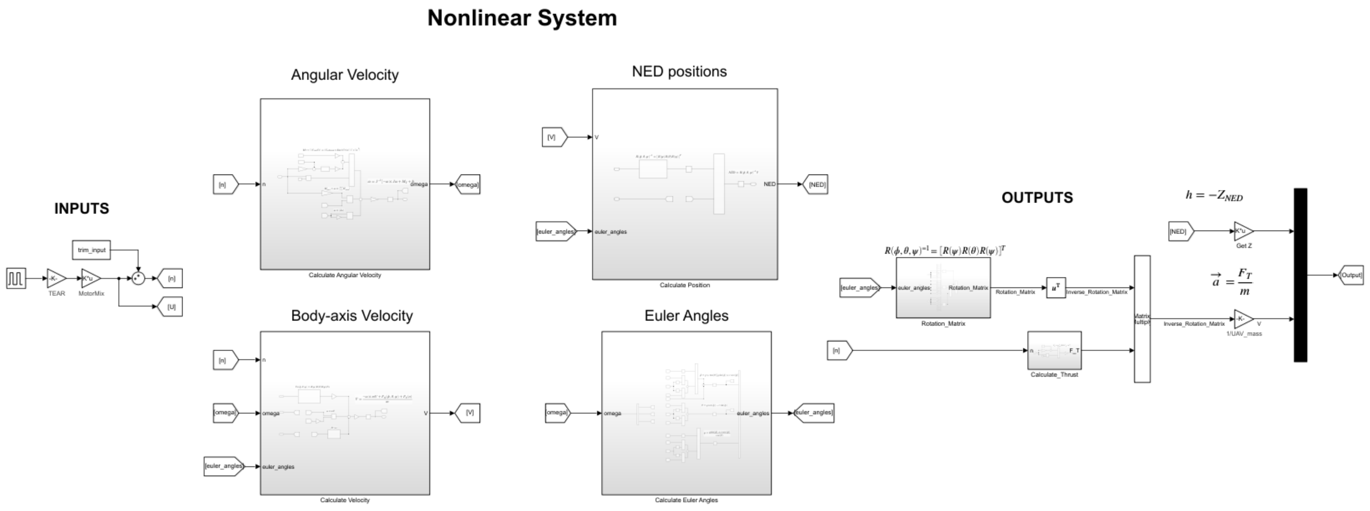

Figure 2: Complete Nonlinear Model

3.1.1 NED Positions

Since the derivative of the NED position is already solved for in equation 8, we simply must program it into simulink. This requires multiplying the body-axis velocity, V by the rotation matrix transpose, $R(\phi, \theta, \psi)^T$, giving the subsystem two inputs. This is shown in figure 3.

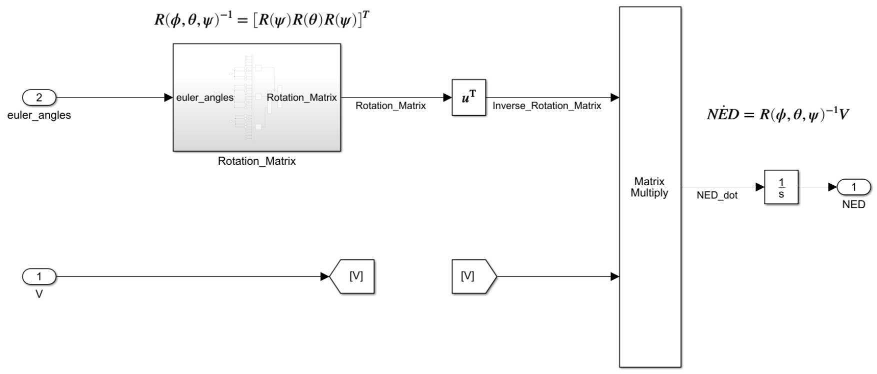

Figure 3: NED position integrator

The rotation matrix, which will also be used later, is made by creating a vector of all the elements of the matrix using three muxs, and reshaping them into the three component matrices $R(\phi)$, $R(\theta)$, $R(\psi)$, which are multiplied together to create R.

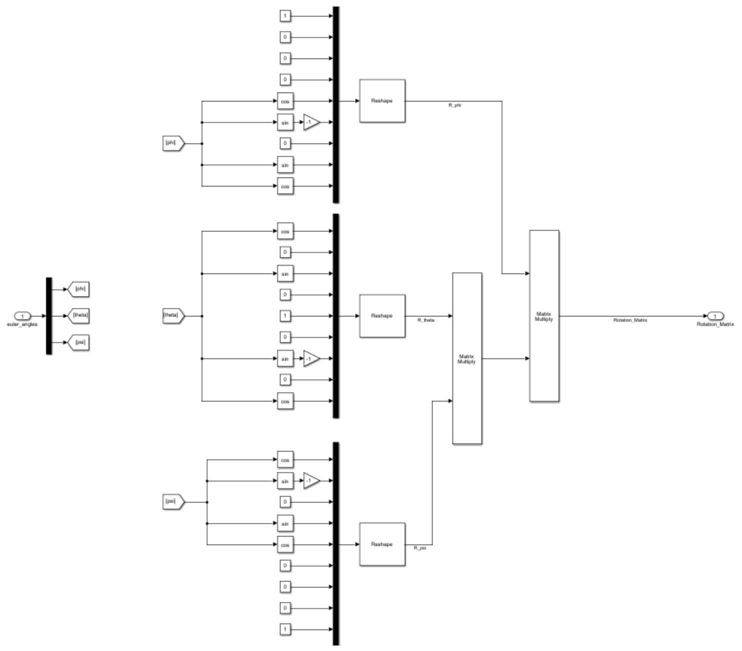

Figure 4: Rotation Matrix

3.1.2 Body-Axis Velocity

We create our differential equation for V from equation 3, and subtract the Coriolis term and divide by m to get

$$\dot{V} = \frac{-\omega \times mV + R(\phi, \theta, \psi)\vec{F}_G + \vec{F}_T}{m} \tag{22}$$

where F_G is mg in the $+z$ NED-axis direction, and F_T is the sum of thrust from all four propellers in the $-z$ body-axis direction using the equation for thrust in the previous assignment,

$$T(n) = C_T(n)\rho n^2 D^4 \tag{23}$$

The means $\dot{V}$ depends on itself, the motor input n, and both the Euler angles and the body-axis angular velocity, giving it three inputs, plus feedback from itself.

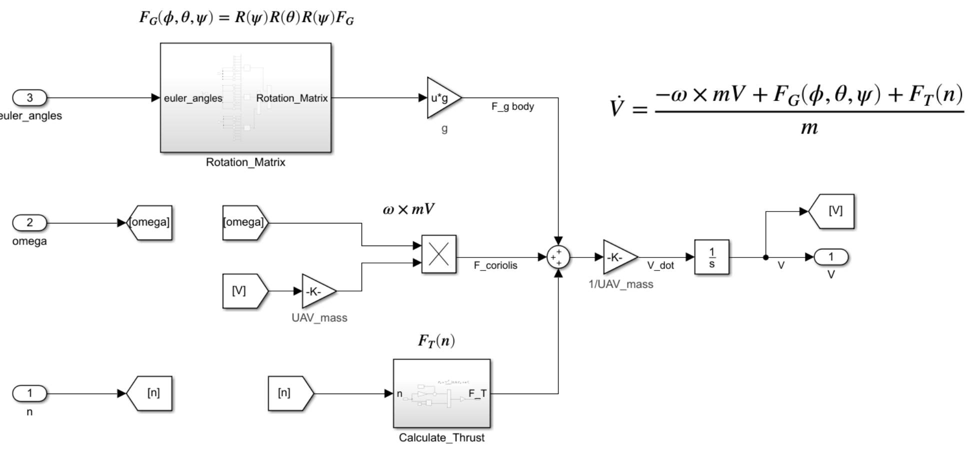

Figure 5: Rotation Matrix

3.1.3 Euler Angles

The Euler angles' time-derivatives are also already solved for in equations 5, 6 and 7, an input of ω and feedback of ϕ and θ. Since each angle is solved for individually, there are no matrix operations, and the three angles are muxed together to create the output.

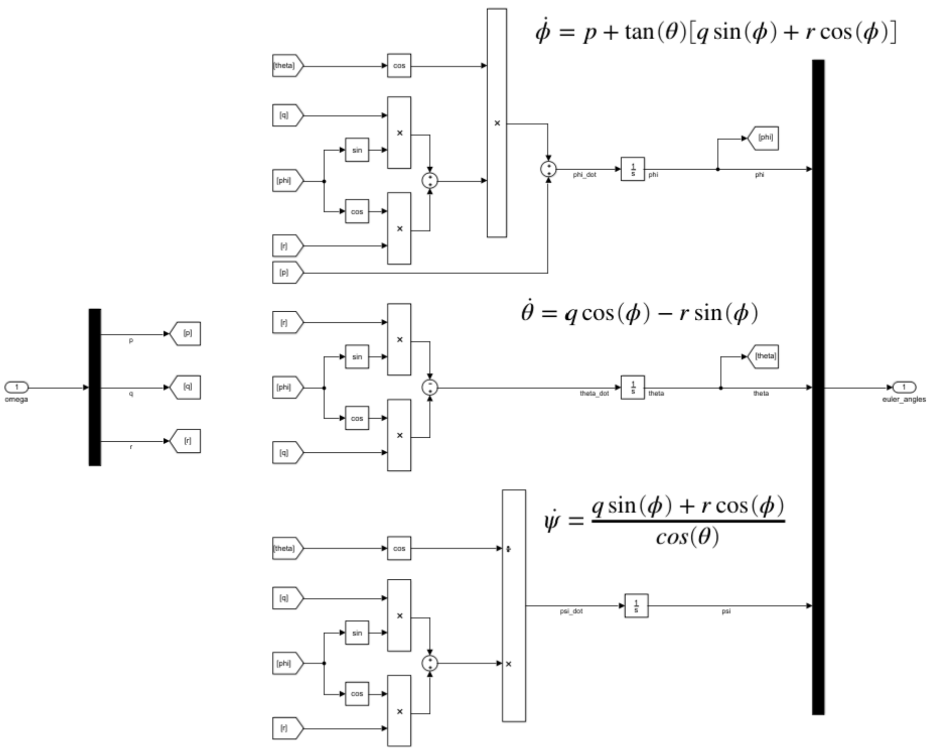
$\dot{\phi} = p + \tan(\theta)[q\sin(\phi) + r\cos(\phi)]$
$\dot{\theta} = q\cos(\phi) - r\sin(\phi)$
$\dot{\psi} = \frac{q\sin(\phi) + r\cos(\phi)}{cos(\theta)}$
omega
euler_angles
phi_dot
theta_dot
psi_dot
phi
theta
psi

Figure 6: Euler Angles

3.1.4 Body-Axis Angular Velocity

Calculating the angular velocity is by far the hardest part of this model. Solving for $\dot{\omega}$ is simple, we just subtract the Coriolis term and left-multiply by J^{-1} to get

$$\dot{\omega} = J^{-1}\left[-\omega \times J\omega + \vec{M}_T + \vec{M}_{gyro}\right] \tag{24}$$

The difficulty comes from the thrust and gyroscopic moments, and making them work with matrix math. Even though the rolling and pitching moments for each propeller just have magnitude

$$M_T = F_T * d_{prop} \tag{25}$$

where F_T is the thrust force and d_{prop} is the distance from the propeller to the center of mass of the UAV, it is split between the x and y directions, with varying plus and minus signs. The solution I ended up with was

$$\vec{M}_T = \left[C_{\text{roll/pitch}}\text{diag}(C_{Ti}(n)) + C_{\text{yaw}}C_P\right](n \odot n) \tag{26}$$

where $C_{\text{roll/pitch}}$ and C_{yaw} are 3×4 matrices and $\odot$ is the symbol for elementwise multiplication. The coefficients of roll/pitch and yaw are

$$C_{\text{roll/pitch}} = \rho D_{prop}^4 d_{prop} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{27}$$

$$C_{\text{yaw}} = \rho D_{prop}^5 \frac{1}{2\pi} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 1 & -1 & 1 \end{bmatrix} \tag{28}$$

with D_{prop} as the diameter of the propeller and rho as the density of air. The diagonal matrix for the coefficient of thrust is necessary because the coefficient is different for each propeller based on its angular velocity, while C_T is a scalar constant. These come from the equations solved for thrust moments in the previous assignment. The implementation of this in Simulink is shown in figure 7.

M_{gyro} was solved for in Section 2.1, leaving us with equation 15 to implement through a matrix,

$$M_{gyro} = \omega \times \left(2\pi J_{prop} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 1 & -1 & 1 \end{bmatrix} n \right) \tag{29}$$

In the Simulink model, the matrix times $2\pi J_{prop}$ is grouped into "Prop_H".

Despite the complexity, the equation only requires the input n and feedback from itself.

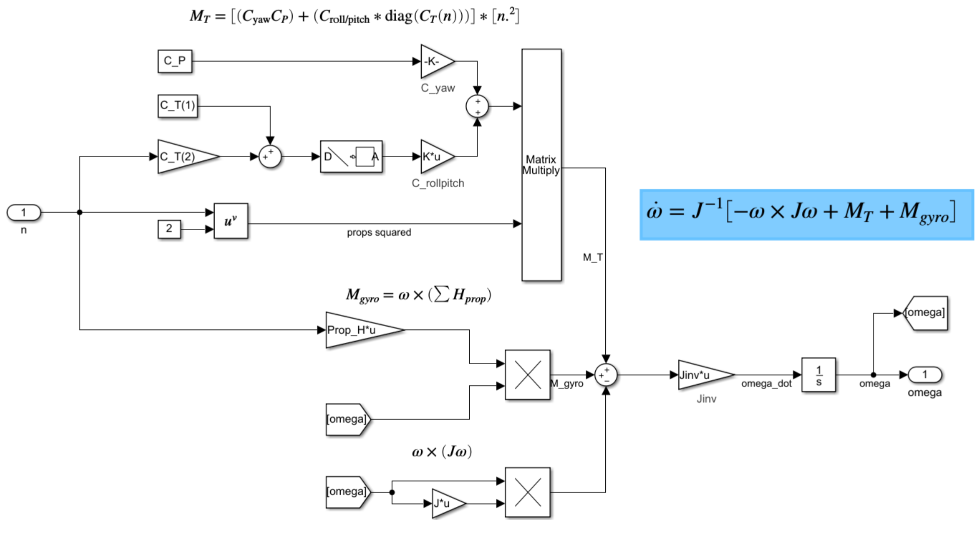
$M_T = [(C_{yaw}C_P) + (C_{roll/pitch} * \text{diag}(C_T(n)))] * [n.^2]$
C_P
-K-
C_yaw
C_T(1)
C_T(2)
D
A
K*u
C_rollpitch
Matrix Multiply
1
n
2
u^v
props squared
M_T
$\dot{\omega} = J^{-1}[-\omega \times J\omega + M_T + M_{gyro}]$
$M_{gyro} = \omega \times (\sum H_{prop})$
Prop_H*u
[omega]
M_gyro
Jinv*u
Jinv
omega_dot
1/s
omega
1
omega
$\omega \times (J\omega)$
[omega]
J*u

Figure 7: Angular Velocity

3.1.5 Outputs

We have outputs of the UAV's height, and acceleration from thrust, as defined in equations 9 and 10. Their implementation is shown in figure 8, muxed together into a 4×1 vector.

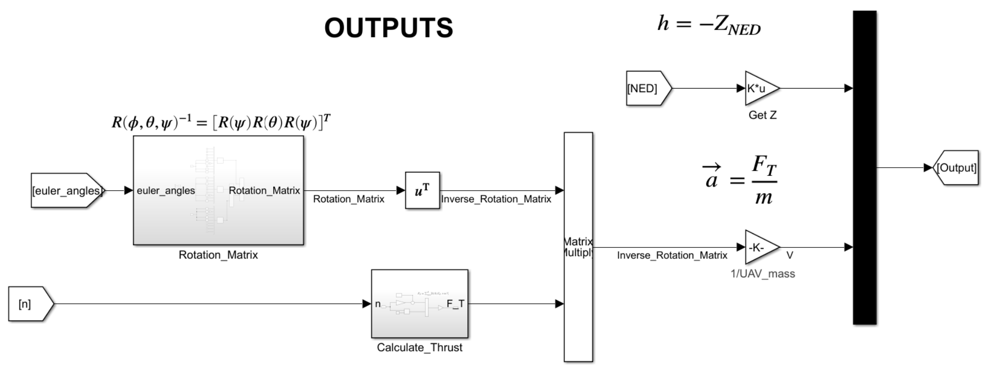

Figure 8: Nonlinear System Outputs

3.2 Inputs

The inputs are in a separate section, because they were defined together for the nonlinear model (n) and the linear model (δU). The difference between them is that the linear model is observing the difference in input from trim, while the nonlinear model requires offset plus trim input.

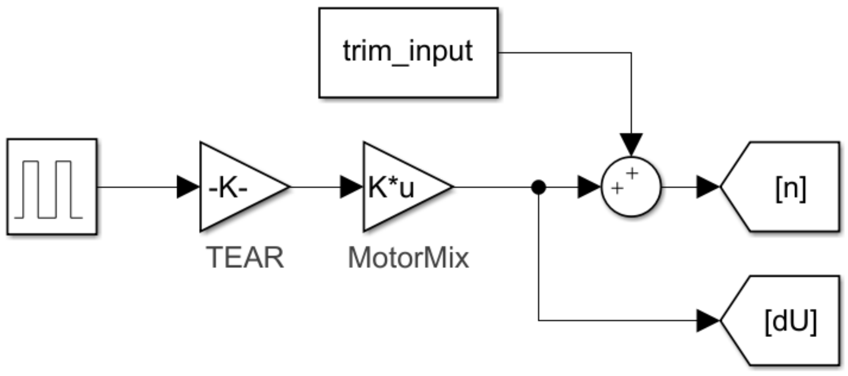

Figure 9: System Inputs

The inputs for this assignment were pulses with a duration of 4 seconds, starting at 1 second, which were provided by the pulse block in figure 9. Since, we were only testing one input of Thrust, Elevator, Aileron or Rudder at a time, the gain $TEAR$ is $e_i \in \mathbb{R}^4$, with e_i as the ith unit vector. This is fed into the motor mixing matrix to get the inputs for n_1, n_2, n_3, n_4,

$$M_{mix} = \begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & -1 & -1 \\ 1 & -1 & 1 & 1 \end{bmatrix} \tag{30}$$

3.3 Linear System

The linear system is a linearized model of the UAV about trim state and input. Trim state is 0 for all states, and trim input is 293 Hz for all motors. We calculated the jacobians to get the matrices for the system in section 2.2, with the results in equations 18 and 19. The resultant model is shown below in figure 10.

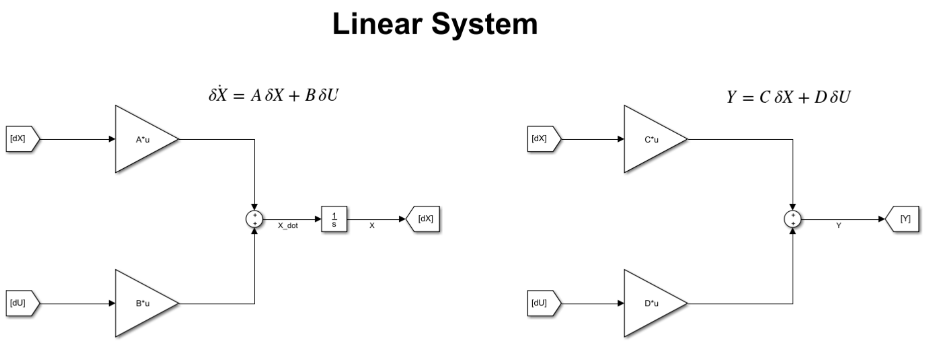
Linear System
$\delta\dot{X} = A\,\delta X + B\,\delta U$
[dX]
A*u
[dU]
B*u
X_dot
$\frac{1}{s}$
X
[dX]
$Y = C\,\delta X + D\,\delta U$
[dX]
C*u
[dU]
D*u
Y
[Y]

Figure 10: Linear System

4 Results

For this assignment there are two purposes for the testing. First, to be able to intuitively confirm the behavior of both the the linear and non-linear models; and second, to observe the differences between the two with some input from trim.

The input to the system will be pulses with a 4 second duration, starting at $t = 1$, and the simulation will last 6 seconds total. The size of the input are scaled so that we don't reach the singularities of our model, ie $\phi,\ \theta = \pi/2$, which is relevant for the Elevator and Aileron inputs.

The graphs for this section will only show state variables and outputs of the model that are non-zero for a given input. Note that the a_z component of the acceleration from thrust for the nonlinear model has been summed with g, to match the linear model, and show changes from equilibrium.

4.1 Thrust Pulse

For thrust, we input a signal with magnitude 5 into the system, increasing the speed up all four propellers 5 Hz. As expected, the only two states that changed were the z position and velocity states, Z_{NED} and w. They are negative, because the up direction is defined as $-z$. The result is shown in figure 11.

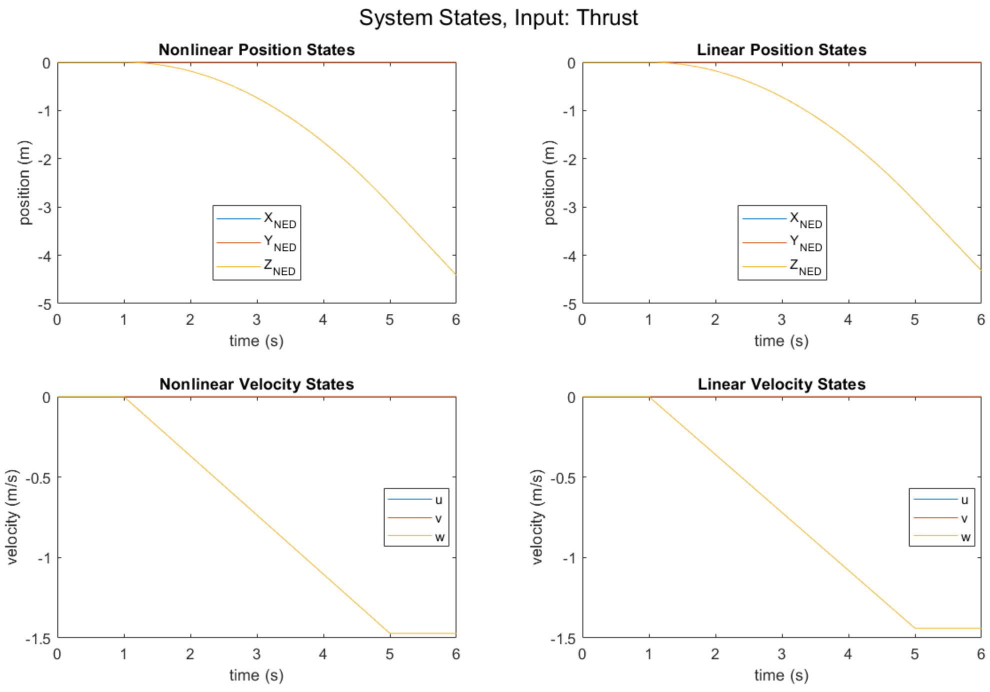

Figure 11: Linear and Nonlinear Model Position States for Thrust Input

The thrust of the nonlinear model is slightly larger, because the thrust is approximately quadratic, so away from trim input, increasing the propeller speed produces a larger increase in thrust, and decreasing it produces a smaller decrease in thrust. This slight change can be seen in figure 12.

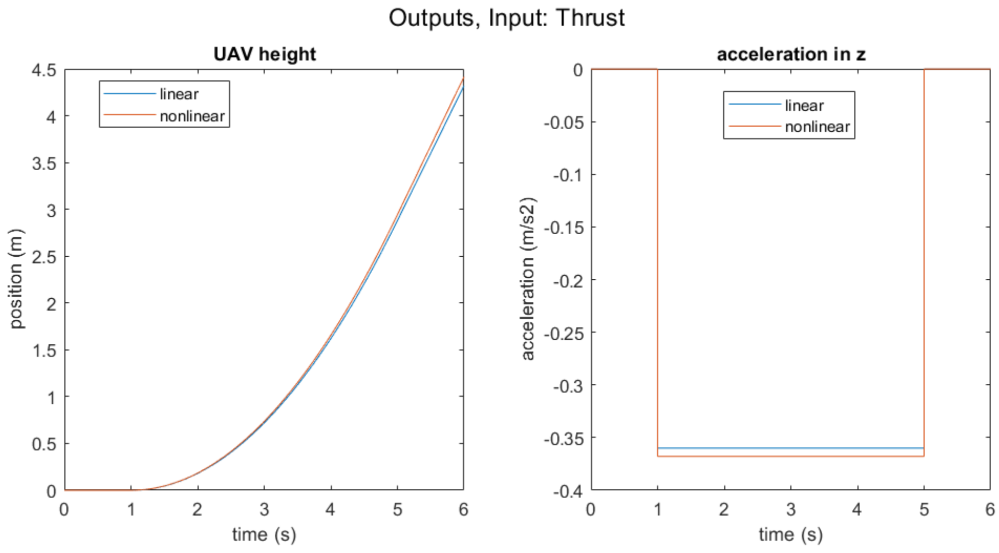

Figure 12: Linear and Nonlinear Model Outputs for Thrust Input

For both systems, given a constant input for thrust, Z and w are both unstable. Given an impulse input for thrust, Z is unstable, since it keeps growing even after the input has been removed. In an actual physical system, this second instability would not exist, since drag would eventually slow the UAV down, and it would also create a cap on w, making it stable for a bounded input. Since the output is just $-Z$, it is also unstable when Z is unstable. It is also clear that Z and w are coupled in both systems from this, which makes intuitive sense and supports the results of equation 21.

The behavior also matches the eigenvalues calculated with a repeated root at 0, and the behavior of the subsystem of a double integrator, with linear velocity from a constant input, and quadratic position.

4.2 Elevator and Aileron Pulse

Elevator and Aileron inputs are very similar for the UAV. Because the propellers are all at 45^o from the x and y axes, the moments of roll and pitch are the same magnitude. However, since the UAV isn't a square mass, the moment of inertia about the y axis, J_{yy}, is larger than the moment about the x axis, J_{xx}, so the angular acceleration of θ is slightly smaller than ϕ. Because of this, the Elevator pulse for θ is slightly larger than the Aileron, at 0.015 Hz and 0.01 Hz changes, respectively. I will only be showing the Elevator Pulse, because the results are the same, exchanging θ and q for ϕ and p, and the $-x$ values for $+y$ in position, and velocity.

The main differences between the linear and nonlinear system states, is the fact that there are changes in Z_{NED} and w in the nonlinear system, but not in the linear one. The lack of change in the linear system is caused by the fact that at equilibrium, the Jacobian of w is zero for the states, and w is what determines Z. Whereas the nonlinear model, it takes into account that significant changes in ϕ and θ create changes in w if they aren't compensated for by increasing thrust. These differences are shown in figure 13.

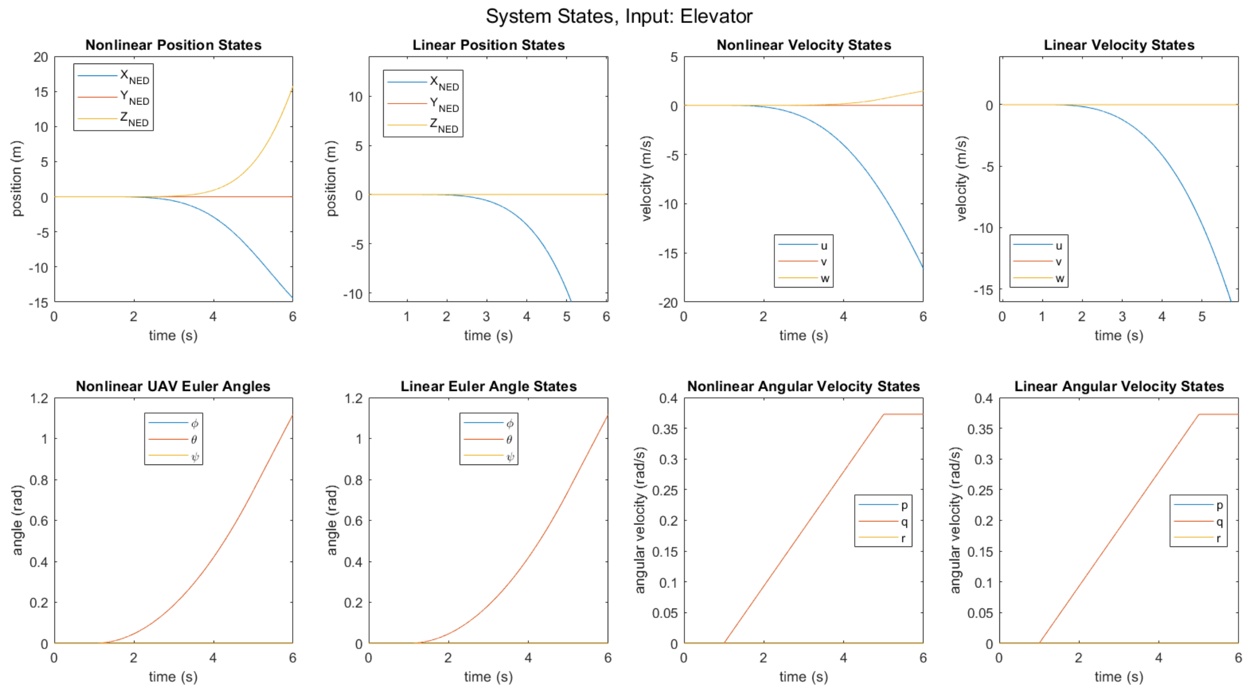

Figure 13: Linear and Nonlinear Model States for Elevator Input

Additionally, in the nonlinear thrust equation, there is a slight increase in total thrust to the system, despite the sum of propeller speeds remaining constant, because of the quadratic thrust equation, as described above in subsection 4.1, though the magnitude of a_z is too large for it to be observed in figure 14.

We can observe that, similar to thrust, ϕ and p, and θ and q are coupled pairs, and they have the same behavior of a double integrator. We can also observe that X and Y are both unstable and coupled to u and v, for Elevator and Aileron inputs, respectively. This supports our results for couplings in equation 21.

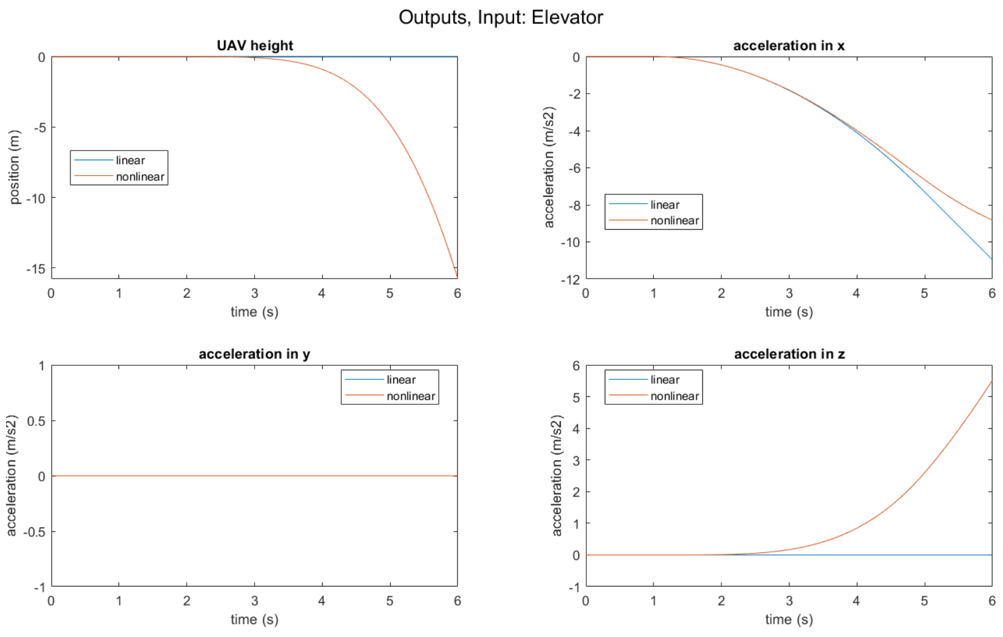

Figure 14: Linear and Nonlinear Model Outputs for Elevator Input

4.3 Rudder Pulse

The Rudder pulse input should create only a yawing moment in the linear model, but in the nonlinear model, there is a slight thrust generated from the quadratics of the propeller thrust equation, similar to the other inputs. Apart from this, the two models are exactly the same for this input, since there isn't any nonlinearity in rotating about ψ at trim.

The dynamics match those for the other inputs of a double integrator. Like the z axis position and velocity, the z axis angular position and velocity are only coupled to each other, which matches the results of the model. The results of this are shown in figures 15 and 16, below.

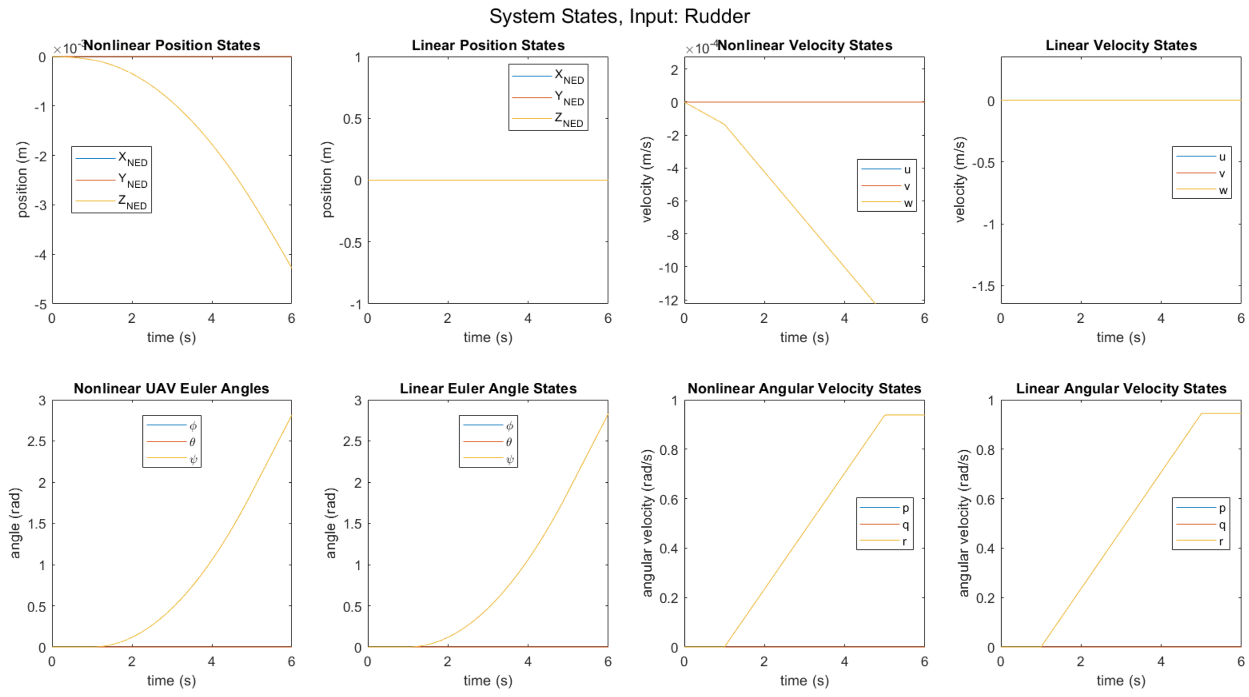

Figure 15: Linear and Nonlinear Model States for Rudder Input

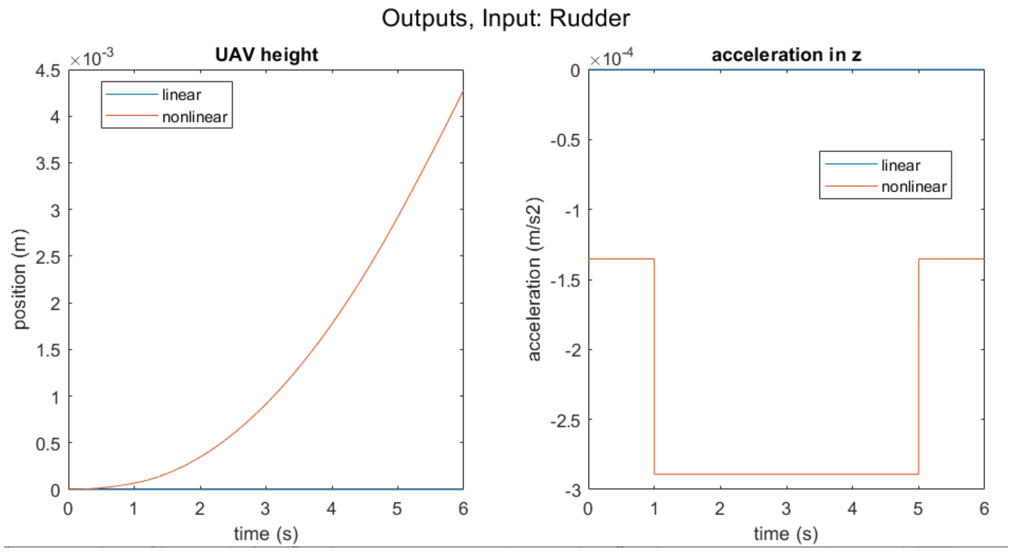

Figure 16: Linear and Nonlinear Model Outputs for Rudder Input

5 Conclusion

All our states of our linear system have eigenvalues of 0, so are unstable with constant inputs. This makes sense since there is no damping term for our system yet. The nonlinear system is also unstable from this fixed point because the linear system is, and it captures the behavior of the nonlinear system close to the point. Physically, this makes sense as well, because any offset from trim states with no correcting input will lead to the UAV drifting away or spiraling and crashing.

Position and velocity states are all coupled for both Cartesian and angular states, and the X, Y position and velocities are additionally coupled with the θ, ϕ position and velocity states. This makes sense, because a rotation in θ makes the thrust point in the $-X$ direction, and a rotation in ϕ makes the thrust point in the $+Y$ direction. It is possible to define other couplings, like the Jordan matrix in equation 20, but they are less intuitive.

For inputs only along the z-axis, both Cartesian and angular, the linear model is adequate. However, it does not handle rotation along the x, y-axes very well. We didn't test multiple inputs at the same time for the linear model to see how it behaves, or start the model from a non-trim state.

A properly designed controller should be able to control the UAV near trim state with the linear system, since we have four inputs and four coupled sets of states to controller, but will struggle with large corrections.

References

[1] https://commons.erau.edu/cgi/viewcontent.cgi?article=2057&context=publication