University of Colorado Boulder

ASEN 3128 - Aircraft Dynamics

Assignment 5 - Afternoon Section

Assignment 5

Author:
Benjiman Smith
105979656

Professor:
D Lawrence

27 February, 2020

College of Engineering & Applied Science
UNIVERSITY OF COLORADO BOULDER

Contents

List of Figures

The objective of this assignment was to utilize feedback gains calculated in Assignment 4 for linearized lateral and longitudinal rotation dynamics of the quad copter as an "inner loop" control law, while implementing a new gain value as the outer loop control law. Constraints were set in place to calculate this third gain value. Simulations were run using this new control system for open loop commands, corresponding with inertial displacements y_E and x_E to change by one meter in 2.0 seconds. These simulations were then compared to experimental data to determine the differences between the predicted and measured behavior.

I. Nomenclature

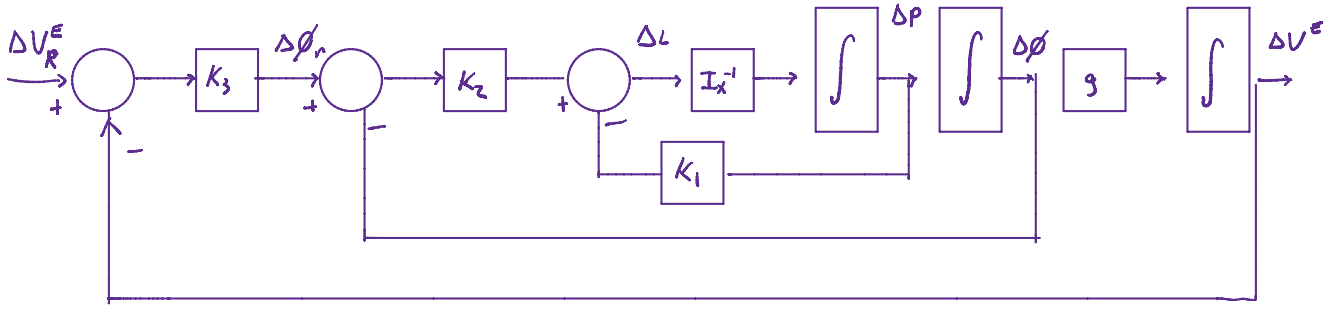
m	=	mass $[kg]$
r	=	body to motor distance $[m]$
I_x	=	moment of inertia in the body x direction $[kgm^2]$
I_y	=	moment of inertia in the body y direction $[kgm^2]$
I_z	=	moment of inertia in the body z direction $[kgm^2]$
u^E	=	inertial velocity in the x direction in body coordinates $[m/s]$
v^E	=	inertial velocity in the y direction in body coordinates $[m/s]$
w^E	=	inertial velocity in the z direction in body coordinates $[m/s]$
$\dot{u^E}$	=	derivative of the inertial in the x direction in body coordinates $[m/s^22]$
$\dot{v^E}$	=	derivative of the inertial velocity in the y direction in body coordinates $[m/s^2]$
$\dot{w^E}$	=	derivative of the inertial velocity in the z direction in body coordinates $[m/s^2]$
$\Delta\dot{u^E}$	=	derivative of the inertial in the body x direction deviated from trim $[m/s^2]$
$\Delta\dot{v^E}$	=	derivative of the inertial velocity in the body y direction deviated from trim $[m/s^2]$
$\Delta\dot{w^E}$	=	derivative of the inertial velocity in the body z direction deviated from trim $[m/s^2]$
p	=	inertial angular velocity component in the body x direction component in the inertial frame $[radian/s]$
q	=	inertial angular velocity component in the body y direction component in the inertial frame $[radian/s]$
r	=	inertial angular velocity component in the body z direction component in the inertial frame $[radian/s]$
Δp	=	roll rate deviated from trim $[radian/s^2]$
Δq	=	pitch rate deviated from trim $[radian/s^2]$
$\dot{p}$	=	roll rate derivative $[radian/s^2]$
$\dot{q}$	=	pitch rate derivative $[radian/s^2]$
$\dot{r}$	=	yaw rate derivative $[radian/s^2]$
$\Delta\dot{p}$	=	roll rate derivative deviated from trim $[radian/s^2]$
$\Delta\dot{q}$	=	pitch rate derivative deviated from trim $[radian/s^2]$
$\Delta\dot{r}$	=	yaw rate derivative deviated from trim $[radian/s^2]$
ϕ	=	bank angle $[radian]$
θ	=	elevation angle $[radian]$
ψ	=	azimuth angle $[radian]$
$\dot{\phi}$	=	bank angle rate of change $[radian/s]$
$\dot{\theta}$	=	elevation angle rate of change $[rradian/s]$
$\dot{\psi}$	=	azimuth angle rate of change $[radian/s]$
$\Delta\dot{\phi}$	=	bank angle rate of change deviated from trim $[radian/s]$
$\Delta\dot{\theta}$	=	elevation angle rate of change deviated from trim $[radian/s]$
$\Delta\dot{\psi}$	=	azimuth angle rate of change deviated from trim $[radian/s]$
X_E	=	position vector in the x direction in inertial coordinates $[m]$
Y_E	=	position vector in the y direction in inertial coordinates $[m]$
Z_E	=	position vector in the z direction in inertial coordinates $[m]$
f_1	=	trim force exerted by motor 1 $[N]$
f_2	=	trim force exerted by motor 2 $[N]$
f_3	=	trim force exerted by motor 3 $[N]$
f_4	=	trim force exerted by motor 4 $[N]$

II. Question 1

In order to understand how to approach calculating the third gain value for both the lateral and longitudinal rotation dynamics models, hand analysis was completed.

Problem 1, assignment 5

Friday, February 21, 2020 10:54 AM



$$\Delta L = \overbrace{-K_2\Delta\phi + K_2K_3(\Delta V_r^E - \Delta V^E)}^{\oplus \text{ SIDE OF } \Delta L} + \overbrace{(-K_1\Delta P)}^{\ominus \text{ SIDE OF } \Delta L}$$

(4.9,3) $$\Delta\dot{P} = \frac{1}{I_x}\Delta L \Rightarrow \Delta\dot{P} = \frac{K_2K_3\Delta V_R^E}{I_x} - \frac{K_2K_3V^E}{I_x} - \frac{K_2\Delta\phi}{I_x} - \frac{K_1\Delta P}{I_x}$$

(4.9,4) $$\Delta\dot{\phi} = \Delta P$$

(4.9,7) $$\Delta\dot{V}^E = g\cdot\Delta\phi$$

$$\Rightarrow \begin{bmatrix} \Delta\dot{P} \\ \Delta\dot{\phi} \\ \Delta\dot{V}^E \end{bmatrix} = \begin{bmatrix} -\frac{K_1}{I_x} & -\frac{K_2}{I_x} & -\frac{K_2K_3}{I_x} \\ 1 & 0 & 0 \\ 0 & g & 0 \end{bmatrix} \begin{bmatrix} \Delta P \\ \Delta\phi \\ \Delta V_E \end{bmatrix} + \begin{bmatrix} K_2K_3\Delta V_R^E \\ 0 \\ 0 \end{bmatrix}$$

$$A - \lambda I = \left[\begin{array}{c|c|c} -\frac{K_1}{I_x} - \lambda & -\frac{K_2}{I_x} & -\frac{K_2K_3}{I_x} \\ \hline 1 & -\lambda & 0 \\ \hline 0 & g & -\lambda \end{array}\right]$$

$$\text{DET}[A-\lambda I] = \left(-\frac{K_1}{I_x} - \lambda\right)(\lambda^2) - \left(-\frac{K_2}{I_x}\right)(-\lambda) + \left(\frac{-K_2K_3}{I_x}\right)(g)$$

$$-\frac{K_1}{I_x}\lambda^2 - \lambda^3 - \frac{K_2\lambda}{I_x} - \frac{K_2K_3}{I_x}g = 0$$

$$\lambda_1 = -2,\ \lambda_2 = -20$$

$$\Rightarrow \lambda^3 + \frac{K_1}{I_x}\lambda^2 + \frac{K_2\lambda}{I_x} + \frac{K_2K_3}{I_x}g = 0$$

Utilizing matlab's symbolic solver, coupled with pre-determined K_1 & K_2 values ($K_1 = 0.0015$, $K_2 = 0.0027$) λ_1, λ_2, & λ_3 could be found for a range of K_3 values ($K_3 = -0.5 \Rightarrow 0.5$)

$\Rightarrow$ The condition that must be met by λ_1, λ_2, & λ_3 are shown below:

FROM CLASS, WE WERE GIVEN THE FOLLOWING EQUATION

$$e^{\lambda_i t} = e^{nt} \cdot e^{j\omega t}$$

IN WHICH THE $e^{j\omega t}$ TERM IS OSCILLOTORY. SINCE THERE IS NO OSCILLATION PRESENT IN OUR SIMULATION, AS OUR CHARACTERISTIC EQUATION DOESN'T OUTPUT IMAGINARY NUMBERS, $\omega \Rightarrow 0$, THEREFORE THE EQUATION SIMPLIFIES TO:

$$e^{\lambda_i t} = e^{nt}e^{0} \Rightarrow e^{\lambda_i t} = e^{nt}$$

$$\Rightarrow \ln(e^{\lambda t}) = \ln(e^{nt}) \Rightarrow \lambda t = nt$$

$$\therefore \boxed{\lambda_i = n}$$

THIS ALLOWS FOR US TO SOLVE FOR $\lambda_1, \lambda_2, \& \lambda_3$ easily

$\tau \equiv$ TIME CONSTANT

$\tau \leq 1.25$ sec (FROM ASSIGNMENT)

$$e^{n\tau} = 0.37 \Rightarrow \ln(e^{n\tau}) = \ln(0.37)$$

$$\Rightarrow n\tau = -1 \Rightarrow \tau \leq -\frac{1}{n} \Rightarrow \tau \leq -\frac{1}{\lambda_3}$$

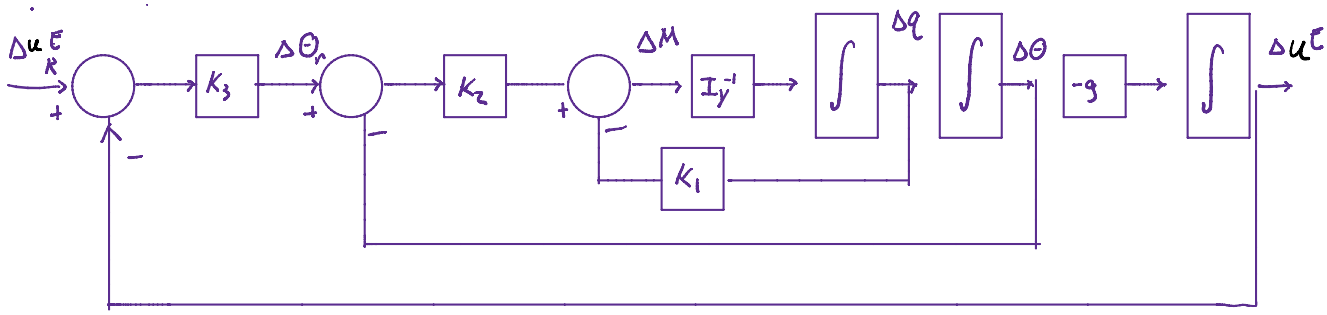
IN THIS CASE, $n = \lambda_1$

$$\lambda_{1,2,3} \geq -\frac{1}{\tau} = -\frac{1}{1.25\ \text{sec}} = \boxed{\lambda_{1,2,3} \geq -0.8\ \text{sec}}$$

USING THIS CONSTRAINT FOR THE EIGENVALUES, THE APPROPRIATE K_3 VALUE COULD BE FOUND BY INDEXING THROUGH THE EIGENVALUES ≥ -0.8 SEC, THEN FINDING A K_3 VALUE IN THE FOUND INDEX

$$\Rightarrow \boxed{K_3 = 0.0470}$$

THESE SAME CALCULATIONS WERE ALSO DONE ALONG THE LONGITUDINAL



(+) SIDE OF ΔL (−) SIDE OF ΔL

$$\Delta M = -K_2\Delta\theta + K_2K_3(\Delta w_R^E - \Delta w^E) + (-K_1\Delta q)$$

(4.9,8) $$\Delta\dot{q} = \frac{1}{I_y}\Delta M \Rightarrow \Delta\dot{q} = \frac{K_2K_3\Delta w_R^E}{I_y} - \frac{K_2K_3 w^E}{I_y} - \frac{K_2\Delta\theta}{I_y} - \frac{K_1\Delta q}{I_x}$$

(4.9,4) $$\Delta\dot{\theta} = \Delta q$$

(4.9,7) $$\Delta\dot{u}^E = -g\Delta\theta$$

$$\Rightarrow \begin{bmatrix} \Delta\dot{q} \\ \Delta\dot{\theta} \\ \Delta\dot{u}^E \end{bmatrix} = \begin{bmatrix} -\frac{K_1}{I_y} & -\frac{K_2}{I_y} & -\frac{K_2K_3}{I_y} \\ 1 & 0 & 0 \\ 0 & -g & 0 \end{bmatrix} \begin{bmatrix} \Delta q \\ \Delta\theta \\ \Delta u^E \end{bmatrix} + \begin{bmatrix} K_2K_3\Delta u_R^E \\ 0 \\ 0 \end{bmatrix}$$

$$A - \lambda I = \begin{bmatrix} -\frac{K_1}{I_y} - \lambda & -\frac{K_2}{I_y} & -\frac{K_2K_3}{I_y} \\ 1 & -\lambda & 0 \\ 0 & -g & -\lambda \end{bmatrix}$$

$$\text{DET}[A-\lambda I] = \left(-\frac{K_1}{I_y} - \lambda\right)(\lambda^2) - \left(-\frac{K_2}{I_y}\right)(-\lambda) + \left(-\frac{K_2K_3}{I_y}\right)(g)$$

$$-\frac{K_1}{I_y}\lambda^2 - \lambda^3 - \frac{K_2\lambda}{I_y} - \frac{K_2K_3}{I_y}g = 0$$

$$\Rightarrow \lambda^3 + \frac{K_1}{I_y}\lambda^2 + \frac{K_2\lambda}{I_y} + \frac{K_2K_3}{I_y}g = 0$$

Utilizing matlab's symbolic solver, coupled with pre-determined K_1 & K_2 values ($K_1 = 0.0015$, $K_2 = 0.0027$)

$\lambda_1, \lambda_2,$ & λ_3 could be found for a range of K_3 values ($K_3 = -0.5 \Rightarrow 0.5$)

$\Rightarrow$ The condition that must be met by $\lambda_1, \lambda_2,$ & λ_3 are shown below:

FROM CLASS, WE WERE GIVEN THE FOLLOWING EQUATION

$$e^{\lambda t} = e^{nt} \cdot e^{j\omega t}$$

IN WHICH THE $e^{j\omega t}$ TERM IS OSCILLOTORY. SINCE THERE S NO OSCILLATION PRESENT N OUR SIMULATION, AS OUR CHARACTERISTIC EQUATION OESN'T OUTPUT IMAGINARY NUMBERS, $\omega \Rightarrow 0$, THEREFORE THE EQUATION SIMPLIFIES TO:

$$e^{\lambda_1 t} = e^{nt} e^{0} \Rightarrow e^{\lambda_1 t} = e^{nt}$$

$$\Rightarrow \ln(e^{\lambda t}) = \ln(e^{nt}) \Rightarrow \lambda t = nt$$

$$\therefore \boxed{\lambda_1 = n}$$

THIS ALLOWS FOR US TO SOLVE FOR $\lambda_1, \lambda_2, \& \lambda_3$ easily

$\tau \equiv$ TIME CONSTANT

$\tau \leq 1.25$ sec (FROM ASSIGNMENT)

$$e^{n\tau} = 0.37 \Rightarrow \ln(e^{n\tau}) = \ln(0.37)$$

$$\Rightarrow n\tau = -1 \Rightarrow \tau \leq -\frac{1}{n} \Rightarrow \tau \leq -\frac{1}{\lambda_3}$$

↗

IN THIS CASE, $n = \lambda_1$

$\lambda_{1,2,3} \leq -\frac{1}{\tau} = -\frac{1}{1.25\ \text{sec}} =$ **$\lambda_{1,2,3} \leq -0.8$ sec**

USING THIS CONSTRAINT FOR THE EIGENVALUES, THE APPROPRIATE K_3 VALUE COULD BE FOUND BY INDEXING THROUGH THE EIGENVALUES ≤ -0.8 SEC, THEN FINDING A K_3 VALUE IN THE FOUND INDEX

$\Rightarrow$ **$K_3 = -0.0470$**

Matlab was utilized in these calculations in order to find the three new eigenvalues for a range of K_3 values. The locus of these calculated eigenvalues for both the lateral and longitudinal simulations were found by using the *eig* function to find the eigenvalues from their associated A matrix, and using the plot function to plot the eigenvalues with respect to the imaginary and real axis.

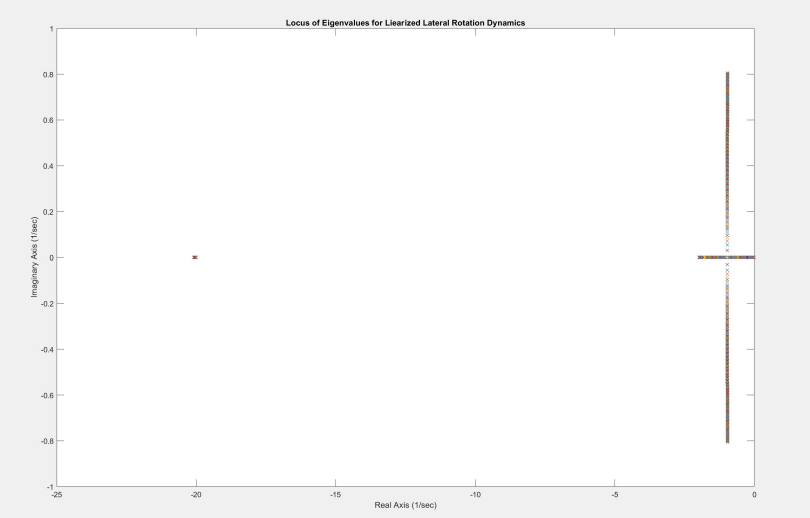


Fig. 1 Locus of Lateral rotation dynamics simulation

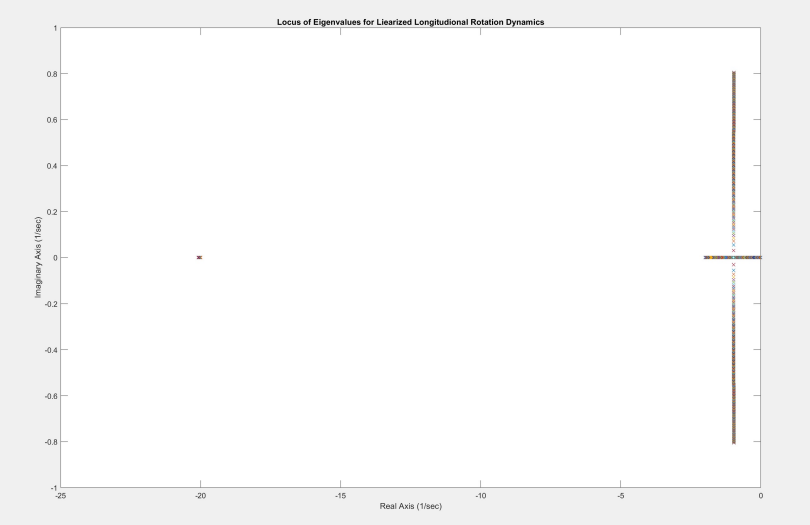


Fig. 2 Locus of Longitudinal rotation dynamics simulation

After developing the plots of both the lateral and longitudinal locus, the appropriate K_3 values for both cases were found using the $find$ function, with a condition that the index from which K_3 eigenvalue couple was ≤ -0.8 seconds, as shown by the lines of code below.

```
index1 = find(Lambda3(3,:) > -0.8, 1, 'last');
k3 = K3(index1+1);
```

Fig. 3 Indexing code used to find K_3 from eigenvalue

III. Question 2

In order to design the feedforeward commands for Δv_R^E and Δu_R^E to cause the corresponding inertial displacements Δy_R^E and Δx_R^E to change by 1.0 m in 2.0 sec, new control laws needed to be implemented into the ODE function developed in Assignment 4. These control law changes, derived from the functional block diagrams shown below, featured the additional gain term.

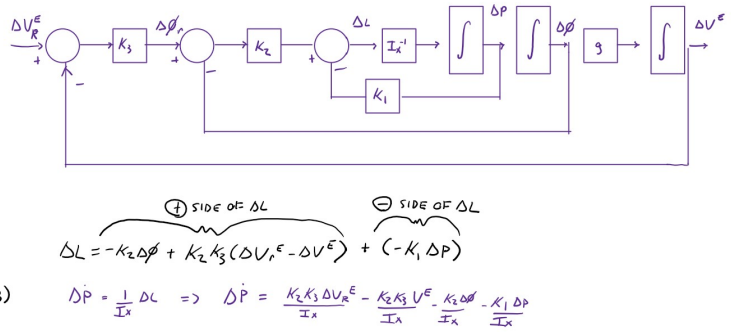


Fig. 4 Lateral simulation block diagram

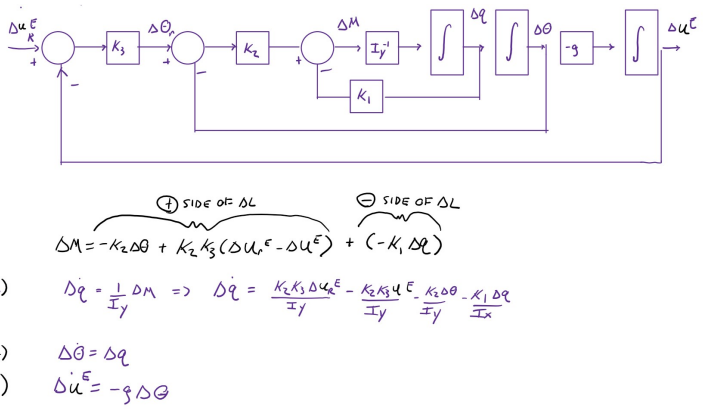


Fig. 5 Longitudinal simulation block diagram

The changes to both the $\Delta\dot{q}$ and $\Delta\dot{p}$ equations were implemented into the ODE function as follows.

```
deltapdot = ((-k_1*deltap)/I_x) -((k_2*deltaphi)/I_x -((k_2*k_3)*(deltavr-deltav))/I_x  ); % roll rate derrivative
deltaqdot = ((-k_4*deltaq)/I_y) -((k_5*deltatheta)/I_y -((k_5*k_6)*(deltaur-deltau))/I_y); % pitch rate derrivative
```

Fig. 6 Implementation of new $\Delta\dot{q}$ and $\Delta\dot{p}$ equations

in order to initiate the feedforeward commands for Δv_R^E and Δu_R^E, the ODE function developed in Assignment 4 was modified again with an if statement which allowed for Δv_R^E and Δu_R^E to be set to non-zero constants over a specified time, which was two seconds. Due to the goal of causing the corresponding inertial displacements Δy_R^E and Δx_R^E to change by 1.0 meters in 2.0 seconds independently, the non-zero constant values assigned to v_R^E and Δu_R^E were 0.5 m/s. After this two second time interval, an $else$ statement was used to set Δv_R^E and Δu_R^E back to zero m/s. This set of statements is shown below for the lateral case, including Δv_R^E as the feedforeward command, and the longitudinal case, including Δu_R^E as the feedforeward command.

```
if t <= 2 % time interval for added velocity
       deltavr = 0.5; % set delta vr = 0.5 m/s
else % else statement to return velocity to zero
       deltavr = 0;

   end
```

(a) Lateral Case

```
if t <= 2 % time interval for added velocity
       deltaur = 0.5; % set delta ur = 0.5 m/s
else % else statement to return velocity to zero
       deltaur = 0;

   end
```

(b) Longitudinal Case

Fig. 7 If/Else statements for feedforeward commands

Implementing these feedforeward commands for both the Lateral and Longitudinal simulations showed the following behaviors.

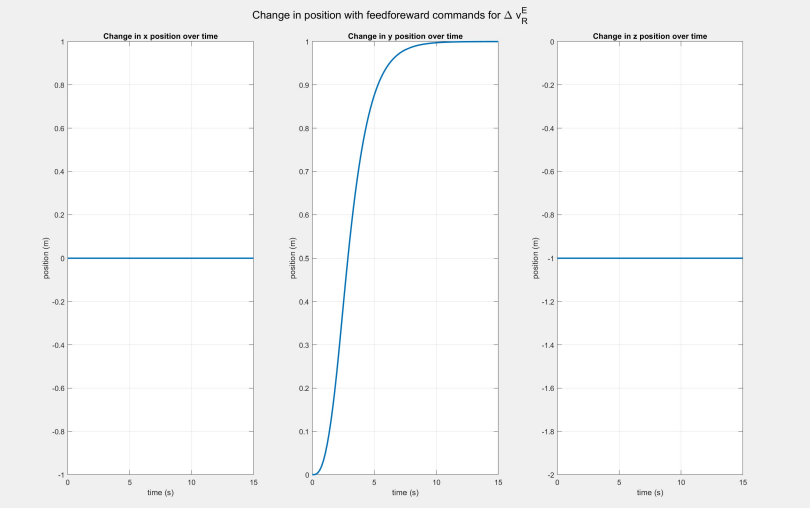


Fig. 8 Lateral Change in position over time

As seen in figure 8, the position in the $\hat{y}$ increases to 1 meter displacement in a time period more than two seconds, which was the main goal of this simulation. This lack of responsiveness to the change in position is due to the time constant constraints of this lab, which resulted in a substantially small 3rd gain value for the lateral simulation. The small K_3 values are what caused the slow rise to one meter $\hat{y}$ direction. Another constraint of this lab was the constraint of only using real eigenvalues, for which time constants less than 1.25 seconds were not sufficient to model the given problem. As seen in the equation for roll rate derivative derived in question 1:

$$\Delta\dot{p} = -\frac{K_1\Delta p}{I_x} - \frac{K_2\Delta\phi}{I_x} - \frac{K_3K_2(\Delta v_R^E - \Delta v^E)}{I_x} \tag{1}$$

For small K_3 values, the value of $\Delta\dot{p}$ is smaller, therefore the rolling of the quadcopter becomes slower as K_3 gets smaller. This justifies why the simulation's response time is slower than expected.

A plot of the quadcopter's position was created to see the behavior of the quadcopter during the time of the applied Δv_R^E.

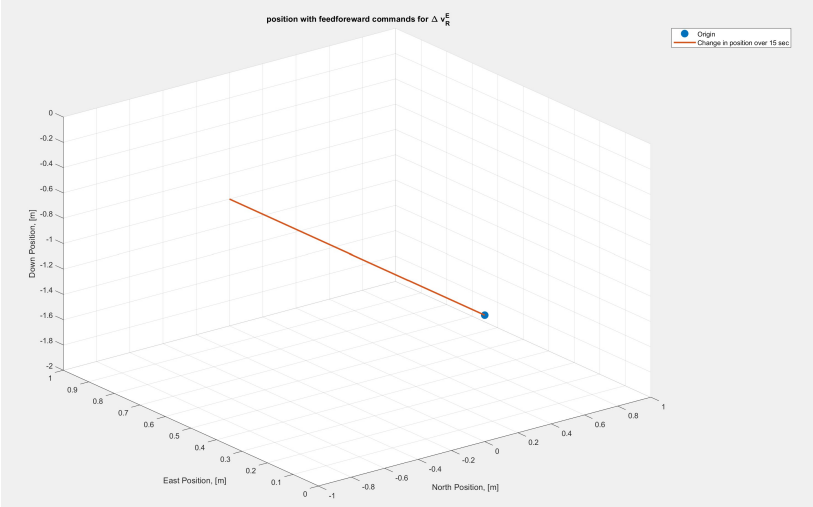


Fig. 9 Lateral Change in position over time

As seen from figure 9 the quadcopter translates steadily in the positive east direction. This behavior was expected, as the Δv_R^E applied to the quadcopter would cause a small initial positive spike in bank, as a positive Δv_R^E requires a positive bank angle. This spike was corrected by the implemented K_1 values, and the bank angle returned to zero radians quickly. Similar results are true of the roll rate, as the initial velocity caused an initial positive spike in roll rate due to the positive Δv_R^E, but the K_1 gain value quickly brought the roll rate back to zero radians per second. These results are also clearly seen in the following plot.

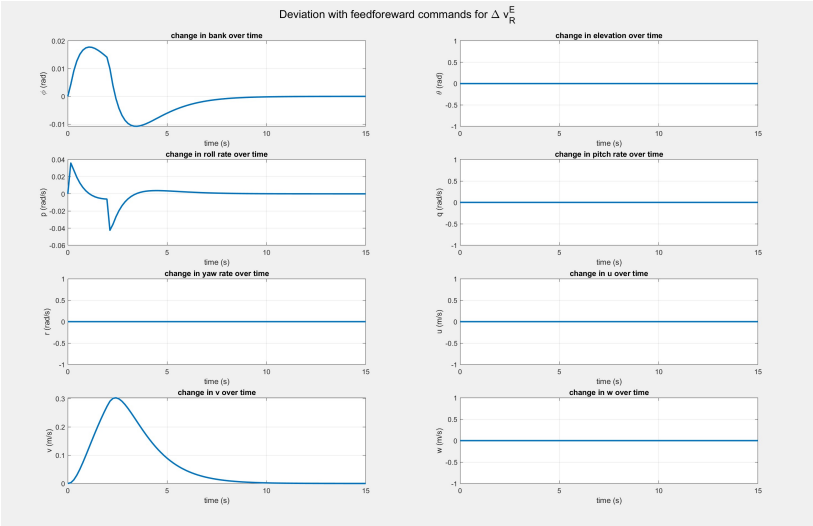


Fig. 10 Deviation laterally

From figure 10 the lack of maneuverability of the quadcopter can clearly be seen by the change in $\hat{y}$ direction velocity plot. The low magnitude K_3 value keeps the velocity value from even reaching the specified Δv_R^E of 0.5 meters per second. For a higher magnitude K_3 value, the shape of this plot would be more square/sharp, and would actually reach the specified Δv_R^E due to the higher magnitude $\Delta \dot{p}$ values resulting from the increase in K_3 magnitude.

Now, analyzing the longitudinal rotational dynamics simulation,

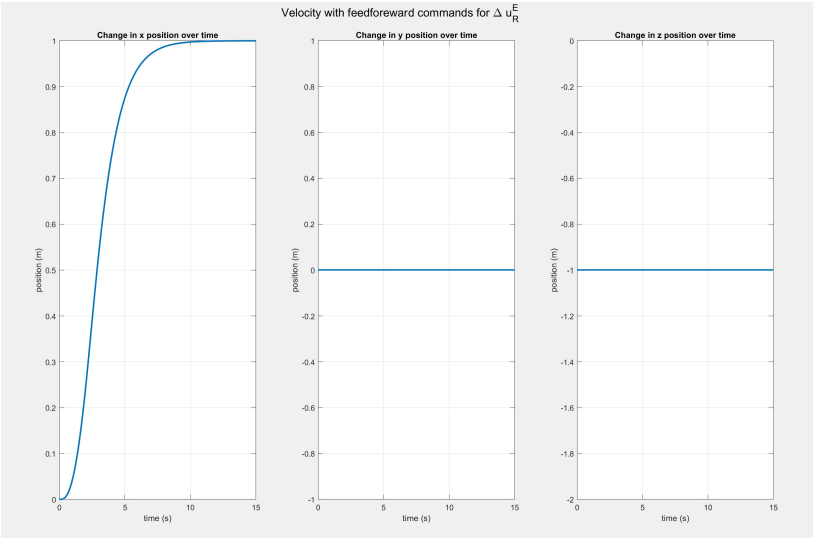


Fig. 11 Longitudinal Change in position over time

As seen in figure 11, the position in the $\hat{x}$ increases to 1 meter displacement in a time period more than two seconds; the main goal of this simulation. This lack of response for the change in position is due to the time constant constraints of this lab, which resulted in a substantially small 3rd gain value for the longitudinal simulation. The small K_3 values are what caused the slow rise to one meter $\hat{x}$ direction. As seen in the equation for the pitch rate derivative derived in question 1:

$$\Delta\dot{q} = -\frac{K_1\Delta p}{I_y} - \frac{K_2\Delta\theta}{I_y} - \frac{K_3K_2(\Delta u_R^E - \Delta u^E)}{I_y} \tag{2}$$

For small magnitude K_3 values, the magnitude of $\Delta\dot{q}$ is smaller, therefore the pitching of the quadcopter becomes slower as K_3 gets smaller. This justifies why the simulation's response time is slower than expected.

A plot of the quadcopter's position was created to see the behavior of the quadcopter during the time of the applied Δu_R^E.

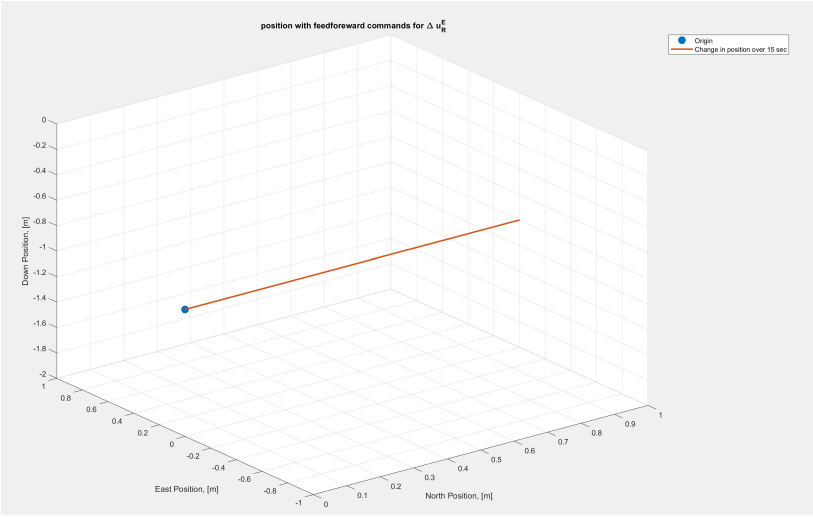


Fig. 12 Longitudinal Change in position over time

As seen from figure 12 the quadcopter translates steadily in the positive north direction. This behavior was expected, as the Δu_R^E applied to the quadcopter would cause a small initial spike in elevation in the negative direction, as in order

to have a positive Δu_R^E the elevation must be initially negative. This initial spike was corrected by the implimented K_2 values, and the elevation angle returned to zero radians quickly. Similar results are true of the pitch rate, as the initial velocity caused an initial spike in pitch rate in the negative direction, due to the negative initial elevation angle required to have a positive Δu_R^E. The K_2 gain value quickly brought the pitch rate back to zero radians per second after this spike. These results are also clearly seen in the following plot.

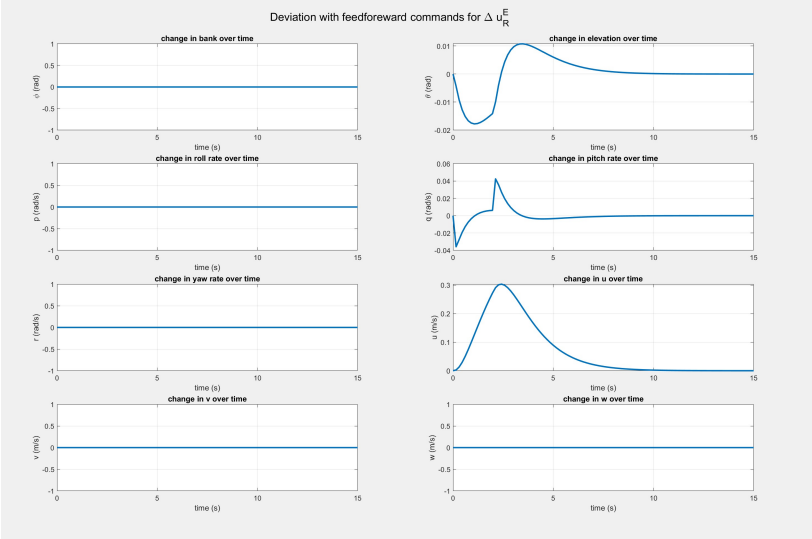


Fig. 13 Deviation longitudinally

From figure 13 the lack of maneuverability of the quadcopter can clearly be seen by the change in $\hat{x}$ direction velocity plot. The low magnitude K_3 value keeps the velocity value from even reaching the specified Δu_R^E of 0.5 meters per second. For a higher magnitude K_3 value, the shape of this plot would be more square shaped, and would actually reach the specified Δu_R^E due to the higher magnitude $\Delta \dot{p}$ values resulting from the increase in K_3 magnitude.

IV. Question 3

Implimenting the gain values developed for the first part of this assignment to the Mambo quad copter, experimental data was collected and compared to the control feedback system results. The following picture shows the GUI input to the experimental quadcopter.

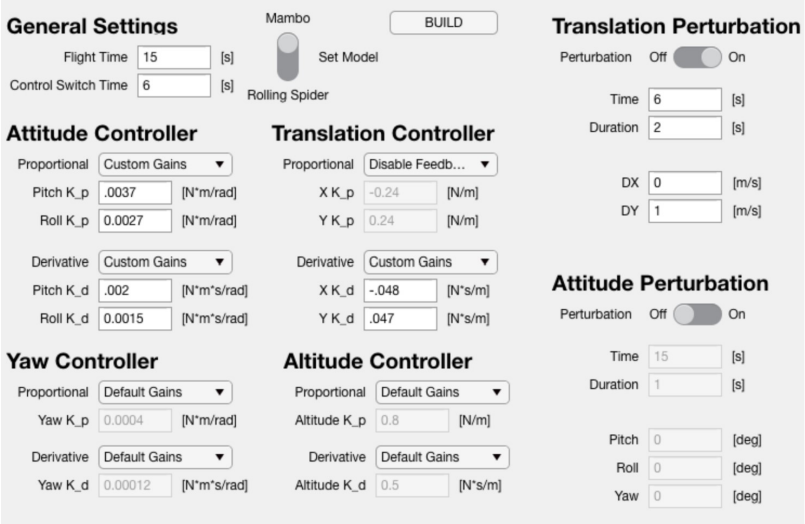


Fig. 14 GUI for experimental data

As seen from this picture, the gain values used were the same as derrived in Question 1 of this assignment ($k_1 = 0.0015, k_2 = 0.0027, k_3 = 0.047, k_4 = 0.0020, k_5 = 0.0037$, and $k_6 = 0.048$). The deviation used for this test was a translation perturbation of 1 m/s occurring at the six second mark of the flight, for two seconds. Using a modified ODE function that has both Δu_R^E and Δv_R^E feedforeward commands, simulations were run to compare to the experimental data

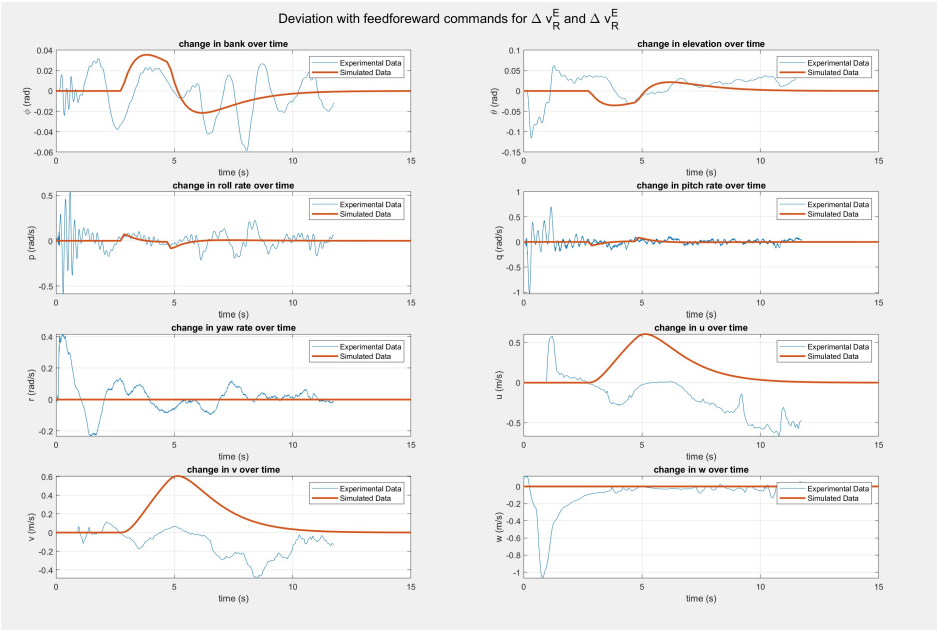


Fig. 15 Experimental data, and Simulated data

As seen from figure 16, The simulated data matches up relatively well with the experimental data, given the noise encountered by the quadcopter due to disturbances while collecting data. The main differences seen between the experimental data and the simulated data are seen in the $\hat{x}$ and $\hat{y}$ direction velocities. Due to the altitude control implemented in the experimental data, the experimental u and v values over time have lower overall magnitudes when compared to the simulated data, as these velocity values are constrained in order to keep the quadcopter from changing altitude during flight. When looking at the shape of these plots, the shapes do make overall sense. As the bank angle increases positively to allow for the positive Δv_R^E feedforeward command, gain values are activated to bring the

quadcopter's bank angle back to zero radians. The roll rate also has a slight positive increase as the bank angle increases positively to allow for the positive Δv_R^E feedforeward command. This change in roll rate is also corrected by the gains, and brought back to zero radians per second in less than 6 seconds. As the elevation angle increases negatively to allow for the positive Δu_R^E feedforeward command, gain values are activated to bring the quadcopter's elevation angle back to zero radians. The pitch rate also has a slight negative increase as the bank angle increases positively to allow for the positive Δu_R^E feedforeward command. This change in pitch rate is also corrected by the gains, and brought back to zero radians per second in less than 6 seconds. Both the change in the position in the $\hat{z}$ direction and the change in yaw rate over time remain zero, as expected.

Along with the deviation comparisons, plots were also generated for the displacement vs. time for both the experimental data and the simulated data.

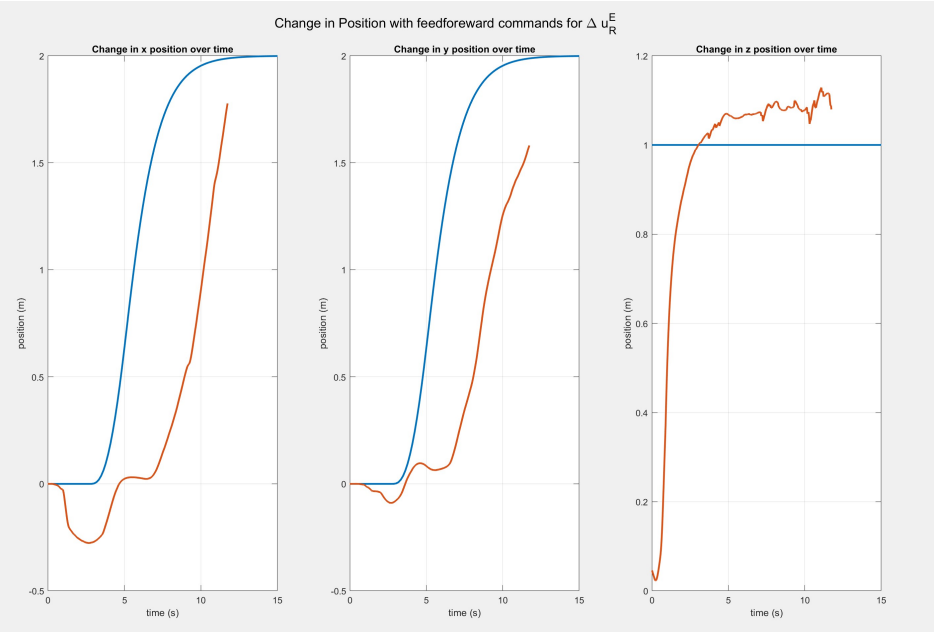


Fig. 16 Experimental data, and Simulated data

As seen from this plot, the values for the experimental data vs. the simulated data are similar, but there are some key differences. The experimental data has an overall longer time to rise in both the $\hat{x}$ and $\hat{y}$ directions. This is due to both the altitude control implemented into the experimental quadcopter, coupled with the air perturbances not modelled in the simulation.

V. MATLAB Code

Question 1 Script

```
clear all
close all
%% lateral
Lambda1 = -2; % 1st lambda value, found from lab calculations
Lambda2 = -20;% 2nd lambda value is 10x Lambda 1, in order to make lambda 1 dominant
Ix = 6.8e-5;   % moment of inertia in the x direction [kg m^2]
Iy = 9.2e-5; % Moment of inertia abt y (kgm^2)
g = 9.81
syms K1 K2 % symbollically solving for K1 and K2
eqn1 = Lambda1^2 +Lambda1*(K1/Ix) + (K2/Ix) == 0; % 1st eigenvalue solution
eqn2 = Lambda2^2 +Lambda2*(K1/Ix) + (K2/Ix) == 0; % 2nd eigenvalue solution


[A,B] = equationsToMatrix([eqn1, eqn2], [K1, K2]); % convert both the equations and the unknowns
    into independent vectors

Solution = linsolve(A,B); % solve the system of equations
Solution = double(Solution); % convert symbolic solution vector to vector of doubles
K1 = Solution(1); % bank control
K2 = Solution(2); % bank control

i = 1;
Lambda3 =zeros(3, 491);
for K3 = (.001:.0001:.08); % loop for differing K3 values
 A= [ -K1/Ix, -K2/Ix, -K2*K3/Ix; 1, 0, 0; 0, g, 0]; % matrix derrived in notes from functional
     block diagram
 Lambda3(:, i) = eig(A); % take the eigenvector of the matrix to get the eigenvalues
 i = i+1; % itterator
end

figure()
  K3 = (0.001:0.0001:.08); % reinitialize K3 vector
plot(real(Lambda3),imag(Lambda3), 'x'); % plot Eigenvalues 3 vs. K3
 xlabel('Real Axis (1/sec)');
 ylabel('Imaginary Axis (1/sec)');
 title('Locus of Eigenvalues for Liearized Lateral Rotation Dynamics');
index1 = find(Lambda3(3,:) > -0.8, 1, 'last');
k3 = K3(index1+1);
%% longitudional
syms K4 K5 % symbollically solving for K3 and K4
eqn3 = Lambda1^2 +Lambda1*(K4/Iy) + (K5/Iy) == 0; % 1st eigenvalue solution
eqn4 = Lambda2^2 +Lambda2*(K4/Iy) + (K5/Iy) == 0; % 1st eigenvalue solution


[C,D] = equationsToMatrix([eqn3, eqn4], [K4, K5]);

ForceVect2 = linsolve(C,D);
ForceVect2 = double(ForceVect2);
K4 = ForceVect2(1); % elevation control
K5 = ForceVect2(2); % elevation control



i = 1;
for K6 = -(.001:.0001:.08) % loop for differing K3 values
 B= [ -K4/Iy, -K5/Iy, -(K5*K6)/Iy; 1, 0, 0; 0, -g, 0]; % matrix derrived in notes from functional
     block diagram
 Lambda6(:, i) = eig(B); % take the eigenvector of the matrix to get the eigenvalues
 i = i+1; % itterator
end



  K6 = -(.001:.0001:.08); % reinitialize K3 vector
```

```
%plot(K6,real(Lambda6(3,:))); % plot Eigenvalues 3 vs. K3
figure()
plot(real(Lambda6),imag(Lambda6), 'x'); % plot Eigenvalues 3 vs. K3
 xlabel('Real Axis (1/sec)');
 ylabel('Imaginary Axis (1/sec)');
 title('Locus of Eigenvalues for Liearized Longitudional Rotation Dynamics');

 index2 = find(Lambda6(3,:) > -0.8, 1, 'last');
k6 = K6(index2+1);
%%
```

Question 2 Script

```
% This script initializes initial conditions for the trajectory of a drone
% in steady flight conditions (hovering) then calls the ODE45 function to
% numerically integrate the position of the drone with respect to the
% initial conditions, along with plotting the results answering Question
% 7
%
%   Author: Benjiman Smith
%   Collaborators: E. Owen, I. Quezada
%   Date: 1/25/2020
clear all
close all
%% lateral
Lambda1 = -2; % 1st lambda value, found from lab calculations
Lambda2 = -20;% 2nd lambda value is 10x Lambda 1, in order to make lambda 1 dominant
Ix = 6.8e-5;   % moment of inertia in the x direction [kg m^2]
Iy = 9.2e-5; % Moment of inertia abt y (kgm^2)
g = 9.81;
syms K1 K2 % symbollically solving for K1 and K2
eqn1 = Lambda1^2 +Lambda1*(K1/Ix) + (K2/Ix) == 0; % 1st eigenvalue solution
eqn2 = Lambda2^2 +Lambda2*(K1/Ix) + (K2/Ix) == 0; % 2nd eigenvalue solution


[A,B] = equationsToMatrix([eqn1, eqn2], [K1, K2]); % convert both the equations and the unknowns
    into independent vectors

Solution = linsolve(A,B); % solve the system of equations
Solution = double(Solution); % convert symbolic solution vector to vector of doubles
K1 = Solution(1); % bank control
K2 = Solution(2); % bank control

i = 1;
Lambda3 =zeros(3, 491);
for K3 = (.001:.0001:.08) % loop for differing K3 values
 A= [ -K1/Ix, -K2/Ix, -K2*K3/Ix; 1, 0, 0; 0, g, 0]; % matrix derrived in notes from functional
     block diagram
 Lambda3(:, i) = eig(A); % take the eigenvector of the matrix to get the eigenvalues
 i = i+1; % itterator
end

%figure(6)
hold on
  K3 = (0.001:0.0001:.08); % reinitialize K3 vector
%plot(K3,real(Lambda3(3,:))); % plot Eigenvalues 3 vs. K3

 index1 = find(Lambda3(3,:) > -0.8, 1, 'last');
k3 = K3(index1+1);
%% longitudional
syms K4 K5 % symbollically solving for K3 and K4
eqn3 = Lambda1^2 +Lambda1*(K4/Iy) + (K5/Iy) == 0; % 1st eigenvalue solution
eqn4 = Lambda2^2 +Lambda2*(K4/Iy) + (K5/Iy) == 0; % 1st eigenvalue solution


[C,D] = equationsToMatrix([eqn3, eqn4], [K4, K5]);

```

```
ForceVect2 = linsolve(C,D);
ForceVect2 = double(ForceVect2);
K4 = ForceVect2(1); % elevation control
K5 = ForceVect2(2); % elevation control



i = 1;
for K6 = -(.001:.0001:.08) % loop for differing K3 values
 B= [ -K4/Iy, -K5/Iy, -(K5*K6)/Iy; 1, 0, 0; 0, -g, 0]; % matrix derrived in notes from functional
      block diagram
 Lambda6(:, i) = eig(B); % take the eigenvector of the matrix to get the eigenvalues
 i = i+1; % itterator
end



  K6 = -(.001:.0001:.08); % reinitialize K3 vector
%plot(K6,real(Lambda6(3,:))); % plot Eigenvalues 3 vs. K3


 index2 = find(Lambda6(3,:) > -0.8, 1, 'last');
k6 = K6(index2+1);
%% feedforeward commands for \Delta v^{E}_{R}

m = 0.068; % mass of the drone [kg]
r = 0.06;  % body to motor distance [m]
k = 0.0024;  % [Nm/N]
rad = r/sqrt(2);  % [m]
g = 9.81; % gravity [m/s^2]
alpha = 2e-6;   % [N/(m/s)^2]
eta = 1e-3;    % [N/(rad/s)^2]
Ix = 6.8e-5;   % moment of inertia in the x direction [kg m^2]
Iy = 9.2e-5;   % moment of inertia in the y direction [kg m^2]
Iz = 1.35e-4;  % moment of inertia in the z direction [kg m^2]
givens = [alpha eta Ix Iy Iz m r k rad g K1 K2 k3 K4 K5 k6]; % givens vector
F = m*g;
tspan = linspace(0,15); % time vector
Pertubations = zeros(1, 3); % No perturbation
TrimForces = ones(1, 4) * m * g / 4; % forces required by each motor to maintain hover
conditions = zeros(1, 12); % initialize conditions vector (very large);
conditions(12) = -1; % set down direction to 1 to make signs correct for plotting
t1 = 0;
t2 = 0;
X = 0;
X2 = 0;
options = odeset('Events', @StopFnct, 'RelTol', 1e-8); % stop function that ends ODE when a
    tolerance of 1e-8 is met
[t1, X] = ode45(@(t, F)Specs2LB5LCV(t, F, TrimForces, Pertubations, givens), tspan, conditions,
    options);
figure()
sgtitle('Change in position with feedforeward commands for \Delta v^{E}_{R}');
subplot(1,3,1);
plot(t1, X(:,10),'linewidth', 2);
grid on
xlabel('time (s)')
ylabel('position (m)')
title('Change in x position over time')
subplot(1,3,2);
plot(t1, X(:,11),'linewidth', 2);
grid on
xlabel('time (s)')
ylabel('position (m)')
title('Change in y position over time')
subplot(1,3,3);
plot(t1, X(:,12),'linewidth', 2);
grid on
xlabel('time (s)')
```

```
ylabel('position (m)')
title('Change in z position over time')

figure()
plot3(0, 0, -1, '.', 'MarkerSize',35);
hold on
plot3(X(:, 10), X(:, 11), X(:, 12), 'lineWidth',2)
grid on
zlabel('Down Position, [m]')
xlabel('North Position, [m]')
ylabel('East Position, [m]')
title(' position with feedforeward commands for \Delta v^{E}_{R}');
legend('Origin', 'Change in position over 15 sec');
hold off




figure()
sgtitle('Deviation with feedforeward commands for \Delta v^{E}_{R}');
 subplot(4,2,1);
plot(t1, X(:,7),'linewidth', 2);

grid on
xlabel('time (s)')
ylabel('\phi (rad)')
title('change in bank over time')

hold off
%
 subplot(4,2,2);
plot(t1, X(:,8),'linewidth', 2);

grid on
xlabel('time (s)')
ylabel('\theta (rad)')
title('change in elevation over time')
hold off
%
 subplot(4,2,3);
plot(t1, X(:,4),'linewidth', 2);

grid on
xlabel('time (s)')
ylabel('p (rad/s)')
title('change in roll rate over time')

hold off
%
 subplot(4,2,4);
plot(t1, X(:,5),'linewidth', 2);

grid on
xlabel('time (s)')
ylabel('q (rad/s)')
title('change in pitch rate over time')

hold off
%
 subplot(4,2,5);
plot(t1, X(:,6),'linewidth', 2);

grid on
xlabel('time (s)')
ylabel('r (rad/s)')
title('change in yaw rate over time')

hold off
```

```
%
 subplot(4,2,6);
plot(t1, X(:,1),'linewidth', 2);

grid on
xlabel('time (s)')
ylabel('u (m/s)')
title('change in u over time')

hold off
%
 subplot(4,2,7);
plot(t1, X(:,2),'linewidth', 2);

grid on
xlabel('time (s)')
ylabel('v (m/s)')
title('change in v over time')

hold off
%
subplot(4,2,8);
plot(t1, X(:,3),'linewidth', 2);
grid on
xlabel('time (s)')
ylabel('w (m/s)')
title('change in w over time')
hold off


%% feedforeward commands for \Delta u^{E}_{R}
m = 0.068; % mass of the drone [kg]
r = 0.06;  % body to motor distance [m]
k = 0.0024;  % [Nm/N]
rad = r/sqrt(2);  % [m]
g = 9.81; % gravity [m/s^2]
alpha = 2e-6;   % [N/(m/s)^2]
eta = 1e-3;    % [N/(rad/s)^2]
Ix = 6.8e-5;   % moment of inertia in the x direction [kg m^2]
Iy = 9.2e-5;   % moment of inertia in the y direction [kg m^2]
Iz = 1.35e-4;  % moment of inertia in the z direction [kg m^2]
givens = [alpha eta Ix Iy Iz m r k rad g K1 K2 k3 K4 K5 k6]; % givens vector
F = m*g;
tspan = linspace(0,15); % time vector
Pertubations = zeros(1, 3); % No perturbation
TrimForces = ones(1, 4) * m * g / 4; % forces required by each motor to maintain hover
conditions = zeros(1, 12); % initialize conditions vector (very large);
conditions(12) = -1; % set down direction to 1 to make signs correct for plotting
t1 = 0;
t2 = 0;
X = 0;
X2 = 0;
options = odeset('Events', @StopFnct, 'RelTol', 1e-8); % stop function that ends ODE when a
    tolerance of 1e-8 is met
[t1, X] = ode45(@(t, F)Specs2LB5LCU(t, F, TrimForces, Pertubations, givens), tspan, conditions,
    options);
figure()
sgtitle('Velocity with feedforeward commands for \Delta u^{E}_{R}');
subplot(1,3,1);
plot(t1, X(:,10),'linewidth', 2);
grid on
xlabel('time (s)')
ylabel('position (m)')
title('Change in x position over time')
subplot(1,3,2);
plot(t1, X(:,11),'linewidth', 2);
grid on
xlabel('time (s)')
```

```
ylabel('position (m)')
title('Change in y position over time')
subplot(1,3,3);
plot(t1, X(:,12),'linewidth', 2);
grid on
xlabel('time (s)')
ylabel('position (m)')
title('Change in z position over time')

figure()
plot3(0, 0, -1, '.', 'MarkerSize',35);
hold on
plot3(X(:, 10), X(:, 11), X(:, 12), 'lineWidth',2)
grid on
zlabel('Down Position, [m]')
xlabel('North Position, [m]')
ylabel('East Position, [m]')
title(' position with feedforeward commands for \Delta u^{E}_{R}');
legend('Origin', 'Change in position over 15 sec');
hold off
figure()


sgtitle('Deviation with feedforeward commands for \Delta u^{E}_{R}');
 subplot(4,2,1);
plot(t1, X(:,7),'linewidth', 2);

grid on
xlabel('time (s)')
ylabel('\phi (rad)')
title('change in bank over time')

hold off
%
 subplot(4,2,2);
plot(t1, X(:,8),'linewidth', 2);

grid on
xlabel('time (s)')
ylabel('\theta (rad)')
title('change in elevation over time')
hold off
%
 subplot(4,2,3);
plot(t1, X(:,4),'linewidth', 2);

grid on
xlabel('time (s)')
ylabel('p (rad/s)')
title('change in roll rate over time')

hold off
%
 subplot(4,2,4);
plot(t1, X(:,5),'linewidth', 2);

grid on
xlabel('time (s)')
ylabel('q (rad/s)')
title('change in pitch rate over time')

hold off
%
 subplot(4,2,5);
plot(t1, X(:,6),'linewidth', 2);

grid on
xlabel('time (s)')
```

```
ylabel('r (rad/s)')
title('change in yaw rate over time')

hold off
%
 subplot(4,2,6);
plot(t1, X(:,1),'linewidth', 2);

grid on
xlabel('time (s)')
ylabel('u (m/s)')
title('change in u over time')

hold off
%
 subplot(4,2,7);
plot(t1, X(:,2),'linewidth', 2);

grid on
xlabel('time (s)')
ylabel('v (m/s)')
title('change in v over time')

hold off
%
subplot(4,2,8);
plot(t1, X(:,3),'linewidth', 2);
grid on
xlabel('time (s)')
ylabel('w (m/s)')
title('change in w over time')
hold off
```

Question 3 Script

```
% This script answers question 4 by plotting the experimental data from
% 11:37, 2/14/2020, and plotting the simulated linearized feedback control
% simulation on top for comparison
%   Author: Benjiman Smith
%   Collaborators: E. Owen, I. Quezada
%   Date: 2/20/2020
%

clear all
close all
clc
Lambda1 = -2; % 1st lambda value, found from lab calculations
Lambda2 = -20;% 2nd lambda value is 10x Lambda 1, in order to make lambda 1 dominant
m = 0.068; % mass of the drone [kg]
r = 0.06;  % body to motor distance [m]
k = 0.0024;  % [Nm/N]
rad = r/sqrt(2);  % [m]
g = 9.81; % gravity [m/s^2]
alpha = 2e-6;   % [N/(m/s)^2]
eta = 1e-3;    % [N/(rad/s)^2]
Ix = 6.8e-5;   % moment of inertia in the x direction [kg m^2]
Iy = 9.2e-5;   % moment of inertia in the y direction [kg m^2]
Iz = 1.35e-4;  % moment of inertia in the z direction [kg m^2]
syms K1 K2 % symbollically solving for K1 and K2
eqn1 = Lambda1^2 +Lambda1*(K1/Ix) + (K2/Ix) == 0; % 1st eigenvalue solution
eqn2 = Lambda2^2 +Lambda2*(K1/Ix) + (K2/Ix) == 0; % 2nd eigenvalue solution


[A,B] = equationsToMatrix([eqn1, eqn2], [K1, K2]); % convert both the equations and the unknowns
    into independent vectors

Solution = linsolve(A,B); % solve the system of equations
```

```
Solution = double(Solution); % convert symbolic solution vector to vector of doubles
K1 = Solution(1); % bank control
K2 = Solution(2); % bank control

i = 1;
Lambda3 =zeros(3, 491);
for K3 = (.001:.0001:.08) % loop for differing K3 values
 A= [ -K1/Ix, -K2/Ix, -K2*K3/Ix; 1, 0, 0; 0, g, 0]; % matrix derrived in notes from functional
     block diagram
 Lambda3(:, i) = eig(A); % take the eigenvector of the matrix to get the eigenvalues
 i = i+1; % itterator
end

%figure(6)
hold on
  K3 = (0.001:0.0001:.08); % reinitialize K3 vector
%plot(K3,real(Lambda3(3,:))); % plot Eigenvalues 3 vs. K3

 index1 = find(Lambda3(3,:) > -0.8, 1, 'last');
k3 = K3(index1+1);
%% longitudional
syms K4 K5 % symbollically solving for K3 and K4
eqn3 = Lambda1^2 +Lambda1*(K4/Iy) + (K5/Iy) == 0; % 1st eigenvalue solution
eqn4 = Lambda2^2 +Lambda2*(K4/Iy) + (K5/Iy) == 0; % 1st eigenvalue solution


[C,D] = equationsToMatrix([eqn3, eqn4], [K4, K5]);

ForceVect2 = linsolve(C,D);
ForceVect2 = double(ForceVect2);
K4 = ForceVect2(1); % elevation control
K5 = ForceVect2(2); % elevation control



i = 1;
for K6 = -(.001:.0001:.08) % loop for differing K3 values
 B= [ -K4/Iy, -K5/Iy, -(K5*K6)/Iy; 1, 0, 0; 0, -g, 0]; % matrix derrived in notes from functional
     block diagram
 Lambda6(:, i) = eig(B); % take the eigenvector of the matrix to get the eigenvalues
 i = i+1; % itterator
end



  K6 = -(.001:.0001:.08); % reinitialize K3 vector
%plot(K6,real(Lambda6(3,:))); % plot Eigenvalues 3 vs. K3


 index2 = find(Lambda6(3,:) > -0.8, 1, 'last');
k6 = K6(index2+1);

givens = [alpha eta Ix Iy Iz m r k rad g K1 K2 k3 K4 K5 k6]; % givens vector
F = m*g;


load('RSdata_1023.mat'); % load in data
% loading in of parameters
xE = rt_estim.signals.values(:,1);
yE = rt_estim.signals.values(:,2);
zE = rt_estim.signals.values(:,3);
theta = rt_estim.signals.values(:,5);
phi = rt_estim.signals.values(:,6);
u = rt_estim.signals.values(:,7);
v = rt_estim.signals.values(:,8);
w = rt_estim.signals.values(:,9);
p = rt_estim.signals.values(:,10);
q = rt_estim.signals.values(:,11);
```

```
r = rt_estim.signals.values(:,12);
times =rt_estim.time(:);

startpoint = max(theta); % find maximum theta angle, beginning of disturbance
timestart = find(theta==startpoint); % find the index of this maximum value
subtracttime = times(timestart); % find the time at which the maximum occurs
subtracttime = 4;

tspan = linspace(0,15); % time vector
Pertubations = zeros(1, 3); % No perturbation
TrimForces = ones(1, 4) * m * g / 4; % forces required by each motor to maintain hover
conditions = zeros(1, 12); % initialize conditions vector (very large)
conditions(12) = 1; % set down direction to 1 to make signs correct for plotting
t1 = 0; % initialize time
X = 0;% initialize x

options = odeset('Events', @StopFnct, 'RelTol', 1e-8); % stop function that ends ODE when a
    tolerance of 1e-8 is met
[t1, X] = ode45(@(t, F)Specs2LB5LC(t, F, TrimForces, Pertubations, givens), tspan, conditions,
    options); % linear controlled ode call

%% plotting
% sgtitle('Deviation with feedforeward commands for \Delta v^{E}_{R} and \Delta u^{E}_{R}'); %
    subplot title
%  subplot(4,2,1);
% plot(times, phi); % plot experimental data
%
% grid on
% xlabel('time (s)')
% ylabel('\phi (rad)')
% title('change in bank over time')
%  legend('Experimental Data');
% hold off
%  subplot(4,2,2);
% plot(times, theta);
%
% xlabel('time (s)')
% ylabel('\theta (rad)')
% title('change in elevation over time')
% legend('Experimental Data');
% hold off
%  subplot(4,2,3);
% plot(times, p);
%
% grid on
% xlabel('time (s)')
% ylabel('p (rad/s)')
% title('change in roll rate over time')
%  legend('Experimental Data');
% hold off
%  subplot(4,2,4);
% plot(times, q);
%
% grid on
% xlabel('time (s)')
% ylabel('q (rad/s)')
% title('change in pitch rate over time')
%  legend('Experimental Data');
% hold off
%  subplot(4,2,5);
% plot(times, r);
%
% grid on
% xlabel('time (s)')
% ylabel('r (rad/s)')
% title('change in yaw rate over time')
%  legend('Experimental Data');
% hold off
```

```
%  subplot(4,2,6);
% plot(times, u);
%
% grid on
% xlabel('time (s)')
% ylabel('u (m/s)')
% title('change in u over time')
%  legend('Experimental Data');
% hold off
%  subplot(4,2,7);
% plot(times, v);
%
% grid on
% xlabel('time (s)')
% ylabel('v (m/s)')
% title('change in v over time')
%  legend('Experimental Data');
% hold off
%  subplot(4,2,8);
% plot(times, w);
%
% grid on
% xlabel('time (s)')
% ylabel('w (m/s)')
% title('change in w over time')
%
%  legend('Experimental Data');
% hold off
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% figure()
% sgtitle('Deviation with feedforeward commands for \Delta v^{E}_{R}');
%  subplot(4,2,1);
% plot(t1, X(:,7),'linewidth', 2);
%
% grid on
% xlabel('time (s)')
% ylabel('\phi (rad)')
% title('change in bank over time')
%
% hold off
% %
%  subplot(4,2,2);
% plot(t1, X(:,8),'linewidth', 2);
%
% grid on
% xlabel('time (s)')
% ylabel('\theta (rad)')
% title('change in elevation over time')
% hold off
% %
%  subplot(4,2,3);
% plot(t1, X(:,4),'linewidth', 2);
%
% grid on
% xlabel('time (s)')
% ylabel('p (rad/s)')
% title('change in roll rate over time')
%
% hold off
% %
%  subplot(4,2,4);
% plot(t1, X(:,5),'linewidth', 2);
%
% grid on
% xlabel('time (s)')
% ylabel('q (rad/s)')
```

```
% title('change in pitch rate over time')
%
% hold off
% %
%  subplot(4,2,5);
% plot(t1, X(:,6),'linewidth', 2);
%
% grid on
% xlabel('time (s)')
% ylabel('r (rad/s)')
% title('change in yaw rate over time')
%
% hold off
% %
%  subplot(4,2,6);
% plot(t1, X(:,1),'linewidth', 2);
%
% grid on
% xlabel('time (s)')
% ylabel('u (m/s)')
% title('change in u over time')
%
% hold off
% %
%  subplot(4,2,7);
% plot(t1, X(:,2),'linewidth', 2);
%
% grid on
% xlabel('time (s)')
% ylabel('v (m/s)')
% title('change in v over time')
%
% hold off
% %
% subplot(4,2,8);
% plot(t1, X(:,3),'linewidth', 2);
% grid on
% xlabel('time (s)')
% ylabel('w (m/s)')
% title('change in w over time')
% hold off

% plotting
sgtitle('Deviation with feedforeward commands for \Delta v^{E}_{R} and \Delta v^{E}_{R}'); %
    subplot title
 subplot(4,2,1);
plot(times, phi); % plot experimental data
hold on
plot(t1 , X(:,7), 'linewidth', 2) % plot simulator data
grid on
xlabel('time (s)')
ylabel('\phi (rad)')
title('change in bank over time')
 legend('Experimental Data', 'Simulated Data');
hold off
 subplot(4,2,2);
plot(times, theta);
hold on
plot(t1 , X(:,8),'linewidth', 2);
grid on
xlabel('time (s)')
ylabel('\theta (rad)')
title('change in elevation over time')
legend('Experimental Data', 'Simulated Data');
hold off
 subplot(4,2,3);
plot(times, p);
hold on
```

```
plot(t1 , X(:,4),'linewidth', 2);
grid on
xlabel('time (s)')
ylabel('p (rad/s)')
title('change in roll rate over time')
 legend('Experimental Data', 'Simulated Data');
hold off
 subplot(4,2,4);
plot(times, q);
hold on
plot(t1 , X(:,5),'linewidth', 2);
grid on
xlabel('time (s)')
ylabel('q (rad/s)')
title('change in pitch rate over time')
 legend('Experimental Data', 'Simulated Data');
hold off
 subplot(4,2,5);
plot(times, r);
hold on
plot(t1 , X(:,6),'linewidth', 2);
grid on
xlabel('time (s)')
ylabel('r (rad/s)')
title('change in yaw rate over time')
 legend('Experimental Data', 'Simulated Data');
hold off
 subplot(4,2,6);
plot(times, u);
hold on
plot(t1 , X(:,1),'linewidth', 2);
grid on
xlabel('time (s)')
ylabel('u (m/s)')
title('change in u over time')
 legend('Experimental Data', 'Simulated Data');
hold off
 subplot(4,2,7);
plot(times, v);
hold on
plot(t1 , X(:,2),'linewidth', 2);
grid on
xlabel('time (s)')
ylabel('v (m/s)')
title('change in v over time')
 legend('Experimental Data', 'Simulated Data');
hold off
 subplot(4,2,8);
plot(times, w);
hold on
plot(t1 , X(:,3),'linewidth', 2);
grid on
xlabel('time (s)')
ylabel('w (m/s)')
title('change in w over time')

 legend('Experimental Data', 'Simulated Data');
hold off
figure()
sgtitle('Change in Position with feedforeward commands for \Delta u^{E}_{R}');
subplot(1,3,1);
plot(t1, X(:,10),'linewidth', 2);
hold on
plot(times, -xE,'linewidth', 2);
grid on
xlabel('time (s)')
ylabel('position (m)')
title('Change in x position over time')
```

```
subplot(1,3,2);
plot(t1, X(:,11),'linewidth', 2);
grid on
hold on
plot(times, -yE,'linewidth', 2);
xlabel('time (s)')
ylabel('position (m)')
title('Change in y position over time')
subplot(1,3,3);
plot(t1, X(:,12),'linewidth', 2);
grid on
hold on
plot(times, -zE,'linewidth', 2);
xlabel('time (s)')
ylabel('position (m)')
title('Change in z position over time')
```

linearized feedback control ODE function with Δu_R^E and Δv_R^E varying

```
% linear ODE Function
%Benjiman Smith
%02/13/2020
function dydt = Specs2LB5LC(t, Conditions, Force, Disturb, givens)
    % define givens
    m = givens(6);      % mass of the drone [kg]
    r = givens(7);       % body to motor distance [m]
    k = givens(8);     % [Nm/N]
    R = givens(9);  % [m]
    g = givens(10);% gravity [m/s^2]
    alpha = givens(1);   % [N/(m/s)^2]
    eta = givens(2);    % [N/(rad/s)^2]
    I_x = givens(3);   % moment of inertia in the x direction [kg m^2]
    I_y = givens(4);   % moment of inertia in the y direction[kg m^2]
    I_z = givens(5);  % moment of inertia in the z direction[kg m^2]
    k_1 = givens(11);
    k_2 = givens(12);
    k_3 = givens(13);
    k_4 = givens(14);
    k_5 = givens(15);
    k_6 = givens(16);

    % define conditions vector
    deltau = Conditions(1); % inertial velocity in the u direction in body coordinates [m/s]
    deltav = Conditions(2); % inertial velocity in the v direction in body coordinates [m/s]
    deltaw = Conditions(3); % inertial velocity in the w direction in body coordinates [m/s]
    deltap = Conditions(4); % inertial angular velocity in the p direction in body coordinates [
        rad/s]
    deltaq = Conditions(5); % inertial angular velocity in the q direction in body coordinates [
        rad/s]
    deltar = Conditions(6); % inertial angular velocity in the r direction in body coordinates [
        rad/s]
   deltaphi = Conditions(7); % bank [rad]
    deltatheta = Conditions(8); % elevation [rad]
    psi = Conditions(9); % azimuth [rad]
    x_E = Conditions(10); % position vector in the x direction in inertial coordinates [m]
    y_E = Conditions(11); % position vector in the y direction in inertial coordinates [m]
    z_E = Conditions(12); % position vector in the z direction in inertial coordinates [m]
    f1 = Force(1); % trim force exerted by motor 1 [N]
    f2 = Force(2); % trim force exerted by motor 1 [N]
    f3 = Force(3); % trim force exerted by motor 1 [N]
    f4 = Force(4); % trim force exerted by motor 1 [N]
     if (2.74<t) && (t<4.74)
         deltavr = 1;
         deltaur = 1;
     else
         deltavr = 0;
         deltaur=  0;
```

```
     end
    % Aerodynamic/Control Moments and Forces
    Laero = - alpha^2 * deltap^2 * sign(deltap); % p component of the aerodynamic moments
    Maero = - alpha^2 * deltaq^2 * sign(deltaq); % q component of the aerodynamic moments
    Naero = - alpha^2 * deltar^2 * sign(deltar); % w component of the aerodynamic moments
 Lcontrol = ((f1 + f2) - (f3 + f4)) * R; % p component of the control moments
    Mcontrol = ((f3 + f2) - (f1 + f4)) * R; % q component of the control moments
    Ncontrol = -0.004*deltar; % w component of the aerodynamic moments
    deltaL = Laero + Lcontrol; % L moment sum (abt x axis)
    deltaM = Maero + Mcontrol; % M moment sum (abt y axis)
    deltaN = Naero + Ncontrol; % N moment sum (abt Z axis)

    if t > 2 && t < 2.5
        deltaL = deltaL + Disturb(2); % Disturbed L moment (abt x axis)
        deltaM = deltaM + Disturb(1); % Disturbed M moment (abt y axis)
        deltaN = deltaN + Disturb(3); % Disturbed N moment (abt z axis)
    end
    Xaero = - eta^2 * deltau^2 * sign(deltau); % x component of aerodynamic force
    Yaero = - eta^2 * deltav^2 * sign(deltav); % y component of aerodynamic force
    Zaero = - eta^2 * deltaw^2 * sign(deltaw); % z component of aerodynamic force
    Xcontrol = 0; % x control
    Ycontrol = 0; % y control
    Zcontrol = -sum(Force); % gravitational force counteraction
    deltaX = Xaero + Xcontrol; % sum of x forces
    deltaY = Yaero + Ycontrol; % sum of y forces


    deltaZ = 0; % sum of z forces
    deltapdot = ((-k_1*deltap)/I_x) -((k_2*deltaphi)/I_x -((k_2*k_3)*(deltavr-deltav))/I_x  ); %
        roll rate derrivative
    deltaqdot = ((-k_4*deltaq)/I_y) -((k_5*deltatheta)/I_y -((k_5*k_6)*(deltaur-deltau))/I_y); %
        pitch rate derrivative


    deltardot = deltaN/I_z; % yaw rate derrivative

    dOmega_bdt = [deltapdot, deltaqdot, deltardot]';
    deltaudot = -g*deltatheta; % 4.7,1 acceleration in the x axis
    deltavdot = g*deltaphi; % acceleration in the y axis
    deltawdot = deltaZ/m; % acceleration in the z axis
    dVbdt = [deltaudot, deltavdot, deltawdot]';
    xdot =  deltau;
    ydot =  deltav;
    zdot = deltaw;
    dVEdt = [xdot, ydot, zdot]';
    phidot  = deltap; % bank roc (pg 104)
    thetadot = deltaq; % elevation roc (pg 104)
    psidot   = deltar; % azimuth roc (pg 104)
    dEuldt = [phidot, thetadot, psidot]';
    dydt = [dVbdt; dOmega_bdt; dEuldt; dVEdt];


end
```

linearized feedback control ODE function with Δu_R^E varying

```
% linear ODE Function
%Benjiman Smith
%02/13/2020
function dydt = Specs2LB5LCU(t, Conditions, Force, Disturb, givens)
    % define givens
    m = givens(6);      % mass of the drone [kg]
    r = givens(7);       % body to motor distance [m]
    k = givens(8);     % [Nm/N]
    R = givens(9);  % [m]
    g = givens(10);% gravity [m/s^2]
```

```
    alpha = givens(1);   % [N/(m/s)^2]
    eta = givens(2);    % [N/(rad/s)^2]
    I_x = givens(3);   % moment of inertia in the x direction [kg m^2]
    I_y = givens(4);   % moment of inertia in the y direction[kg m^2]
    I_z = givens(5);  % moment of inertia in the z direction[kg m^2]
    k_1 = givens(11);
    k_2 = givens(12);
    k_3 = givens(13);
    k_4 = givens(14);
    k_5 = givens(15);
    k_6 = givens(16);

    % define conditions vector
    deltau = Conditions(1); % inertial velocity in the u direction in body coordinates [m/s]
    deltav = Conditions(2); % inertial velocity in the v direction in body coordinates [m/s]
    deltaw = Conditions(3); % inertial velocity in the w direction in body coordinates [m/s]
    deltap = Conditions(4); % inertial angular velocity in the p direction in body coordinates [
        rad/s]
    deltaq = Conditions(5); % inertial angular velocity in the q direction in body coordinates [
        rad/s]
    deltar = Conditions(6); % inertial angular velocity in the r direction in body coordinates [
        rad/s]
   deltaphi = Conditions(7); % bank [rad]
    deltatheta = Conditions(8); % elevation [rad]
    psi = Conditions(9); % azimuth [rad]
    x_E = Conditions(10); % position vector in the x direction in inertial coordinates [m]
    y_E = Conditions(11); % position vector in the y direction in inertial coordinates [m]
    z_E = Conditions(12); % position vector in the z direction in inertial coordinates [m]
    f1 = Force(1); % trim force exerted by motor 1 [N]
    f2 = Force(2); % trim force exerted by motor 1 [N]
    f3 = Force(3); % trim force exerted by motor 1 [N]
    f4 = Force(4); % trim force exerted by motor 1 [N]
     if t <= 2
         deltavr = 0;
         deltaur = 0.5;
     else
         deltavr = 0;
         deltaur=  0;
     end
    % Aerodynamic/Control Moments and Forces
    Laero = - alpha^2 * deltap^2 * sign(deltap); % p component of the aerodynamic moments
    Maero = - alpha^2 * deltaq^2 * sign(deltaq); % q component of the aerodynamic moments
    Naero = - alpha^2 * deltar^2 * sign(deltar); % w component of the aerodynamic moments
Lcontrol = ((f1 + f2) - (f3 + f4)) * R; % p component of the control moments
    Mcontrol = ((f3 + f2) - (f1 + f4)) * R; % q component of the control moments
    Ncontrol = -0.004*deltar; % w component of the aerodynamic moments
    deltaL = Laero + Lcontrol; % L moment sum (abt x axis)
    deltaM = Maero + Mcontrol; % M moment sum (abt y axis)
    deltaN = Naero + Ncontrol; % N moment sum (abt Z axis)

    if t > 2 && t < 2.5
        deltaL = deltaL + Disturb(2); % Disturbed L moment (abt x axis)
        deltaM = deltaM + Disturb(1); % Disturbed M moment (abt y axis)
        deltaN = deltaN + Disturb(3); % Disturbed N moment (abt z axis)
    end
    Xaero = - eta^2 * deltau^2 * sign(deltau); % x component of aerodynamic force
    Yaero = - eta^2 * deltav^2 * sign(deltav); % y component of aerodynamic force
    Zaero = - eta^2 * deltaw^2 * sign(deltaw); % z component of aerodynamic force
    Xcontrol = 0; % x control
    Ycontrol = 0; % y control
    Zcontrol = -sum(Force); % gravitational force counteraction
    deltaX = Xaero + Xcontrol; % sum of x forces
    deltaY = Yaero + Ycontrol; % sum of y forces


    deltaZ = 0; % sum of z forces
    deltapdot = ((-k_1*deltap)/I_x) -((k_2*deltaphi)/I_x -((k_2*k_3)*(deltavr-deltav))/I_x  ); %
        roll rate derrivative
```

```
    deltaqdot = ((-k_4*deltaq)/I_y) -((k_5*deltatheta)/I_y -((k_5*k_6)*(deltaur-deltau))/I_y); % pitch rate derrivative


    deltardot = deltaN/I_z; % yaw rate derrivative

    dOmega_bdt = [deltapdot, deltaqdot, deltardot]';
    deltaudot = -g*deltatheta; % 4.7,1 acceleration in the x axis
    deltavdot = g*deltaphi; % acceleration in the y axis
    deltawdot = deltaZ/m; % acceleration in the z axis
    dVbdt = [deltaudot, deltavdot, deltawdot]';
    xdot =  deltau;
    ydot =  deltav;
    zdot = deltaw;
    dVEdt = [xdot, ydot, zdot]';
    phidot  = deltap; % bank roc (pg 104)
    thetadot = deltaq; % elevation roc (pg 104)
    psidot   = deltar; % azimuth roc (pg 104)
    dEuldt = [phidot, thetadot, psidot]';
    dydt = [dVbdt; dOmega_bdt; dEuldt; dVEdt];



end
```

linearized feedback control ODE function with Δv_R^E varying

```
% linear ODE Function
%Benjiman Smith
%02/13/2020
function dydt = Specs2LB5LCV(t, Conditions, Force, Disturb, givens)
    % define givens
    m = givens(6);      % mass of the drone [kg]
    r = givens(7);       % body to motor distance [m]
    k = givens(8);     % [Nm/N]
    R = givens(9);  % [m]
    g = givens(10);% gravity [m/s^2]
    alpha = givens(1);   % [N/(m/s)^2]
    eta = givens(2);    % [N/(rad/s)^2]
    I_x = givens(3);   % moment of inertia in the x direction [kg m^2]
    I_y = givens(4);   % moment of inertia in the y direction[kg m^2]
    I_z = givens(5);  % moment of inertia in the z direction[kg m^2]
    k_1 = givens(11);
    k_2 = givens(12);
    k_3 = givens(13);
    k_4 = givens(14);
    k_5 = givens(15);
    k_6 = givens(16);

    % define conditions vector
    deltau = Conditions(1); % inertial velocity in the u direction in body coordinates [m/s]
    deltav = Conditions(2); % inertial velocity in the v direction in body coordinates [m/s]
    deltaw = Conditions(3); % inertial velocity in the w direction in body coordinates [m/s]
    deltap = Conditions(4); % inertial angular velocity in the p direction in body coordinates [rad/s]
    deltaq = Conditions(5); % inertial angular velocity in the q direction in body coordinates [rad/s]
    deltar = Conditions(6); % inertial angular velocity in the r direction in body coordinates [rad/s]
   deltaphi = Conditions(7); % bank [rad]
    deltatheta = Conditions(8); % elevation [rad]
    psi = Conditions(9); % azimuth [rad]
    x_E = Conditions(10); % position vector in the x direction in inertial coordinates [m]
    y_E = Conditions(11); % position vector in the y direction in inertial coordinates [m]
    z_E = Conditions(12); % position vector in the z direction in inertial coordinates [m]
    f1 = Force(1); % trim force exerted by motor 1 [N]
    f2 = Force(2); % trim force exerted by motor 1 [N]
```

```
    f3 = Force(3); % trim force exerted by motor 1 [N]
    f4 = Force(4); % trim force exerted by motor 1 [N]
     if t <= 2
         deltavr = .5; % 2.1 m/s
         deltaur = 0;
     else
         deltavr = 0;
         deltaur=  0;
     end
    % Aerodynamic/Control Moments and Forces
    Laero = - alpha^2 * deltap^2 * sign(deltap); % p component of the aerodynamic moments
    Maero = - alpha^2 * deltaq^2 * sign(deltaq); % q component of the aerodynamic moments
    Naero = - alpha^2 * deltar^2 * sign(deltar); % w component of the aerodynamic moments
 Lcontrol = ((f1 + f2) - (f3 + f4)) * R; % p component of the control moments
    Mcontrol = ((f3 + f2) - (f1 + f4)) * R; % q component of the control moments
    Ncontrol = -0.004*deltar; % w component of the aerodynamic moments
    deltaL = Laero + Lcontrol; % L moment sum (abt x axis)
    deltaM = Maero + Mcontrol; % M moment sum (abt y axis)
    deltaN = Naero + Ncontrol; % N moment sum (abt Z axis)

    if t > 2 && t < 2.5
        deltaL = deltaL + Disturb(2); % Disturbed L moment (abt x axis)
        deltaM = deltaM + Disturb(1); % Disturbed M moment (abt y axis)
        deltaN = deltaN + Disturb(3); % Disturbed N moment (abt z axis)
    end
    Xaero = - eta^2 * deltau^2 * sign(deltau); % x component of aerodynamic force
    Yaero = - eta^2 * deltav^2 * sign(deltav); % y component of aerodynamic force
    Zaero = - eta^2 * deltaw^2 * sign(deltaw); % z component of aerodynamic force
    Xcontrol = 0; % x control
    Ycontrol = 0; % y control
    Zcontrol = -sum(Force); % gravitational force counteraction
    deltaX = Xaero + Xcontrol; % sum of x forces
    deltaY = Yaero + Ycontrol; % sum of y forces


    deltaZ = 0; % sum of z forces
    deltapdot = ((-k_1*deltap)/I_x) -((k_2*deltaphi)/I_x -((k_2*k_3)*(deltavr-deltav))/I_x  ); %
        roll rate derrivative
    deltaqdot = ((-k_4*deltaq)/I_y) -((k_5*deltatheta)/I_y -((k_5*k_6)*(deltaur-deltau))/I_y); %
        pitch rate derrivative


    deltardot = deltaN/I_z; % yaw rate derrivative

    dOmega_bdt = [deltapdot, deltaqdot, deltardot]';
    deltaudot = -g*deltatheta; % 4.7,1 acceleration in the x axis
    deltavdot = g*deltaphi; % acceleration in the y axis
    deltawdot = deltaZ/m; % acceleration in the z axis
    dVbdt = [deltaudot, deltavdot, deltawdot]';
    xdot =  deltau;
    ydot =  deltav;
    zdot = deltaw;
    dVEdt = [xdot, ydot, zdot]';
    phidot  = deltap; % bank roc (pg 104)
    thetadot = deltaq; % elevation roc (pg 104)
    psidot   = deltar; % azimuth roc (pg 104)
    dEuldt = [phidot, thetadot, psidot]';
    dydt = [dVbdt; dOmega_bdt; dEuldt; dVEdt];


end
```

Stop Function

```
% This function is used by ODE45 in options to end the drones
% path after a certain tolerance of values is reached
```

```
%
%   Author: Benjiman Smith
%   Collaborators: E. Owen, I. Quezada
%   Date: 1/26/2020
%
function [value, isTerm, direction] = StopFnct(t, F)
value = F(12);
isTerm = 1;
direction = [];
end
```