

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

<b>I Nomenclature</b>	<b>2</b>
<b>II Question 1</b>	<b>3</b>
<b>III Question 2</b>	<b>11</b>
<b>IV Question 3</b>	<b>15</b>
<b>V MATLAB Code</b>	<b>18</b>

## List of Figures

1 Locus of Lateral rotation dynamics simulation . . . . .	10
2 Locus of Longitudinal rotation dynamics simulation . . . . .	10
3 Indexing code used to find $K_3$ from eigenvalue . . . . .	10
4 Lateral simulation block diagram . . . . .	11
5 Longitudinal simulation block diagram . . . . .	11
6 Implementation of new $\Delta\dot{q}$ and $\Delta\dot{p}$ equations . . . . .	11
7 If/Else statements for feedforward commands . . . . .	12
8 Lateral Change in position over time . . . . .	12
9 Lateral Change in position over time . . . . .	13
10 Deviation laterally . . . . .	13
11 Longitudinal Change in position over time . . . . .	14
12 Longitudinal Change in position over time . . . . .	14
13 Deviation longitudinally . . . . .	15
14 GUI for experimental data . . . . .	16
15 Experimental data, and Simulated data . . . . .	16
16 Experimental data, and Simulated data . . . . .	17

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

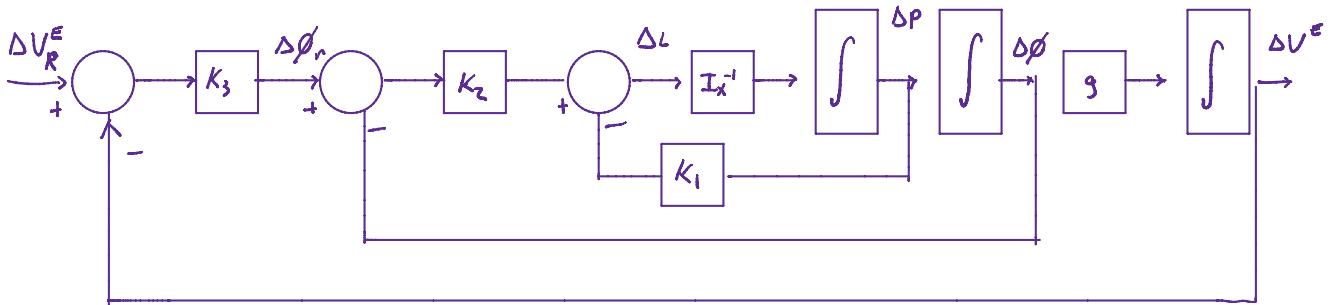
$m$	= mass [kg]
$r$	= body to motor distance [m]
$I_x$	= moment of inertia in the body x direction [ $kgm^2$ ]
$I_y$	= moment of inertia in the body y direction [ $kgm^2$ ]
$I_z$	= moment of inertia in the body z direction [ $kgm^2$ ]
$u^E$	= inertial velocity in the x direction in body coordinates [m/s]
$v^E$	= 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 [ $m/s^2$ ]
$\dot{v}^E$	= 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 [ $m/s^2$ ]
$\Delta u^E$	= derivative of the inertial in the body x direction deviated from trim [m/s <sup>2</sup> ]
$\Delta v^E$	= derivative of the inertial velocity in the body y direction deviated from trim [m/s <sup>2</sup> ]
$\Delta w^E$	= derivative of the inertial velocity in the body z direction deviated from trim [m/s <sup>2</sup> ]
$p$	= 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]
$\Delta p$	= roll rate deviated from trim [radian/s <sup>2</sup> ]
$\Delta q$	= pitch rate deviated from trim [radian/s <sup>2</sup> ]
$\dot{p}$	= roll rate derivative [radian/s <sup>2</sup> ]
$\dot{q}$	= pitch rate derivative [radian/s <sup>2</sup> ]
$\dot{r}$	= yaw rate derivative [radian/s <sup>2</sup> ]
$\Delta \dot{p}$	= roll rate derivative deviated from trim [radian/s <sup>2</sup> ]
$\Delta \dot{q}$	= pitch rate derivative deviated from trim [radian/s <sup>2</sup> ]
$\Delta \dot{r}$	= yaw rate derivative deviated from trim [radian/s <sup>2</sup> ]
$\phi$	= bank angle [radian]
$\theta$	= elevation angle [radian]
$\psi$	= azimuth angle [radian]
$\dot{\phi}$	= bank angle rate of change [radian/s]
$\dot{\theta}$	= elevation angle rate of change [radian/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



$$\Delta L = \underbrace{-K_2 \Delta \phi + K_2 K_3 (\Delta V_R^E - \Delta V^E)}_{\text{+ SIDE OF } \Delta L} + \underbrace{(-K_1 \Delta P)}_{\text{- SIDE OF } \Delta L}$$

$$(4.9,3) \quad \dot{\Delta P} = \frac{1}{I_x} \Delta L \Rightarrow \dot{\Delta P} = \frac{K_2 K_3 \Delta V_R^E}{I_x} - \frac{K_2 K_3 V^E}{I_x} - \frac{K_2 \Delta \phi}{I_x} - \frac{K_1 \Delta P}{I_x}$$

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

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

$$\Rightarrow \begin{bmatrix} \dot{\Delta P} \\ \dot{\Delta \phi} \\ \dot{\Delta V^E} \end{bmatrix} = \begin{bmatrix} -\frac{K_1}{I_x} & -\frac{K_2}{I_x} & -\frac{K_2 K_3}{I_x} \\ 1 & 0 & 0 \\ 0 & g & 0 \end{bmatrix} \begin{bmatrix} \Delta P \\ \Delta \phi \\ \Delta V^E \end{bmatrix} + \begin{bmatrix} K_2 K_3 \Delta V_R^E \\ 0 \\ 0 \end{bmatrix}$$

$$A - \lambda I = \begin{bmatrix} -\frac{K_1}{I_x} - \lambda & -\frac{K_2}{I_x} & -\frac{K_2 K_3}{I_x} \\ 1 & -\lambda & 0 \\ 0 & g & -\lambda \end{bmatrix}$$

$$\det[A - \lambda I] = \left( -\frac{K_1}{I_x} - \lambda \right) \left( \lambda^2 - \left( -\frac{K_2}{I_x} \right) (-\lambda) + \left( \frac{-K_2 K_3}{I_x} \right) (g) \right)$$

$$\frac{-K_1}{I_x} \lambda^2 - \lambda^3 - \frac{K_2 \lambda}{I_x} - \frac{K_2 K_3}{I_x} s = 0$$

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

$$\Rightarrow \lambda^3 + \frac{K_1}{I_x} \lambda^2 + \frac{K_2 \lambda}{I_x} + \frac{K_2 K_3 s}{I_x} = 0$$

Utilizing Matlab's symbolic solver, coupled

with pre-determined  $K_1$  &  $K_2$  values ( $K_1 = 0.0015, K_2 = 0.0027$ )

$\lambda_1, \lambda_2$ , &  $\lambda_3$  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  $\lambda_1, \lambda_2$ , &  $\lambda_3$  are shown below:

FROM CLASS, WE WERE GIVEN THE

FOLLOWING EQUATION

$$e^{\lambda_1 t} = e^{nt} \cdot e^{j\omega t}$$

IN WHICH THE  $e^{j\omega t}$  TERM

IS OSCILLATORY. 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:

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

$$\Rightarrow \ln(e^{\lambda_1 t}) = \ln(e^{nt}) \Rightarrow \lambda_1 t = nt$$

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

THIS ALLOWS FOR US TO SOLVE FOR

$\lambda_1, \lambda_2, \text{ & } \lambda_3$  easily

$\tau = \text{TIME CONSTANT}$

$\tau \leq 1.25 \text{ sec} \text{ (FROM ASSIGNMENT)}$

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

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

↑

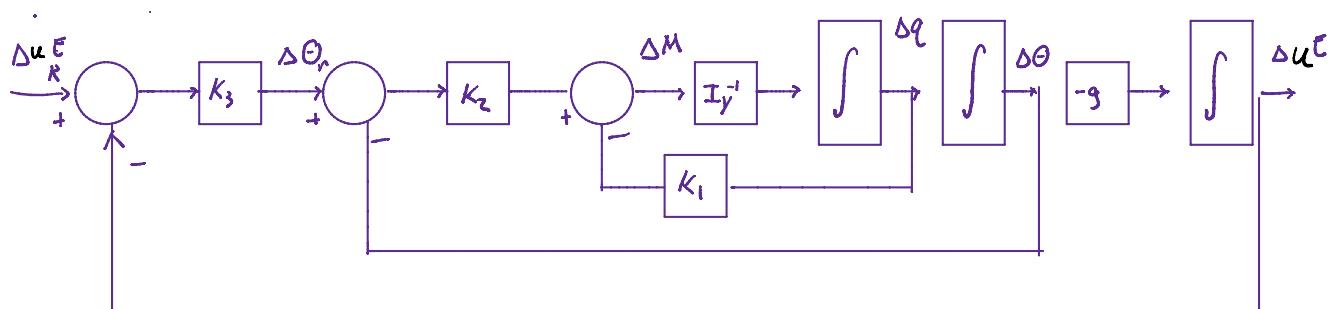
IN THIS CASE,  $n = \lambda_1$

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

USING THIS CONSTRAINT FOR THE EIGENVALUES, THE APPROPRIATE  $K_3$  VALUE COULD BE FOUND BY INDEXING THROUGH THE EIGENVALUES  $\geq -0.8 \text{ sec}$ , THEN FINDING A  $K_3$  VALUE IN THE ROUND INDEX

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

THESE SAME CALCULATIONS WERE ALSO DONE ALONG THE LONGITUDINAL



$$\Delta M = -K_2 \Delta \theta + K_2 K_3 (\Delta w_r^E - \Delta w^E) + (-K_1 \Delta q)$$

$\oplus$  SIDE OF  $\Delta L$

$\ominus$  SIDE OF  $\Delta L$

$$(4.9,3) \quad \dot{\Delta q} = \frac{1}{I_y} \Delta M \Rightarrow \dot{\Delta q} = \frac{K_2 K_3 \Delta w_r^E}{I_y} - \frac{K_2 K_3 w^E}{I_y} - \frac{K_2 \Delta \theta}{I_y} - \frac{K_1 \Delta q}{I_x}$$

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

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

$$\Rightarrow \begin{bmatrix} \dot{\Delta q} \\ \dot{\Delta \theta} \\ \dot{\Delta u}^E \end{bmatrix} = \begin{bmatrix} -\frac{K_1}{I_y} & -\frac{K_2}{I_y} & -\frac{K_2 K_3}{I_y} \\ 1 & 0 & 0 \\ 0 & -g & 0 \end{bmatrix} \begin{bmatrix} \Delta q \\ \Delta \theta \\ \Delta u^E \end{bmatrix} + \begin{bmatrix} K_2 K_3 \Delta u_r^E \\ 0 \\ 0 \end{bmatrix}$$

$$A - \lambda I = \begin{bmatrix} -\frac{K_1}{I_y} - \lambda & -\frac{K_2}{I_y} & -\frac{K_2 K_3}{I_y} \\ 1 & -\lambda & 0 \\ 0 & -g & -\lambda \end{bmatrix}$$

$$\det[A - \lambda I] = \left(-\frac{K_1}{I_y} - \lambda\right)(\lambda^2 - \left(-\frac{K_2}{I_y}\right)(-\lambda) + \left(\frac{-K_2 K_3}{I_y}\right)g)$$

$$\frac{-K_1}{I_y} \lambda^2 - \lambda^3 - \frac{K_2}{I_y} \lambda - \frac{K_2 K_3}{I_y} g = 0$$

$$\Rightarrow \frac{\lambda^3}{I_y} + \frac{K_1}{I_y} \lambda^2 + \frac{K_2}{I_y} \lambda + \frac{K_2 K_3}{I_y} g = 0$$

Utilizing Matlab's symbolic solver, coupled

with pre-determined  $K_1$  &  $K_2$  values ( $K_1 = 0.0015$ ,  $K_2 = 0.0027$ )

$\lambda_1$ ,  $\lambda_2$ , &  $\lambda_3$  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  $\lambda_1$ ,  $\lambda_2$ , &  $\lambda_3$  are shown below:

FROM CLASS, WE WERE GIVEN THE

FOLLOWING EQUATION

$$e^{\lambda_1 t} = e^{nt} \cdot e^{j\omega t}$$

IN WHICH THE  $e^{j\omega t}$  TERM  
 IS OSCILLATORY. 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:

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

$$\begin{aligned} > \ln(e^{\lambda_1 t}) &= \ln(e^{nt}) \Rightarrow \lambda_1 t = nt \\ &\therefore \boxed{\lambda_1 = n} \end{aligned}$$

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

$$\begin{aligned} \tau &\equiv \text{TIME CONSTANT} \\ \tau &\leq 1.25 \text{ sec (FROM ASSIGNMENT)} \end{aligned}$$

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

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

↑

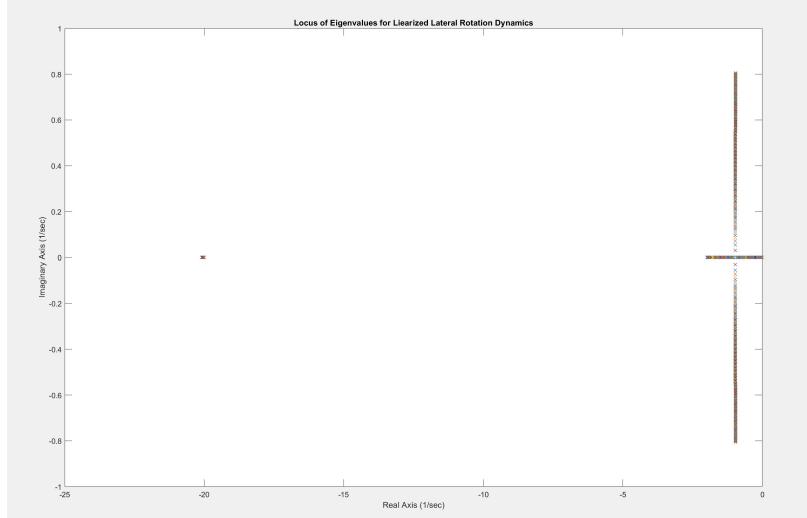
IN THIS CASE,  $n = \lambda_1$

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

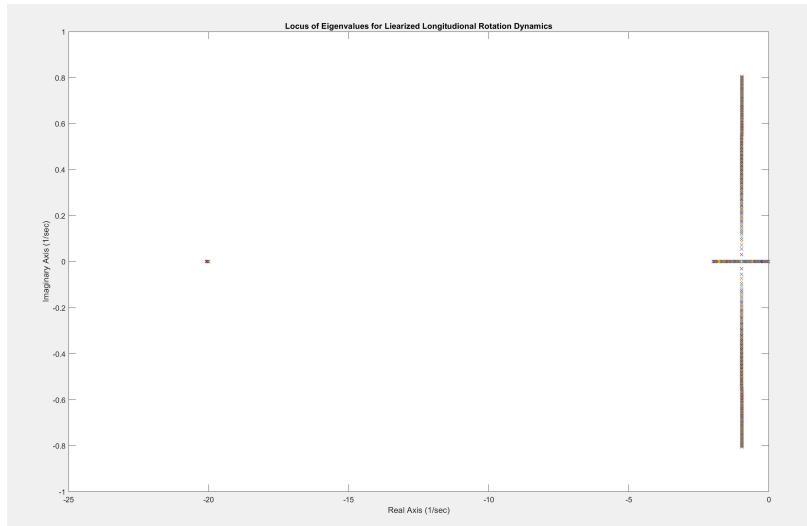
USING THIS CONSTRAINT FOR THE EIGENVALUES, THE APPROPRIATE  $K_3$  VALUE COULD BE FOUND BY INDEXING THROUGH THE EIGENVALUES  $\geq -0.8 \text{ sec}$ , THEN FINDING A  $K_3$  VALUE IN THE FOUND INDEX

$$\Rightarrow K_3 = -0.0470$$

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  $\leq -0.8$  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 feedforward commands for  $\Delta v_R^E$  and  $\Delta u_R^E$  to cause the corresponding inertial displacements  $\Delta y_R^E$  and  $\Delta x_R^E$  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.

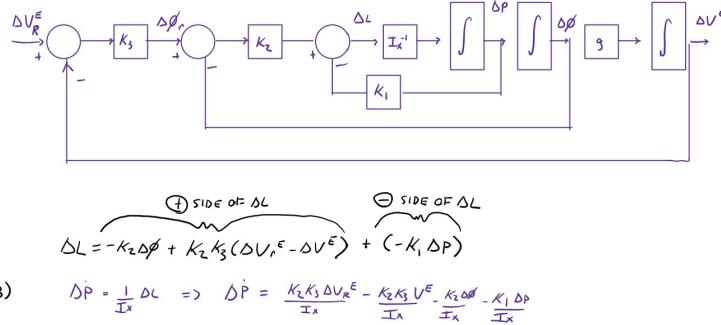


Fig. 4 Lateral simulation block diagram

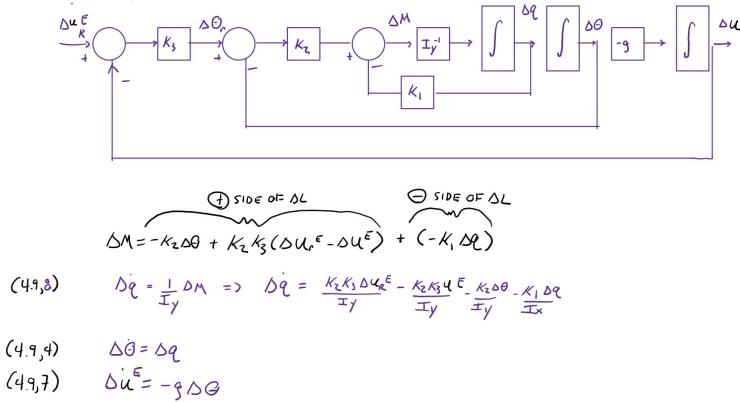


Fig. 5 Longitudinal simulation block diagram

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

```
deltaqdot = ((-k_1*delatap)/I_x) -((k_2*delaphi)/I_x -((k_2*k_3)*(deltavr-deltav))/I_x ); % roll rate derivative
deltaqdot = ((-k_4*delataq)/I_y) -((k_5*deltatheta)/I_y -((k_5*k_6)*(deltaur-deltau))/I_y); % pitch rate derivative
```

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

in order to initiate the feedforward commands for  $\Delta v_R^E$  and  $\Delta u_R^E$ , the *ODE* function developed in Assignment 4 was modified again with an *if* statement which allowed for  $\Delta v_R^E$  and  $\Delta u_R^E$  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  $\Delta y_R^E$  and  $\Delta x_R^E$  to change by 1.0 meters in 2.0 seconds independently, the non-zero constant values assigned to  $v_R^E$  and  $u_R^E$  were 0.5 m/s. After this two second time interval, an *else* statement was used to set  $\Delta v_R^E$  and  $\Delta u_R^E$  back to zero m/s. This set of statements is shown below for the lateral case, including  $\Delta v_R^E$  as the feedforward command, and the longitudinal case, including  $\Delta u_R^E$  as the feedforward 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

```

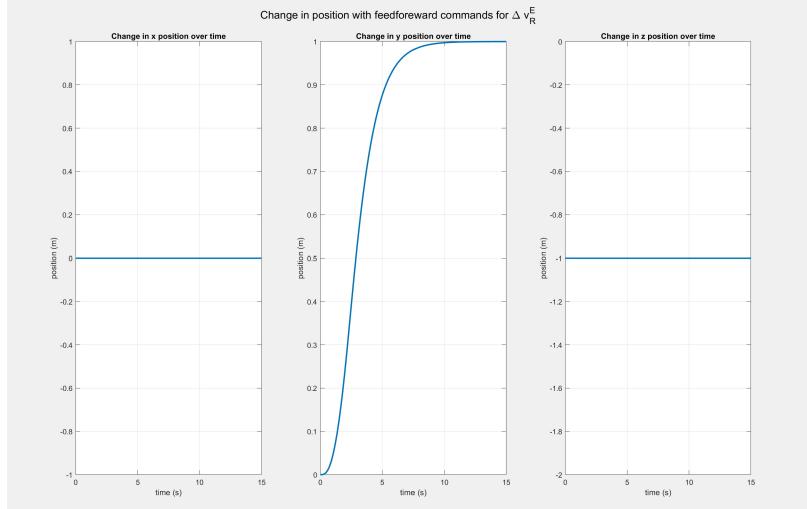
if t <= 2 % time interval for added velocity
    deltaur = 0.5; % set delta ur = 0.5 m/s
else % else statement to return velocity to zero
    deltaur = 0;
end

```

(b) Longitudinal Case

**Fig. 7 If/Else statements for feedforward commands**

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



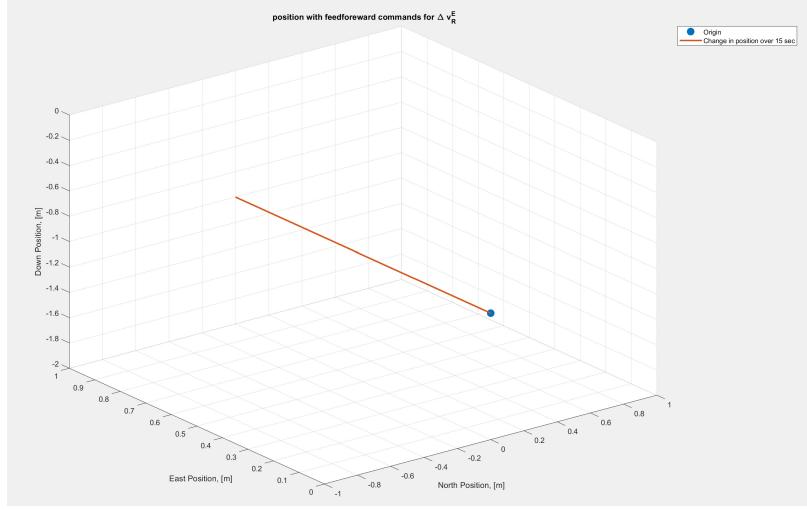
**Fig. 8 Lateral Change in position over time**

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:

$$\Delta \dot{p} = -\frac{K_1 \Delta p}{I_x} - \frac{K_2 \Delta \phi}{I_x} - \frac{K_3 K_2 (\Delta v_R^E - \Delta v^E)}{I_x} \quad (1)$$

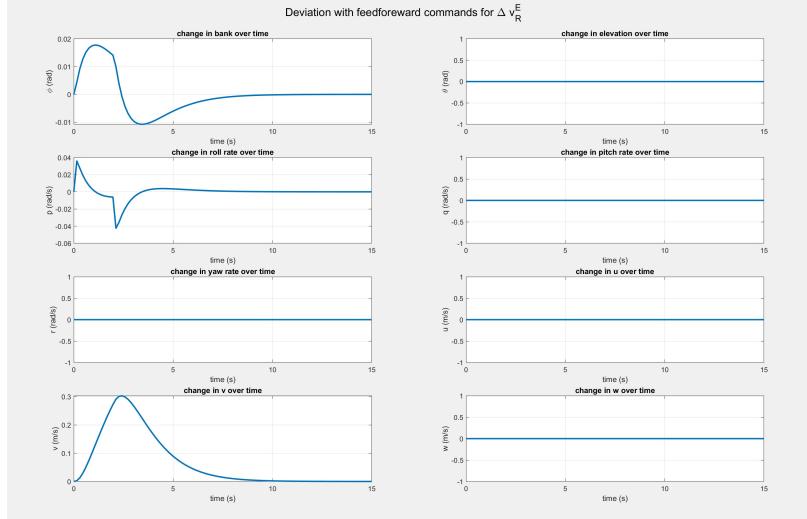
For small  $K_3$  values, the value of  $\Delta \dot{p}$  is smaller, therefore the rolling of the quadcopter becomes slower as  $K_3$  gets smaller. This justifies why the simulation's response time is slower than expected.

A plot of the quadcopter's position was created to see the behavior of the quadcopter during the time of the applied  $\Delta v_R^E$ .



**Fig. 9 Lateral Change in position over time**

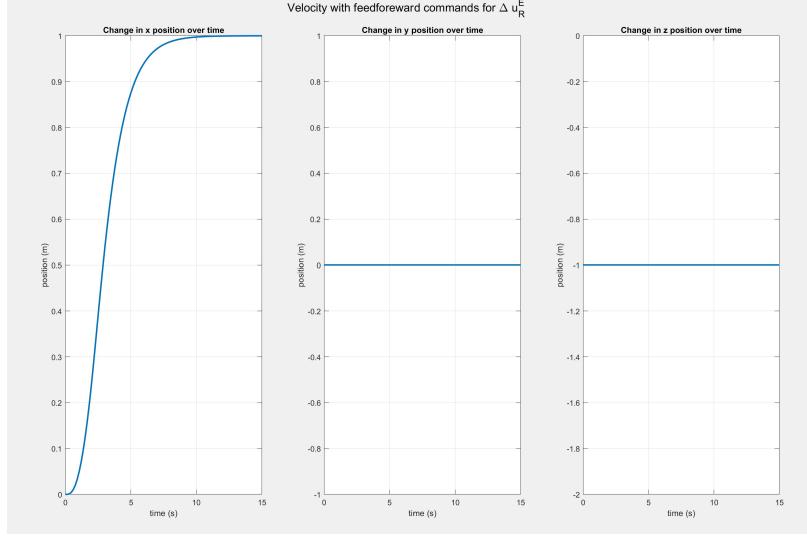
As seen from figure 9 the quadcopter translates steadily in the positive east direction. This behavior was expected, as the  $\Delta v_R^E$  applied to the quadcopter would cause a small initial positive spike in bank, as a positive  $\Delta v_R^E$  requires a positive bank angle. This spike was corrected 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  $\Delta v_R^E$ , 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  $\Delta v_R^E$  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  $\Delta v_R^E$  due to the higher magnitude  $\Delta \dot{\rho}$  values resulting from the increase in  $K_3$  magnitude.

Now, analyzing the longitudinal rotational dynamics simulation,



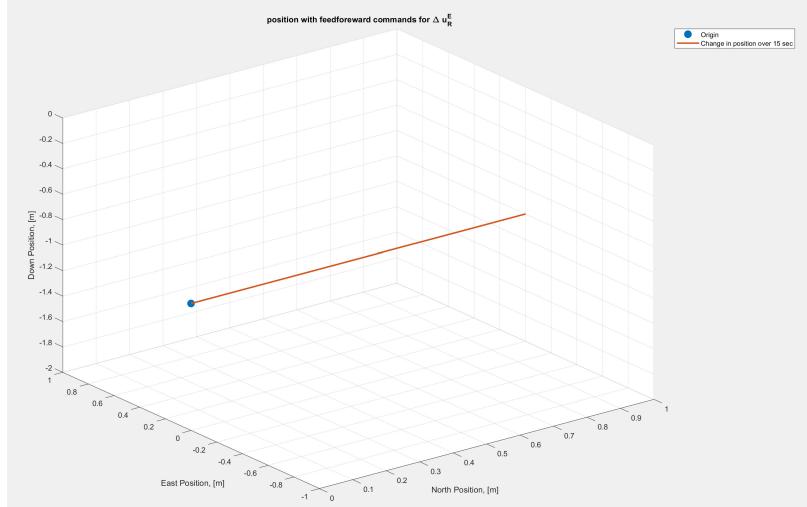
**Fig. 11 Longitudinal Change in position over time**

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:

$$\Delta \dot{q} = -\frac{K_1 \Delta p}{I_y} - \frac{K_2 \Delta \theta}{I_y} - \frac{K_3 K_2 (\Delta u_R^E - \Delta u^E)}{I_y} \quad (2)$$

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.

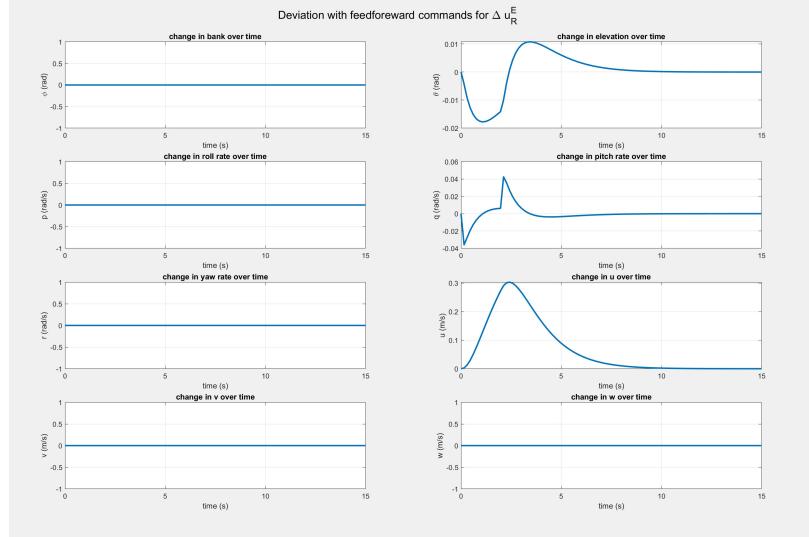
A plot of the quadcopter's position was created to see the behavior of the quadcopter during the time of the applied  $\Delta u_R^E$ .



**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  $\Delta u_R^E$  applied to the quadcopter would cause a small initial spike in elevation in the negative direction, as in order

to have a positive  $\Delta u_R^E$  the elevation must be initially negative. This initial spike was corrected by the implemented  $K_2$  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  $\Delta u_R^E$ . 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.

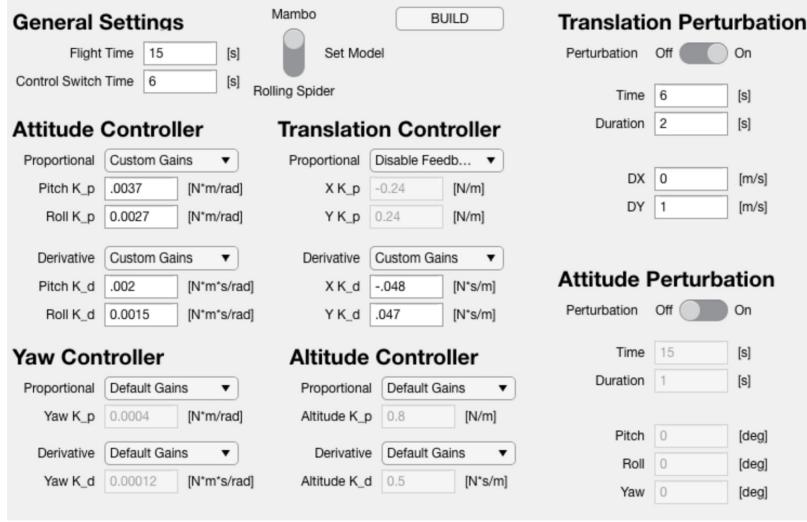


**Fig. 13 Deviation longitudinally**

From figure 13 the lack of maneuverability of the quadcopter can clearly be seen by the change in  $\dot{x}$  direction velocity plot. The low magnitude  $K_3$  value keeps the velocity value from even reaching the specified  $\Delta u_R^E$  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  $\Delta u_R^E$  due to the higher magnitude  $\Delta \dot{p}$  values resulting from the increase in  $K_3$  magnitude.

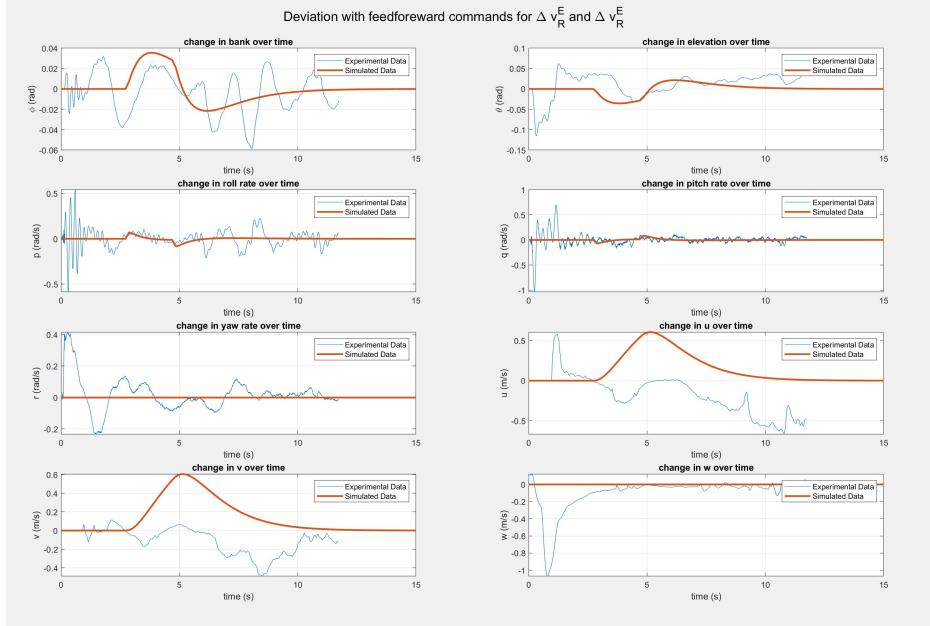
#### IV. Question 3

Implementing 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 derived 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  $\Delta v_R^E$  and  $\Delta v_R^E$  feedforward commands, simulations were run to compare to the experimental data

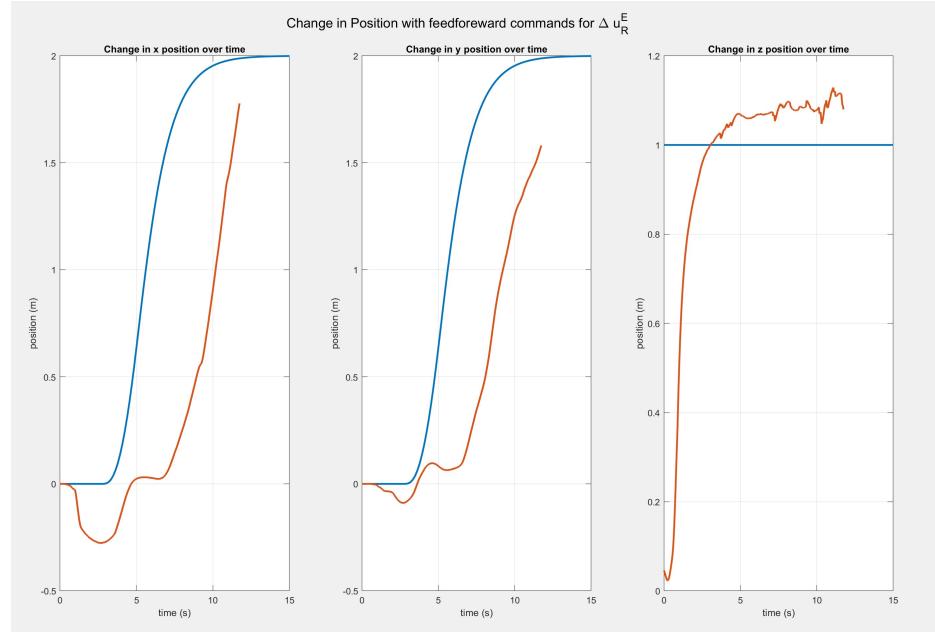


**Fig. 15 Experimental data, and Simulated data**

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  $\Delta v_R^E$  feedforward 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  $\Delta v_R^E$  feedforward 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  $\Delta u_R^E$  feedforward 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  $\Delta u_R^E$  feedforward 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

```

1 clear all
2 close all
3 %% lateral
4 Lambda1 = -2; % 1st lambda value, found from lab calculations
5 Lambda2 = -20;% 2nd lambda value is 10x Lambda 1, in order to make lambda 1 dominant
6 Ix = 6.8e-5; % moment of inertia in the x direction [kg m^2]
7 Iy = 9.2e-5; % Moment of inertia abt y (kgm^2)
8 g = 9.81
9 syms K1 K2 % symbollically solving for K1 and K2
10 eqn1 = Lambda1^2 +Lambda1*(K1/Ix) + (K2/Ix) == 0; % 1st eigenvalue solution
11 eqn2 = Lambda2^2 +Lambda2*(K1/Ix) + (K2/Ix) == 0; % 2nd eigenvalue solution
12
13
14 [A,B] = equationsToMatrix([eqn1, eqn2], [K1, K2]); % convert both the equations and the unknowns
   into independent vectors
15
16 Solution = linsolve(A,B); % solve the system of equations
17 Solution = double(Solution); % convert symbolic solution vector to vector of doubles
18 K1 = Solution(1); % bank control
19 K2 = Solution(2); % bank control
20
21 i = 1;
22 Lambda3 =zeros(3, 491);
23 for K3 = (.001:.0001:.08); % loop for differing K3 values
24 A= [ -K1/Ix, -K2/Ix, -K2*K3/Ix; 1, 0, 0; 0, 0, g, 0]; % matrix derrived in notes from functional
   block diagram
25 Lambda3(:, i) = eig(A); % take the eigenvector of the matrix to get the eigenvalues
26 i = i+1; % iterator
27 end
28
29 figure()
30 K3 = (0.001:0.0001:.08); % reinitialize K3 vector
31 plot(real(Lambda3),imag(Lambda3), 'x'); % plot Eigenvalues 3 vs. K3
32 xlabel('Real Axis (1/sec)');
33 ylabel('Imaginary Axis (1/sec)');
34 title('Locus of Eigenvalues for Lateralized Lateral Rotation Dynamics');
35 index1 = find(Lambda3(3,:) > -0.8, 1, 'last');
36 k3 = K3(index1+1);
37 %% longitudinal
38 syms K4 K5 % symbollically solving for K3 and K4
39 eqn3 = Lambda1^2 +Lambda1*(K4/Iy) + (K5/Iy) == 0; % 1st eigenvalue solution
40 eqn4 = Lambda2^2 +Lambda2*(K4/Iy) + (K5/Iy) == 0; % 1st eigenvalue solution
41
42
43 [C,D] = equationsToMatrix([eqn3, eqn4], [K4, K5]);
44
45 ForceVect2 = linsolve(C,D);
46 ForceVect2 = double(ForceVect2);
47 K4 = ForceVect2(1); % elevation control
48 K5 = ForceVect2(2); % elevation control
49
50
51
52 i = 1;
53 for K6 = -(0.001:.0001:.08) % loop for differing K3 values
54 B= [ -K4/Iy, -K5/Iy, -(K5*K6)/Iy; 1, 0, 0; 0, 0, -g, 0]; % matrix derrived in notes from functional
   block diagram
55 Lambda6(:, i) = eig(B); % take the eigenvector of the matrix to get the eigenvalues
56 i = i+1; % iterator
57 end
58
59
60 K6 = -(0.001:.0001:.08); % reinitialize K3 vector

```

```

62 %plot(K6,real(Lambda6(3,:))); % plot Eigenvalues 3 vs. K3
63 figure()
64 plot(real(Lambda6),imag(Lambda6), 'x'); % plot Eigenvalues 3 vs. K3
65 xlabel('Real Axis (1/sec)');
66 ylabel('Imaginary Axis (1/sec)');
67 title('Locus of Eigenvalues for Linearized Longitudinal Rotation Dynamics');
68
69 index2 = find(Lambda6(3,:) > -0.8, 1, 'last');
70 k6 = K6(index2+1);
71 %%

```

## Question 2 Script

```

1 % This script initializes initial conditions for the trajectory of a drone
2 % in steady flight conditions (hovering) then calls the ODE45 function to
3 % numerically integrate the position of the drone with respect to the
4 % initial conditions, along with plotting the results answering Question
5 % 7
6 %
7 % Author: Benjamin Smith
8 % Collaborators: E. Owen, I. Quezada
9 % Date: 1/25/2020
10 clear all
11 close all
12 %% lateral
13 Lambda1 = -2; % 1st lambda value, found from lab calculations
14 Lambda2 = -20;% 2nd lambda value is 10x Lambda 1, in order to make lambda 1 dominant
15 Ix = 6.8e-5; % moment of inertia in the x direction [kg m^2]
16 Iy = 9.2e-5; % Moment of inertia abt y (kgm^2)
17 g = 9.81;
18 syms K1 K2 % symbollically solving for K1 and K2
19 eqn1 = Lambda1^2 +Lambda1*(K1/Ix) + (K2/Ix) == 0; % 1st eigenvalue solution
20 eqn2 = Lambda2^2 +Lambda2*(K1/Ix) + (K2/Ix) == 0; % 2nd eigenvalue solution
21
22
23 [A,B] = equationsToMatrix([eqn1, eqn2], [K1, K2]); % convert both the equations and the unknowns
   into independent vectors
24
25 Solution = linsolve(A,B); % solve the system of equations
26 Solution = double(Solution); % convert symbolic solution vector to vector of doubles
27 K1 = Solution(1); % bank control
28 K2 = Solution(2); % bank control
29
30 i = 1;
31 Lambda3 =zeros(3, 491);
32 for K3 = (.001:0.0001:.08) % loop for differing K3 values
33 A= [ -K1/Ix, -K2/Ix, -K2*K3/Ix; 1, 0, 0; 0, 0, g, 0]; % matrix derived in notes from functional
   block diagram
34 Lambda3(:, i) = eig(A); % take the eigenvector of the matrix to get the eigenvalues
35 i = i+1; % iterator
36 end
37
38 %figure(6)
39 hold on
40 K3 = (0.001:0.0001:.08); % reinitialize K3 vector
41 %plot(K3,real(Lambda3(3,:))); % plot Eigenvalues 3 vs. K3
42
43 index1 = find(Lambda3(3,:) > -0.8, 1, 'last');
44 k3 = K3(index1+1);
45 %% longitudinal
46 syms K4 K5 % symbollically solving for K3 and K4
47 eqn3 = Lambda1^2 +Lambda1*(K4/Iy) + (K5/Iy) == 0; % 1st eigenvalue solution
48 eqn4 = Lambda2^2 +Lambda2*(K4/Iy) + (K5/Iy) == 0; % 1st eigenvalue solution
49
50
51 [C,D] = equationsToMatrix([eqn3, eqn4], [K4, K5]);
52

```

```

53 ForceVect2 = linsolve(C,D);
54 ForceVect2 = double(ForceVect2);
55 K4 = ForceVect2(1); % elevation control
56 K5 = ForceVect2(2); % elevation control
57
58
59
60 i = 1;
61 for K6 = -.001:.0001:.08) % loop for differing K3 values
62 B= [ -K4/Iy, -K5/Iy, -(K5*K6)/Iy; 1, 0, 0; 0, 0, -g, 0]; % matrix derrived in notes from functional
       block diagram
63 Lambda6(:, i) = eig(B); % take the eigenvector of the matrix to get the eigenvalues
64 i = i+1; % iterator
65 end
66
67
68
69 K6 = -.001:.0001:.08); % reinitialize K3 vector
70 %plot(K6,real(Lambda6(3,:))); % plot Eigenvalues 3 vs. K3
71
72
73 index2 = find(Lambda6(3,:) > -0.8, 1, 'last');
74 k6 = K6(index2+1);
75 %% feedforward commands for \Delta v^{E}_{R}
76
77 m = 0.068; % mass of the drone [kg]
78 r = 0.06; % body to motor distance [m]
79 k = 0.0024; % [Nm/N]
80 rad = r/sqrt(2); % [m]
81 g = 9.81; % gravity [m/s^2]
82 alpha = 2e-6; % [N/(m/s)^2]
83 eta = 1e-3; % [N/(rad/s)^2]
84 Ix = 6.8e-5; % moment of inertia in the x direction [kg m^2]
85 Iy = 9.2e-5; % moment of inertia in the y direction [kg m^2]
86 Iz = 1.35e-4; % moment of inertia in the z direction [kg m^2]
87 givens = [alpha eta Ix Iy Iz m r k rad g K1 K2 k3 K4 K5 k6]; % givens vector
88 F = m*g;
89 tspan = linspace(0,15); % time vector
90 Pertubations = zeros(1, 3); % No perturbation
91 TrimForces = ones(1, 4) * m * g / 4; % forces required by each motor to maintain hover
92 conditions = zeros(1, 12); % initialize conditions vector (very large);
93 conditions(12) = -1; % set down direction to 1 to make signs correct for plotting
94 t1 = 0;
95 t2 = 0;
96 X = 0;
97 X2 = 0;
98 options = odeset('Events', @StopFnct, 'RelTol', 1e-8); % stop function that ends ODE when a
   tolerance of 1e-8 is met
99 [t1, X] = ode45(@(t, F)Specs2LB5LCV(t, F, TrimForces, Pertubations, givens), tspan, conditions,
   options);
100 figure()
101 sgttitle('Change in position with feedforward commands for \Delta v^{E}_{R}');
102 subplot(1,3,1);
103 plot(t1, X(:,10), 'linewidth', 2);
104 grid on
105 xlabel('time (s)')
106 ylabel('position (m)')
107 title('Change in x position over time')
108 subplot(1,3,2);
109 plot(t1, X(:,11), 'linewidth', 2);
110 grid on
111 xlabel('time (s)')
112 ylabel('position (m)')
113 title('Change in y position over time')
114 subplot(1,3,3);
115 plot(t1, X(:,12), 'linewidth', 2);
116 grid on
117 xlabel('time (s)')

```

```

118 ylabel('position (m)')
119 title('Change in z position over time')
120
121 figure()
122 plot3(0, 0, -1, '.', 'MarkerSize', 35);
123 hold on
124 plot3(X(:, 10), X(:, 11), X(:, 12), 'LineWidth', 2)
125 grid on
126 xlabel('Down Position, [m]')
127 xlabel('North Position, [m]')
128 ylabel('East Position, [m]')
129 title(' position with feedforward commands for \Delta v^{E}_{-R}');
130 legend('Origin', 'Change in position over 15 sec');
131 hold off
132
133
134
135
136 figure()
137 sgttitle('Deviation with feedforward commands for \Delta v^{E}_{-R}');
138 subplot(4,2,1);
139 plot(t1, X(:,7), 'linewidth', 2);
140
141 grid on
142 xlabel('time (s)')
143 ylabel('\phi (rad)')
144 title('change in bank over time')
145
146 hold off
147 %
148 subplot(4,2,2);
149 plot(t1, X(:,8), 'linewidth', 2);
150
151 grid on
152 xlabel('time (s)')
153 ylabel('\theta (rad)')
154 title('change in elevation over time')
155 hold off
156 %
157 subplot(4,2,3);
158 plot(t1, X(:,4), 'linewidth', 2);
159
160 grid on
161 xlabel('time (s)')
162 ylabel('p (rad/s)')
163 title('change in roll rate over time')
164
165 hold off
166 %
167 subplot(4,2,4);
168 plot(t1, X(:,5), 'linewidth', 2);
169
170 grid on
171 xlabel('time (s)')
172 ylabel('q (rad/s)')
173 title('change in pitch rate over time')
174
175 hold off
176 %
177 subplot(4,2,5);
178 plot(t1, X(:,6), 'linewidth', 2);
179
180 grid on
181 xlabel('time (s)')
182 ylabel('r (rad/s)')
183 title('change in yaw rate over time')
184
185 hold off

```

```

186 %
187 subplot(4,2,6);
188 plot(t1, X(:,1), 'linewidth', 2);
189
190 grid on
191 xlabel('time (s)')
192 ylabel('u (m/s)')
193 title('change in u over time')
194
195 hold off
196 %
197 subplot(4,2,7);
198 plot(t1, X(:,2), 'linewidth', 2);
199
200 grid on
201 xlabel('time (s)')
202 ylabel('v (m/s)')
203 title('change in v over time')
204
205 hold off
206 %
207 subplot(4,2,8);
208 plot(t1, X(:,3), 'linewidth', 2);
209 grid on
210 xlabel('time (s)')
211 ylabel('w (m/s)')
212 title('change in w over time')
213 hold off
214
215
216 %% feedforward commands for \Delta u^{(E)}_{-(R)}
217 m = 0.068; % mass of the drone [kg]
218 r = 0.06; % body to motor distance [m]
219 k = 0.0024; % [Nm/N]
220 rad = r/sqrt(2); % [m]
221 g = 9.81; % gravity [m/s^2]
222 alpha = 2e-6; % [N/(m/s)^2]
223 eta = 1e-3; % [N/(rad/s)^2]
224 Ix = 6.8e-5; % moment of inertia in the x direction [kg m^2]
225 Iy = 9.2e-5; % moment of inertia in the y direction [kg m^2]
226 Iz = 1.35e-4; % moment of inertia in the z direction [kg m^2]
227 givens = [alpha eta Ix Iy Iz m r k rad g K1 K2 k3 K4 K5 k6]; % givens vector
228 F = m*g;
229 tspan = linspace(0,15); % time vector
230 Pertubations = zeros(1, 3); % No perturbation
231 TrimForces = ones(1, 4) * m * g / 4; % forces required by each motor to maintain hover
232 conditions = zeros(1, 12); % initialize conditions vector (very large);
233 conditions(12) = -1; % set down direction to 1 to make signs correct for plotting
234 t1 = 0;
235 t2 = 0;
236 X = 0;
237 X2 = 0;
238 options = odeset('Events', @StopFnct, 'RelTol', 1e-8); % stop function that ends ODE when a
               % tolerance of 1e-8 is met
239 [t1, X] = ode45(@(t, F) Specs2LB5LCU(t, F, TrimForces, Pertubations, givens), tspan, conditions,
               options);
240 figure()
241 sgttitle('Velocity with feedforward commands for \Delta u^{(E)}_{-(R)}');
242 subplot(1,3,1);
243 plot(t1, X(:,10), 'linewidth', 2);
244 grid on
245 xlabel('time (s)')
246 ylabel('position (m)')
247 title('Change in x position over time')
248 subplot(1,3,2);
249 plot(t1, X(:,11), 'linewidth', 2);
250 grid on
251 xlabel('time (s)')

```

```

252 ylabel('position (m)')
253 title('Change in y position over time')
254 subplot(1,3,3);
255 plot(t1, X(:,12), 'linewidth', 2);
256 grid on
257 xlabel('time (s)')
258 ylabel('position (m)')
259 title('Change in z position over time')
260
261 figure()
262 plot3(0, 0, -1, '.', 'MarkerSize',35);
263 hold on
264 plot3(X(:, 10), X(:, 11), X(:, 12), 'lineWidth',2)
265 grid on
266 zlabel('Down Position, [m]')
267 xlabel('North Position, [m]')
268 ylabel('East Position, [m]')
269 title(' position with feedforward commands for \Delta u^{E}_{(R)}');
270 legend('Origin', 'Change in position over 15 sec');
271 hold off
272 figure()
273
274
275 sgtitle('Deviation with feedforward commands for \Delta u^{E}_{(R)}');
276 subplot(4,2,1);
277 plot(t1, X(:,7), 'linewidth', 2);
278
279 grid on
280 xlabel('time (s)')
281 ylabel('\phi (rad)')
282 title('change in bank over time')
283
284 hold off
285 %
286 subplot(4,2,2);
287 plot(t1, X(:,8), 'linewidth', 2);
288
289 grid on
290 xlabel('time (s)')
291 ylabel('\theta (rad)')
292 title('change in elevation over time')
293 hold off
294 %
295 subplot(4,2,3);
296 plot(t1, X(:,4), 'linewidth', 2);
297
298 grid on
299 xlabel('time (s)')
300 ylabel('p (rad/s)')
301 title('change in roll rate over time')
302
303 hold off
304 %
305 subplot(4,2,4);
306 plot(t1, X(:,5), 'linewidth', 2);
307
308 grid on
309 xlabel('time (s)')
310 ylabel('q (rad/s)')
311 title('change in pitch rate over time')
312
313 hold off
314 %
315 subplot(4,2,5);
316 plot(t1, X(:,6), 'linewidth', 2);
317
318 grid on
319 xlabel('time (s)')

```

```

320 ylabel('r (rad/s)')
321 title('change in yaw rate over time')
322
323 hold off
324 %
325 subplot(4,2,6);
326 plot(t1, X(:,1), 'linewidth', 2);
327
328 grid on
329 xlabel('time (s)')
330 ylabel('u (m/s)')
331 title('change in u over time')
332
333 hold off
334 %
335 subplot(4,2,7);
336 plot(t1, X(:,2), 'linewidth', 2);
337
338 grid on
339 xlabel('time (s)')
340 ylabel('v (m/s)')
341 title('change in v over time')
342
343 hold off
344 %
345 subplot(4,2,8);
346 plot(t1, X(:,3), 'linewidth', 2);
347 grid on
348 xlabel('time (s)')
349 ylabel('w (m/s)')
350 title('change in w over time')
351 hold off

```

### Question 3 Script

```

1 % This script answers question 4 by plotting the experimental data from
2 % 11:37, 2/14/2020, and plotting the simulated linearized feedback control
3 % simulation on top for comparison
4 % Author: Benjiman Smith
5 % Collaborators: E. Owen, I. Quezada
6 % Date: 2/20/2020
7 %

8
9 clear all
10 close all
11 clc
12 Lambda1 = -2; % 1st lambda value, found from lab calculations
13 Lambda2 = -20;% 2nd lambda value is 10x Lambda 1, in order to make lambda 1 dominant
14 m = 0.068; % mass of the drone [kg]
15 r = 0.06; % body to motor distance [m]
16 k = 0.0024; % [Nm/N]
17 rad = r/sqrt(2); % [m]
18 g = 9.81; % gravity [m/s^2]
19 alpha = 2e-6; % [N/(m/s)^2]
20 eta = 1e-3; % [N/(rad/s)^2]
21 Ix = 6.8e-5; % moment of inertia in the x direction [kg m^2]
22 Iy = 9.2e-5; % moment of inertia in the y direction [kg m^2]
23 Iz = 1.35e-4; % moment of inertia in the z direction [kg m^2]
24 syms K1 K2 % symbollically solving for K1 and K2
25 eqn1 = Lambda1^2 +Lambda1*(K1/Ix) + (K2/Ix) == 0; % 1st eigenvalue solution
26 eqn2 = Lambda2^2 +Lambda2*(K1/Ix) + (K2/Ix) == 0; % 2nd eigenvalue solution
27
28
29 [A,B] = equationsToMatrix([eqn1, eqn2], [K1, K2]); % convert both the equations and the unknowns
   into independent vectors
30
31 Solution = linsolve(A,B); % solve the system of equations

```

```

32 Solution = double(Solution); % convert symbolic solution vector to vector of doubles
33 K1 = Solution(1); % bank control
34 K2 = Solution(2); % bank control
35
36 i = 1;
37 Lambda3 =zeros(3, 491);
38 for K3 = (.001:.0001:.08) % loop for differing K3 values
39 A= [ -K1/Ix, -K2/Ix, -K2*K3/Ix; 1, 0, 0; 0, 0, g, 0]; % matrix derrived in notes from functional
    block diagram
40 Lambda3(:, i) = eig(A); % take the eigenvector of the matrix to get the eigenvalues
41 i = i+1; % itterator
42 end
43
44 %figure(6)
45 hold on
46 K3 = (0.001:0.0001:.08); % reinitialize K3 vector
47 %plot(K3,real(Lambda3(3,:))); % plot Eigenvalues 3 vs. K3
48
49 index1 = find(Lambda3(3,:) > -0.8, 1, 'last');
50 k3 = K3(index1+1);
51 %% longitudinal
52 syms K4 K5 % symbollically solving for K3 and K4
53 eqn3 = Lambda1^2 +Lambda1*(K4/Iy) + (K5/Iy) == 0; % 1st eigenvalue solution
54 eqn4 = Lambda2^2 +Lambda2*(K4/Iy) + (K5/Iy) == 0; % 1st eigenvalue solution
55
56
57 [C,D] = equationsToMatrix([eqn3, eqn4], [K4, K5]);
58
59 ForceVect2 = linsolve(C,D);
60 ForceVect2 = double(ForceVect2);
61 K4 = ForceVect2(1); % elevation control
62 K5 = ForceVect2(2); % elevation control
63
64
65
66 i = 1;
67 for K6 = -(0.001:.0001:.08) % loop for differing K3 values
68 B= [ -K4/Iy, -K5/Iy, -(K5*K6)/Iy; 1, 0, 0; 0, 0, -g, 0]; % matrix derrived in notes from functional
    block diagram
69 Lambda6(:, i) = eig(B); % take the eigenvector of the matrix to get the eigenvalues
70 i = i+1; % itterator
71 end
72
73
74
75 K6 = -(0.001:.0001:.08); % reinitialize K3 vector
76 %plot(K6,real(Lambda6(3,:))); % plot Eigenvalues 3 vs. K3
77
78
79 index2 = find(Lambda6(3,:) > -0.8, 1, 'last');
80 k6 = K6(index2+1);
81
82 givens = [alpha eta Ix Iy Iz m r k rad g K1 K2 k3 K4 K5 k6]; % givens vector
83 F = m*g;
84
85
86 load('RSdata_1023.mat'); % load in data
87 % loading in of parameters
88 xE = rt_estim.signals.values(:,1);
89 yE = rt_estim.signals.values(:,2);
90 zE = rt_estim.signals.values(:,3);
91 theta = rt_estim.signals.values(:,5);
92 phi = rt_estim.signals.values(:,6);
93 u = rt_estim.signals.values(:,7);
94 v = rt_estim.signals.values(:,8);
95 w = rt_estim.signals.values(:,9);
96 p = rt_estim.signals.values(:,10);
97 q = rt_estim.signals.values(:,11);

```

```

98 r = rt_estim.signals.values(:,12);
99 times =rt_estim.time(:);
100
101 startpoint = max(theta); % find maximum theta angle, beginning of disturbance
102 timestamp = find(theta==startpoint); % find the index of this maximum value
103 subtracttime = times(timestamp); % find the time at which the maximum occurs
104 subtracttime = 4;
105
106 tspan = linspace(0,15); % time vector
107 Pertubations = zeros(1, 3); % No perturbation
108 TrimForces = ones(1, 4) * m * g / 4; % forces required by each motor to maintain hover
109 conditions = zeros(1, 12); % initialize conditions vector (very large)
110 conditions(12) = 1; % set down direction to 1 to make signs correct for plotting
111 t1 = 0; % initialize time
112 X = 0;% initialize x
113
114 options = odeset('Events', @StopFnct, 'RelTol', 1e-8); % stop function that ends ODE when a
    tolerance of 1e-8 is met
115 [t1, X] = ode45(@(t, F)Specs2LB5LC(t, F, TrimForces, Pertubations, givens), tspan, conditions,
    options); % linear controlled ode call
116
117 %% plotting
118 % sgtitle('Deviation with feedforward commands for \Delta v^{E}_{R} and \Delta u^{E}_{R}'); %
    subplot title
119 % subplot(4,2,1);
120 % plot(times, phi); % plot experimental data
121 %
122 % grid on
123 % xlabel('time (s)')
124 % ylabel('\phi (rad)')
125 % title('change in bank over time')
126 % legend('Experimental Data');
127 % hold off
128 % subplot(4,2,2);
129 % plot(times, theta);
130 %
131 % xlabel('time (s)')
132 % ylabel('\theta (rad)')
133 % title('change in elevation over time')
134 % legend('Experimental Data');
135 % hold off
136 % subplot(4,2,3);
137 % plot(times, p);
138 %
139 % grid on
140 % xlabel('time (s)')
141 % ylabel('p (rad/s)')
142 % title('change in roll rate over time')
143 % legend('Experimental Data');
144 % hold off
145 % subplot(4,2,4);
146 % plot(times, q);
147 %
148 % grid on
149 % xlabel('time (s)')
150 % ylabel('q (rad/s)')
151 % title('change in pitch rate over time')
152 % legend('Experimental Data');
153 % hold off
154 % subplot(4,2,5);
155 % plot(times, r);
156 %
157 % grid on
158 % xlabel('time (s)')
159 % ylabel('r (rad/s)')
160 % title('change in yaw rate over time')
161 % legend('Experimental Data');
162 % hold off

```

```

163 % subplot(4,2,6);
164 % plot(times, u);
165 %
166 % grid on
167 % xlabel('time (s)')
168 % ylabel('u (m/s)')
169 % title('change in u over time')
170 % legend('Experimental Data');
171 % hold off
172 % subplot(4,2,7);
173 % plot(times, v);
174 %
175 % grid on
176 % xlabel('time (s)')
177 % ylabel('v (m/s)')
178 % title('change in v over time')
179 % legend('Experimental Data');
180 % hold off
181 % subplot(4,2,8);
182 % plot(times, w);
183 %
184 % grid on
185 % xlabel('time (s)')
186 % ylabel('w (m/s)')
187 % title('change in w over time')
188 %
189 % legend('Experimental Data');
190 % hold off
191 %
192 %
193 %%%%%%%%%%%%%%
194 % figure()
195 % sgttitle('Deviation with feedforward commands for \Delta v^{E}_{R}');
196 % subplot(4,2,1);
197 % plot(t1, X(:,7), 'linewidth', 2);
198 %
199 % grid on
200 % xlabel('time (s)')
201 % ylabel('\phi (rad)')
202 % title('change in bank over time')
203 %
204 % hold off
205 %
206 % subplot(4,2,2);
207 % plot(t1, X(:,8), 'linewidth', 2);
208 %
209 % grid on
210 % xlabel('time (s)')
211 % ylabel('\theta (rad)')
212 % title('change in elevation over time')
213 % hold off
214 %
215 % subplot(4,2,3);
216 % plot(t1, X(:,4), 'linewidth', 2);
217 %
218 % grid on
219 % xlabel('time (s)')
220 % ylabel('p (rad/s)')
221 % title('change in roll rate over time')
222 %
223 % hold off
224 %
225 % subplot(4,2,4);
226 % plot(t1, X(:,5), 'linewidth', 2);
227 %
228 % grid on
229 % xlabel('time (s)')
230 % ylabel('q (rad/s)')

```

```

231 % title('change in pitch rate over time')
232 %
233 % hold off
234 %
235 % subplot(4,2,5);
236 % plot(t1, X(:,6),'linewidth', 2);
237 %
238 % grid on
239 % xlabel('time (s)')
240 % ylabel('r (rad/s)')
241 % title('change in yaw rate over time')
242 %
243 % hold off
244 %
245 % subplot(4,2,6);
246 % plot(t1, X(:,1),'linewidth', 2);
247 %
248 % grid on
249 % xlabel('time (s)')
250 % ylabel('u (m/s)')
251 % title('change in u over time')
252 %
253 % hold off
254 %
255 % subplot(4,2,7);
256 % plot(t1, X(:,2),'linewidth', 2);
257 %
258 % grid on
259 % xlabel('time (s)')
260 % ylabel('v (m/s)')
261 % title('change in v over time')
262 %
263 % hold off
264 %
265 % subplot(4,2,8);
266 % plot(t1, X(:,3),'linewidth', 2);
267 %
268 % grid on
269 % xlabel('time (s)')
270 % ylabel('w (m/s)')
271 % title('change in w over time')
272 %
273 % plotting
274 sgtitle('Deviation with feedforward commands for \Delta v^{(E)}_{(R)} and \Delta v^{(E)}_{(R)'}'); %
275 % subplot title
276 subplot(4,2,1);
277 plot(times, phi); % plot experimental data
278 hold on
279 plot(t1 , X(:,7), 'linewidth', 2) % plot simulator data
280 grid on
281 xlabel('time (s)')
282 ylabel('\phi (rad)')
283 title('change in bank over time')
284 legend('Experimental Data', 'Simulated Data');
285 hold off
286 subplot(4,2,2);
287 plot(times, theta);
288 hold on
289 plot(t1 , X(:,8), 'linewidth', 2);
290 grid on
291 xlabel('time (s)')
292 ylabel('\theta (rad)')
293 title('change in elevation over time')
294 legend('Experimental Data', 'Simulated Data');
295 hold off
296 subplot(4,2,3);
297 plot(times, p);
298 hold on

```

```

298 plot(t1 , X(:,4), 'linewidth', 2);
299 grid on
300 xlabel('time (s)')
301 ylabel('p (rad/s)')
302 title('change in roll rate over time')
303 legend('Experimental Data', 'Simulated Data');
304 hold off
305 subplot(4,2,4);
306 plot(times, q);
307 hold on
308 plot(t1 , X(:,5), 'linewidth', 2);
309 grid on
310 xlabel('time (s)')
311 ylabel('q (rad/s)')
312 title('change in pitch rate over time')
313 legend('Experimental Data', 'Simulated Data');
314 hold off
315 subplot(4,2,5);
316 plot(times, r);
317 hold on
318 plot(t1 , X(:,6), 'linewidth', 2);
319 grid on
320 xlabel('time (s)')
321 ylabel('r (rad/s)')
322 title('change in yaw rate over time')
323 legend('Experimental Data', 'Simulated Data');
324 hold off
325 subplot(4,2,6);
326 plot(times, u);
327 hold on
328 plot(t1 , X(:,1), 'linewidth', 2);
329 grid on
330 xlabel('time (s)')
331 ylabel('u (m/s)')
332 title('change in u over time')
333 legend('Experimental Data', 'Simulated Data');
334 hold off
335 subplot(4,2,7);
336 plot(times, v);
337 hold on
338 plot(t1 , X(:,2), 'linewidth', 2);
339 grid on
340 xlabel('time (s)')
341 ylabel('v (m/s)')
342 title('change in v over time')
343 legend('Experimental Data', 'Simulated Data');
344 hold off
345 subplot(4,2,8);
346 plot(times, w);
347 hold on
348 plot(t1 , X(:,3), 'linewidth', 2);
349 grid on
350 xlabel('time (s)')
351 ylabel('w (m/s)')
352 title('change in w over time')
353
354 legend('Experimental Data', 'Simulated Data');
355 hold off
356 figure()
357 sgttitle('Change in Position with feedforward commands for \Delta u^E_{R}');
358 subplot(1,3,1);
359 plot(t1, X(:,10), 'linewidth', 2);
360 hold on
361 plot(times, -xE, 'linewidth', 2);
362 grid on
363 xlabel('time (s)')
364 ylabel('position (m)')
365 title('Change in x position over time')

```

```

366 subplot(1,3,2);
367 plot(t1, X(:,11), 'linewidth', 2);
368 grid on
369 hold on
370 plot(times, -yE,'linewidth', 2);
371 xlabel('time (s)')
372 ylabel('position (m)')
373 title('Change in y position over time')
374 subplot(1,3,3);
375 plot(t1, X(:,12), 'linewidth', 2);
376 grid on
377 hold on
378 plot(times, -zE,'linewidth', 2);
379 xlabel('time (s)')
380 ylabel('position (m)')
381 title('Change in z position over time')

```

### linearized feedback control ODE function with $\Delta u_R^E$ and $\Delta v_R^E$ varying

```

1 % linear ODE Function
2 %Benjiman Smith
3 %02/13/2020
4 function dydt = Specs2LB5LC(t, Conditions, Force, Disturb, givens)
5 % define givens
6 m = givens(6); % mass of the drone [kg]
7 r = givens(7); % body to motor distance [m]
8 k = givens(8); % [Nm/N]
9 R = givens(9); % [m]
10 g = givens(10); % gravity [m/s^2]
11 alpha = givens(1); % [N/(m/s)^2]
12 eta = givens(2); % [N/(rad/s)^2]
13 I_x = givens(3); % moment of inertia in the x direction [kg m^2]
14 I_y = givens(4); % moment of inertia in the y direction [kg m^2]
15 I_z = givens(5); % moment of inertia in the z direction [kg m^2]
16 k_1 = givens(11);
17 k_2 = givens(12);
18 k_3 = givens(13);
19 k_4 = givens(14);
20 k_5 = givens(15);
21 k_6 = givens(16);
22
23 % define conditions vector
24 deltau = Conditions(1); % inertial velocity in the u direction in body coordinates [m/s]
25 deltav = Conditions(2); % inertial velocity in the v direction in body coordinates [m/s]
26 deltar = Conditions(3); % inertial velocity in the w direction in body coordinates [m/s]
27 deltap = Conditions(4); % inertial angular velocity in the p direction in body coordinates [
28     rad/s]
29 deltaq = Conditions(5); % inertial angular velocity in the q direction in body coordinates [
30     rad/s]
31 deltar = Conditions(6); % inertial angular velocity in the r direction in body coordinates [
32     rad/s]
33 deltaphi = Conditions(7); % bank [rad]
34 deltatheta = Conditions(8); % elevation [rad]
35 psi = Conditions(9); % azimuth [rad]
36 x_E = Conditions(10); % position vector in the x direction in inertial coordinates [m]
37 y_E = Conditions(11); % position vector in the y direction in inertial coordinates [m]
38 z_E = Conditions(12); % position vector in the z direction in inertial coordinates [m]
39 f1 = Force(1); % trim force exerted by motor 1 [N]
40 f2 = Force(2); % trim force exerted by motor 1 [N]
41 f3 = Force(3); % trim force exerted by motor 1 [N]
42 f4 = Force(4); % trim force exerted by motor 1 [N]
43 if (2.74<t) && (t<4.74)
44     deltavr = 1;
45     deltaur = 1;
46 else
47     deltavr = 0;
48     deltaur = 0;

```

```

46    end
47    % Aerodynamic/Control Moments and Forces
48    Laero = - alpha^2 * deltap^2 * sign(deltap); % p component of the aerodynamic moments
49    Maero = - alpha^2 * deltaq^2 * sign(deltaq); % q component of the aerodynamic moments
50    Naero = - alpha^2 * deltar^2 * sign(deltar); % w component of the aerodynamic moments
51    Lcontrol = ((f1 + f2) - (f3 + f4)) * R; % p component of the control moments
52    Mcontrol = ((f3 + f2) - (f1 + f4)) * R; % q component of the control moments
53    Ncontrol = -0.004*deltar; % w component of the aerodynamic moments
54    deltaL = Laero + Lcontrol; % L moment sum (abt x axis)
55    deltaM = Maero + Mcontrol; % M moment sum (abt y axis)
56    deltaN = Naero + Ncontrol; % N moment sum (abt Z axis)
57
58    if t > 2 && t < 2.5
59        deltaL = deltaL + Disturb(2); % Disturbed L moment (abt x axis)
60        deltaM = deltaM + Disturb(1); % Disturbed M moment (abt y axis)
61        deltaN = deltaN + Disturb(3); % Disturbed N moment (abt z axis)
62    end
63    Xaero = - eta^2 * deltau^2 * sign(deltau); % x component of aerodynamic force
64    Yaero = - eta^2 * deltav^2 * sign(deltav); % y component of aerodynamic force
65    Zaero = - eta^2 * deltar^2 * sign(deltar); % z component of aerodynamic force
66    Xcontrol = 0; % x control
67    Ycontrol = 0; % y control
68    Zcontrol = -sum(Force); % gravitational force counteraction
69    deltaX = Xaero + Xcontrol; % sum of x forces
70    deltaY = Yaero + Ycontrol; % sum of y forces
71
72
73    deltaZ = 0; % sum of z forces
74    deltapdot = ((-k_1*deltap)/I_x) -((k_2*deltaphi)/I_x -((k_2*k_3)*(deltavr-deltav))/I_x ); % roll rate derrivative
75    deltaqdot = ((-k_4*deltaq)/I_y) -((k_5*deltatheta)/I_y -((k_5*k_6)*(deltaur-deltau))/I_y ); % pitch rate derrivative
76
77    deltardot = deltaN/I_z; % yaw rate derrivative
78
79    dOmega_bdt = [deltapdot, deltaqdot, deltardot]';
80    deltaudot = -g*deltatheta; % 4.7,1 acceleration in the x axis
81    deltavdot = g*deltaphi; % acceleration in the y axis
82    deltaridot = deltaZ/m; % acceleration in the z axis
83    dVbdt = [deltaudot, deltavdot, deltaridot]';
84    xdot = deltau;
85    ydot = deltav;
86    zdot = deltar;
87    dVEdt = [xdot, ydot, zdot]';
88    phidot = deltap; % bank roc (pg 104)
89    thetadot = deltaq; % elevation roc (pg 104)
90    psidot = deltar; % azimuth roc (pg 104)
91    dEuldt = [phidot, thetadot, psidot]';
92    dydt = [dVbdt; dOmega_bdt; dEuldt; dVEdt];
93
94
95
96
97 end

```

### linearized feedback control ODE function with $\Delta u_R^E$ varying

```

1 % linear ODE Function
2 %Benjiman Smith
3 %02/13/2020
4 function dydt = Specs2LB5LCU(t, Conditions, Force, Disturb, givens)
5 % define givens
6 m = givens(6); % mass of the drone [kg]
7 r = givens(7); % body to motor distance [m]
8 k = givens(8); % [Nm/N]
9 R = givens(9); % [m]
10 g = givens(10); % gravity [m/s^2]

```

```

11 alpha = givens(1); % [N/(m/s)^2]
12 eta = givens(2); % [N/(rad/s)^2]
13 I_x = givens(3); % moment of inertia in the x direction [kg m^2]
14 I_y = givens(4); % moment of inertia in the y direction [kg m^2]
15 I_z = givens(5); % moment of inertia in the z direction [kg m^2]
16 k_1 = givens(11);
17 k_2 = givens(12);
18 k_3 = givens(13);
19 k_4 = givens(14);
20 k_5 = givens(15);
21 k_6 = givens(16);
22
23 % define conditions vector
24 deltau = Conditions(1); % inertial velocity in the u direction in body coordinates [m/s]
25 deltav = Conditions(2); % inertial velocity in the v direction in body coordinates [m/s]
26 deltaw = Conditions(3); % inertial velocity in the w direction in body coordinates [m/s]
27 deltap = Conditions(4); % inertial angular velocity in the p direction in body coordinates [
28     rad/s]
29 deltaq = Conditions(5); % inertial angular velocity in the q direction in body coordinates [
30     rad/s]
31 deltar = Conditions(6); % inertial angular velocity in the r direction in body coordinates [
32     rad/s]
33 deltaphi = Conditions(7); % bank [rad]
34 deltatheta = Conditions(8); % elevation [rad]
35 psi = Conditions(9); % azimuth [rad]
36 x_E = Conditions(10); % position vector in the x direction in inertial coordinates [m]
37 y_E = Conditions(11); % position vector in the y direction in inertial coordinates [m]
38 z_E = Conditions(12); % position vector in the z direction in inertial coordinates [m]
39 f1 = Force(1); % trim force exerted by motor 1 [N]
40 f2 = Force(2); % trim force exerted by motor 1 [N]
41 f3 = Force(3); % trim force exerted by motor 1 [N]
42 f4 = Force(4); % trim force exerted by motor 1 [N]
43 if t <= 2
44     deltavr = 0;
45     deltaur = 0.5;
46 else
47     deltavr = 0;
48     deltaur = 0;
49 end
50 % Aerodynamic/Control Moments and Forces
51 Laero = -alpha^2 * deltap^2 * sign(deltap); % p component of the aerodynamic moments
52 Maero = -alpha^2 * deltaq^2 * sign(deltaq); % q component of the aerodynamic moments
53 Naero = -alpha^2 * deltar^2 * sign(deltar); % w component of the aerodynamic moments
54 Lcontrol = ((f1 + f2) - (f3 + f4)) * R; % p component of the control moments
55 Mcontrol = ((f3 + f2) - (f1 + f4)) * R; % q component of the control moments
56 Ncontrol = -0.004 * deltar; % w component of the aerodynamic moments
57 deltaL = Laero + Lcontrol; % L moment sum (abt x axis)
58 deltaM = Maero + Mcontrol; % M moment sum (abt y axis)
59 deltaN = Naero + Ncontrol; % N moment sum (abt Z axis)
60
61 if t > 2 && t < 2.5
62     deltaL = deltaL + Disturb(2); % Disturbed L moment (abt x axis)
63     deltaM = deltaM + Disturb(1); % Disturbed M moment (abt y axis)
64     deltaN = deltaN + Disturb(3); % Disturbed N moment (abt z axis)
65 end
66 Xaero = -eta^2 * deltau^2 * sign(deltau); % x component of aerodynamic force
67 Yaero = -eta^2 * deltav^2 * sign(deltav); % y component of aerodynamic force
68 Zaero = -eta^2 * deltaw^2 * sign(deltaw); % z component of aerodynamic force
69 Xcontrol = 0; % x control
70 Ycontrol = 0; % y control
71 Zcontrol = -sum(Force); % gravitational force counteraction
72 deltaX = Xaero + Xcontrol; % sum of x forces
73 deltaY = Yaero + Ycontrol; % sum of y forces
74
75 deltaZ = 0; % sum of z forces
76 deltapdot = ((-k_1 * deltap) / I_x) - ((k_2 * deltaphi) / I_x) - ((k_2 * k_3) * (deltavr - deltav)) / I_x; %
    roll rate derivative

```

```

75 deltaqdot = ((-k_4*deltaq)/I_y) - ((k_5*deltatheta)/I_y - ((k_5*k_6)*(deltaur-deltau))/I_y); %
    pitch rate derrivative
76
77
78 deltardot = deltaN/I_z; % yaw rate derrivative
79
80 dOmega_bdt = [deltapdot, deltaqdot, deltardot]';
81 deltaudot = -g*deltatheta; % 4.7,1 acceleration in the x axis
82 deltavdot = g*deltaphi; % acceleration in the y axis
83 deltawdot = deltaZ/m; % acceleration in the z axis
84 dVbdt = [deltaudot, deltavdot, deltawdot]';
85 xdot = deltau;
86 ydot = deltav;
87 zdot = deltaw;
88 dVEdt = [xdot, ydot, zdot]';
89 phidot = deltap; % bank roc (pg 104)
90 thetadot = deltaq; % elevation roc (pg 104)
91 psidot = deltar; % azimuth roc (pg 104)
92 dEuldt = [phidot, thetadot, psidot]';
93 dydt = [dVbdt; dOmega_bdt; dEuldt; dVEdt];
94
95
96
97 end

```

### linearized feedback control ODE function with $\Delta v_R^E$ varying

```

1 % linear ODE Function
2 %Benjiman Smith
3 %02/13/2020
4 function dydt = Specs2LB5LCV(t, Conditions, Force, Disturb, givens)
5 % define givens
6 m = givens(6); % mass of the drone [kg]
7 r = givens(7); % body to motor distance [m]
8 k = givens(8); % [Nm/N]
9 R = givens(9); % [m]
10 g = givens(10); % gravity [m/s^2]
11 alpha = givens(1); % [N/(m/s)^2]
12 eta = givens(2); % [N/(rad/s)^2]
13 I_x = givens(3); % moment of inertia in the x direction [kg m^2]
14 I_y = givens(4); % moment of inertia in the y direction [kg m^2]
15 I_z = givens(5); % moment of inertia in the z direction [kg m^2]
16 k_1 = givens(11);
17 k_2 = givens(12);
18 k_3 = givens(13);
19 k_4 = givens(14);
20 k_5 = givens(15);
21 k_6 = givens(16);
22
23 % define conditions vector
24 deltau = Conditions(1); % inertial velocity in the u direction in body coordinates [m/s]
25 deltav = Conditions(2); % inertial velocity in the v direction in body coordinates [m/s]
26 deltaw = Conditions(3); % inertial velocity in the w direction in body coordinates [m/s]
27 deltap = Conditions(4); % inertial angular velocity in the p direction in body coordinates [
    rad/s]
28 deltaq = Conditions(5); % inertial angular velocity in the q direction in body coordinates [
    rad/s]
29 deltar = Conditions(6); % inertial angular velocity in the r direction in body coordinates [
    rad/s]
30 deltaphi = Conditions(7); % bank [rad]
31 deltatheta = Conditions(8); % elevation [rad]
32 psi = Conditions(9); % azimuth [rad]
33 x_E = Conditions(10); % position vector in the x direction in inertial coordinates [m]
34 y_E = Conditions(11); % position vector in the y direction in inertial coordinates [m]
35 z_E = Conditions(12); % position vector in the z direction in inertial coordinates [m]
36 f1 = Force(1); % trim force exerted by motor 1 [N]
37 f2 = Force(2); % trim force exerted by motor 1 [N]

```

```

38     f3 = Force(3); % trim force exerted by motor 1 [N]
39     f4 = Force(4); % trim force exerted by motor 1 [N]
40     if t <= 2
41         deltarv = .5; % 2.1 m/s
42         deltaur = 0;
43     else
44         deltarv = 0;
45         deltaur= 0;
46     end
47 % Aerodynamic/Control Moments and Forces
48 Laero = - alpha^2 * deltap^2 * sign(deltap); % p component of the aerodynamic moments
49 Maero = - alpha^2 * deltaq^2 * sign(deltaq); % q component of the aerodynamic moments
50 Naero = - alpha^2 * deltar^2 * sign(deltar); % w component of the aerodynamic moments
51 Lcontrol = ((f1 + f2) - (f3 + f4)) * R; % p component of the control moments
52 Mcontrol = ((f3 + f2) - (f1 + f4)) * R; % q component of the control moments
53 Ncontrol = -0.004*deltar; % w component of the aerodynamic moments
54 deltaL = Laero + Lcontrol; % L moment sum (abt x axis)
55 deltaM = Maero + Mcontrol; % M moment sum (abt y axis)
56 deltaN = Naero + Ncontrol; % N moment sum (abt Z axis)
57
58 if t > 2 && t < 2.5
59     deltaL = deltaL + Disturb(2); % Disturbed L moment (abt x axis)
60     deltaM = deltaM + Disturb(1); % Disturbed M moment (abt y axis)
61     deltaN = deltaN + Disturb(3); % Disturbed N moment (abt z axis)
62 end
63 Xaero = - eta^2 * deltau^2 * sign(deltau); % x component of aerodynamic force
64 Yaero = - eta^2 * deltarv^2 * sign(deltarv); % y component of aerodynamic force
65 Zaero = - eta^2 * deltaq^2 * sign(deltaq); % z component of aerodynamic force
66 Xcontrol = 0; % x control
67 Ycontrol = 0; % y control
68 Zcontrol = -sum(Force); % gravitational force counteraction
69 deltaX = Xaero + Xcontrol; % sum of x forces
70 deltaY = Yaero + Ycontrol; % sum of y forces
71
72
73 deltaZ = 0; % sum of z forces
74 deltapdot = ((-k_1*deltap)/I_x) -((k_2*deltaphi)/I_x -((k_2*k_3)*(deltavr-deltav))/I_x ); % roll rate derivative
75 deltaqdot = ((-k_4*deltaq)/I_y) -((k_5*deltatheta)/I_y -((k_5*k_6)*(deltaur-deltau))/I_y ); % pitch rate derivative
76
77
78 deltardot = deltaN/I_z; % yaw rate derivative
79
80 dOmega_bdt = [deltapdot, deltaqdot, deltardot]';
81 deltaudot = -g*deltatheta; % 4.7,1 acceleration in the x axis
82 deltarvdot = g*deltaphi; % acceleration in the y axis
83 deltarwdot = deltaZ/m; % acceleration in the z axis
84 dVbdt = [deltaudot, deltarvdot, deltarwdot]';
85 xdot = deltau;
86 ydot = deltarv;
87 zdot = deltarw;
88 dVEdt = [xdot, ydot, zdot]';
89 phidot = deltap; % bank roc (pg 104)
90 thetadot = deltaq; % elevation roc (pg 104)
91 psidot = deltar; % azimuth roc (pg 104)
92 dEuldt = [phidot, thetadot, psidot]';
93 dydt = [dVbdt; dOmega_bdt; dEuldt; dVEdt];
94
95
96
97 end

```

## Stop Function

```

1 % This function is used by ODE45 in options to end the drones
2 % path after a certain tolerance of values is reached

```

```
3 %  
4 % Author: Benjiman Smith  
5 % Collaborators: E. Owen, I. Quezada  
6 % Date: 1/26/2020  
7 %  
8 function [value, isTerm, direction] = StopFnct(t, F)  
9 value = F(12);  
10 isTerm = 1;  
11 direction = [];  
12 end
```