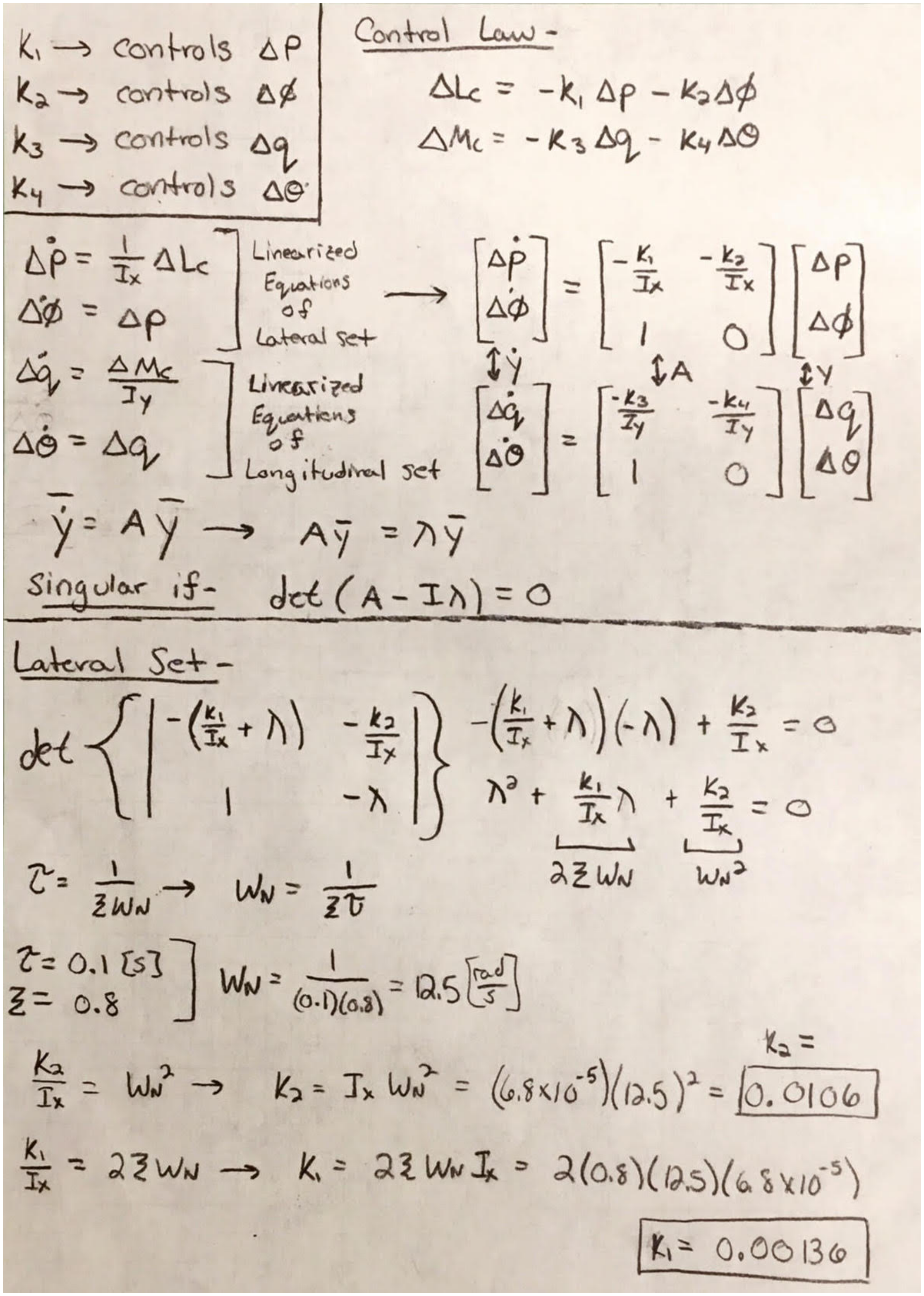
Quad Copter Elevation and Bank Attitude Closed Loop Linearized Response

Jack Lambert[*]
University of Colorado Boulder
Submitted 2/19/2018

This analysis looks at deriving feedback control gains about the linearized lateral and longitudinal dynamics of a Parrot Rolling Spider quad copter. The derived gains are then inputted into the quad copter and the state variables are measured utilizing sensors on the quad copter as the copter's steady state response is deviated using a force-full perturbation. A comparison is then made between this real-life data to a MATLAB simulation of the quad copter that transforms the copter's dynamics into first order differential equations that are solved over discrete time steps utilizing an adaptive 4th and 5th order Runge Kutta ODE solver.

[*]SID: 104414093

I. Feedback Control System for Linearized Model



$K_1 \rightarrow$ controls Δp
$K_2 \rightarrow$ controls $\Delta \phi$
$K_3 \rightarrow$ controls Δq
$K_4 \rightarrow$ controls $\Delta \Theta$

Control Law -

$$\Delta L_c = -K_1 \Delta p - K_2 \Delta \phi$$
$$\Delta M_c = -K_3 \Delta q - K_4 \Delta \Theta$$

$$\left.\begin{array}{l}\Delta \dot{p} = \frac{1}{I_x}\Delta L_c \\ \Delta \dot{\phi} = \Delta p\end{array}\right]$$ Linearized Equations of Lateral set $\longrightarrow$

$$\begin{bmatrix}\Delta \dot{p} \\ \Delta \dot{\phi}\end{bmatrix} = \begin{bmatrix}-\frac{K_1}{I_x} & -\frac{K_2}{I_x} \\ 1 & 0\end{bmatrix}\begin{bmatrix}\Delta p \\ \Delta \phi\end{bmatrix}$$

$\updownarrow \dot{y}$ $\quad$ $\updownarrow A$ $\quad$ $\updownarrow y$

$$\left.\begin{array}{l}\Delta \dot{q} = \frac{\Delta M_c}{I_y} \\ \Delta \dot{\Theta} = \Delta q\end{array}\right]$$ Linearized Equations of Longitudinal set

$$\begin{bmatrix}\Delta \dot{q} \\ \Delta \dot{\Theta}\end{bmatrix} = \begin{bmatrix}-\frac{K_3}{I_y} & -\frac{K_4}{I_y} \\ 1 & 0\end{bmatrix}\begin{bmatrix}\Delta q \\ \Delta \Theta\end{bmatrix}$$

$$\bar{\dot{y}} = A\bar{y} \longrightarrow A\bar{y} = \lambda \bar{y}$$

Singular if- $\det(A - I\lambda) = 0$

Lateral Set -

$$\det\left\{\begin{vmatrix}-\left(\frac{K_1}{I_x}+\lambda\right) & -\frac{K_2}{I_x} \\ 1 & -\lambda\end{vmatrix}\right\} \quad -\left(\frac{K_1}{I_x}+\lambda\right)(-\lambda) + \frac{K_2}{I_x} = 0$$

$$\lambda^2 + \underbrace{\frac{K_1}{I_x}}_{2\zeta W_N}\lambda + \underbrace{\frac{K_2}{I_x}}_{W_N^2} = 0$$

$$\tau = \frac{1}{\zeta W_N} \rightarrow W_N = \frac{1}{\zeta \tau}$$

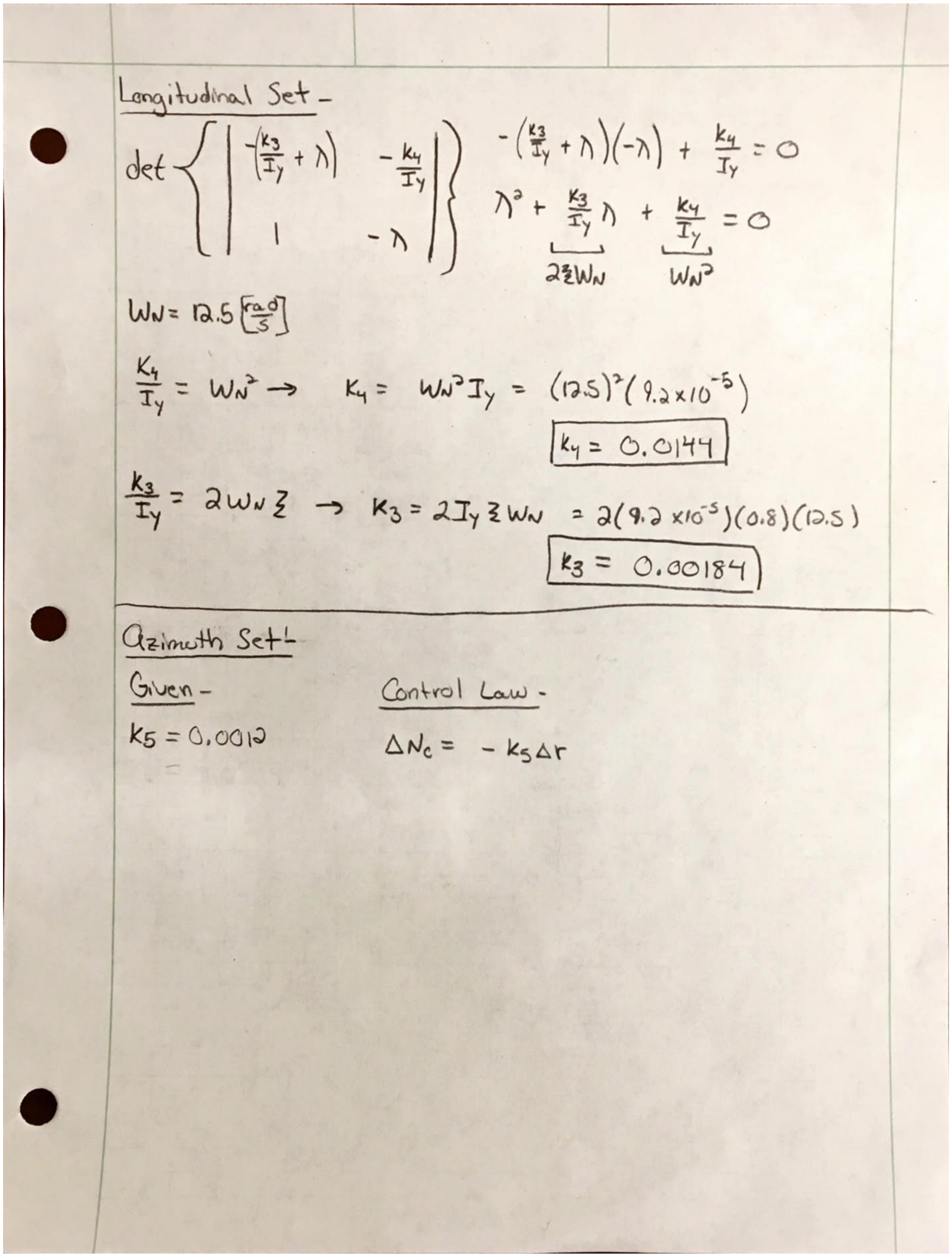
$$\left.\begin{array}{l}\tau = 0.1\,[s] \\ \zeta = 0.8\end{array}\right] \quad W_N = \frac{1}{(0.1)(0.8)} = 12.5\left[\frac{rad}{s}\right]$$

$$\frac{K_2}{I_x} = W_N^2 \rightarrow K_2 = I_x W_N^2 = (6.8\times10^{-5})(12.5)^2 = \overset{K_2 =}{\boxed{0.0106}}$$

$$\frac{K_1}{I_x} = 2\zeta W_N \rightarrow K_1 = 2\zeta W_N I_x = 2(0.8)(12.5)(6.8\times10^{-5})$$

$$\boxed{K_1 = 0.00136}$$

Figure 1. Derivation Quad Copter Stability about Elevation and Bank Attitude



Longitudinal Set -

$$\det\left\{\begin{vmatrix} -\left(\frac{k_3}{I_y}+\lambda\right) & -\frac{k_4}{I_y} \\ 1 & -\lambda \end{vmatrix}\right\} \qquad -\left(\frac{k_3}{I_y}+\lambda\right)(-\lambda) + \frac{k_4}{I_y} = 0$$

$$\lambda^2 + \underbrace{\frac{k_3}{I_y}}_{2\zeta W_N}\lambda + \underbrace{\frac{k_4}{I_y}}_{W_N^2} = 0$$

$W_N = 12.5 \left[\frac{rad}{s}\right]$

$$\frac{K_4}{I_y} = W_N^2 \rightarrow \quad K_4 = W_N^2 I_y = (12.5)^2(9.2\times10^{-5})$$

$$\boxed{k_4 = 0.0144}$$

$$\frac{k_3}{I_y} = 2W_N\zeta \rightarrow K_3 = 2I_y\zeta W_N = 2(9.2\times10^{-5})(0.8)(12.5)$$

$$\boxed{k_3 = 0.00184}$$

Azimuth Set -

Given -

$K_5 = 0.0012$

Control Law -

$\Delta N_c = -k_5\Delta r$

Figure 2. Derivation Quad Copter Stability about Elevation and Bank Attitude

The derivations for the feedback control system about the lateral and longitudinal dynamics were done by the coupled sets. The "lateral set", was comprised of the variables of roll rate and the Euler angle ϕ which are dependent on each other and their corresponding gains were found using the control law and solving for the gains that satisfied having a time constant of $\tau = 0.1[s]$ with a damping ration of $\zeta = 0.8$. The same was done for "longitudinal set", which was comprised of the coupled variables of pitch rate and the Euler angle θ.

II. Simulated - Closed Loop Linearized Response

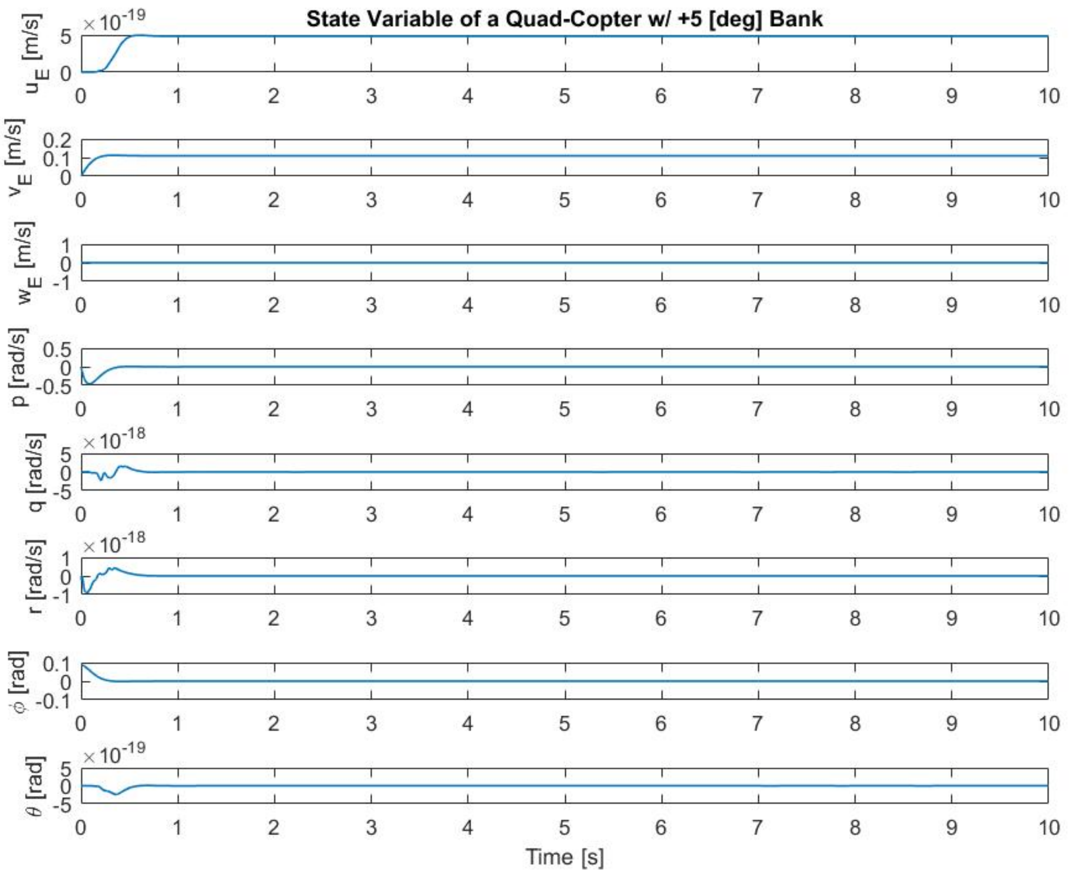


Figure 3. Deviation of $+5°$ in Bank

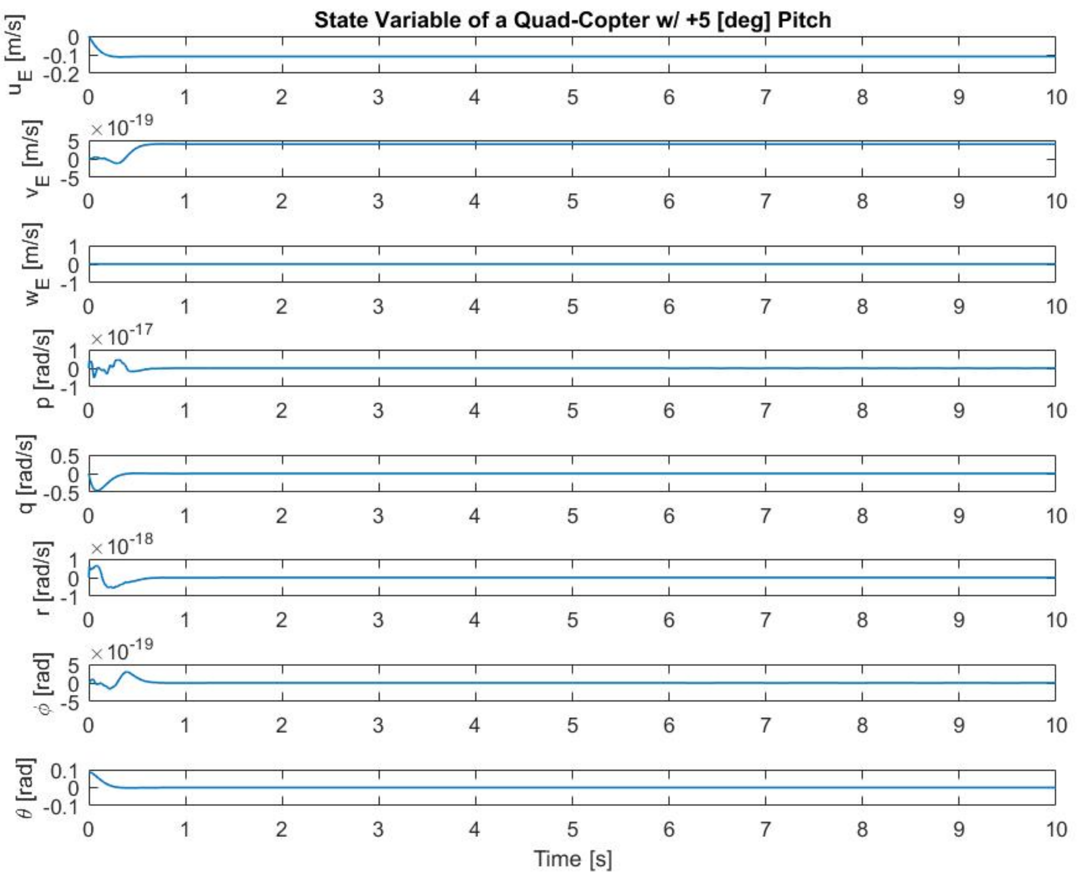


Figure 4. Deviation of $+5^{\circ}$ in Pitch

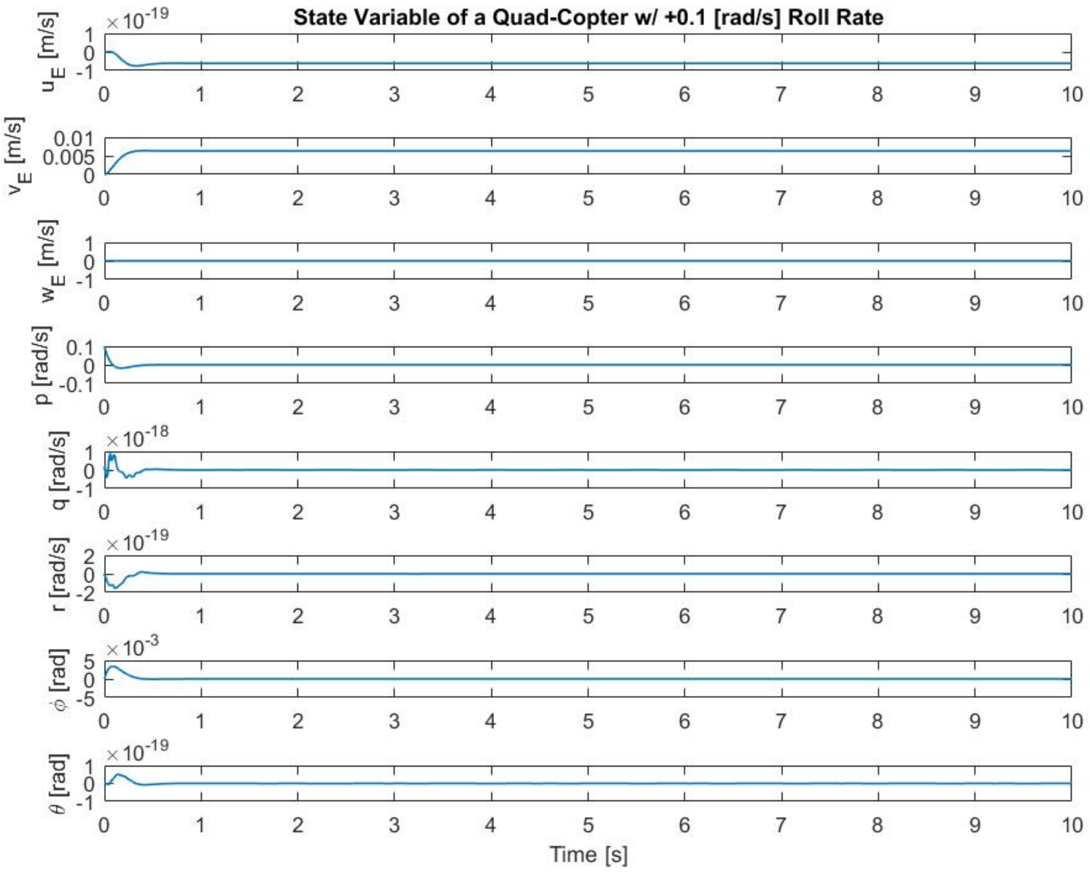


Figure 5. Deviation of $+0.1[\frac{rad}{s}]$ in Roll Rate

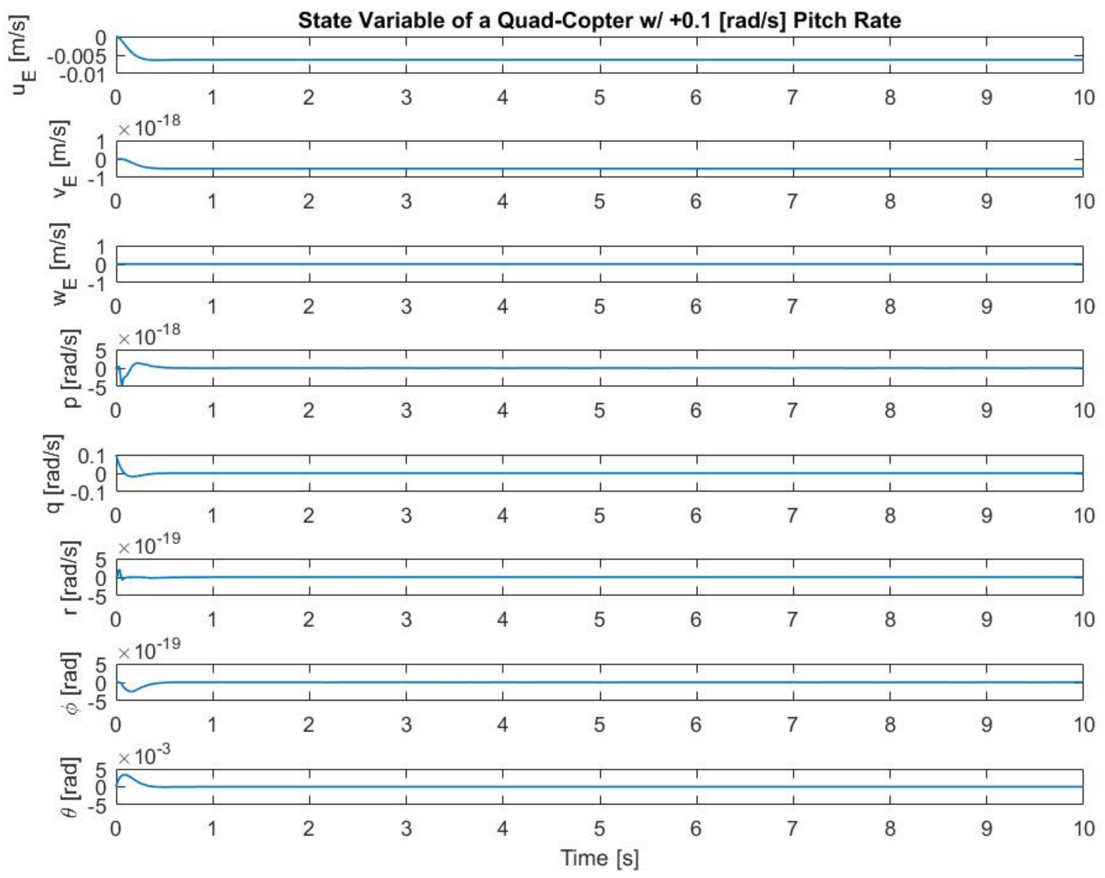


Figure 6. Deviation of $+0.1[\frac{rad}{s}]$ in Pitch Rate

The linearized response of the quad copter reacts in the way we expect for the different deviations in the attitude and rotation. The coupled sets in the linearized model remained coupled after the feedback controls were implemented. This is shown in the figures for the linearized model as the variations in the bank angle resulted in changes in the roll rate and in the y-component of the body fixed coordinate representation of inertial velocity. When the roll rate was varied, changes in bank angle and y-component of the body fixed coordinate representation of inertial velocity were seen. This is due to these three variables (V_E, ϕ, and p) being coupled as the "lateral set" in the linearized model for a steady hover state. The other coupled set, the "longitudinal set", also remain coupled. This can be depicted in the plots for the variations in either θ or q, resulting in changes in the resulting coupled variables of θ, q, and U_E. The resulting system is also noticed to now be neutrally stable. Deviations in the attitude and rotational nature of the body frame, result in a feedback response until the perturbations are resolved back to a steady hovering state. The reason they are only neutrally stable and not completely stable is due to the fact that the quad copter continues to translate after the attitude and rotation are resolved back to their trim states for steady hovering flight.

III. Simulated - Linearized and Non-Linearized Response Comparison

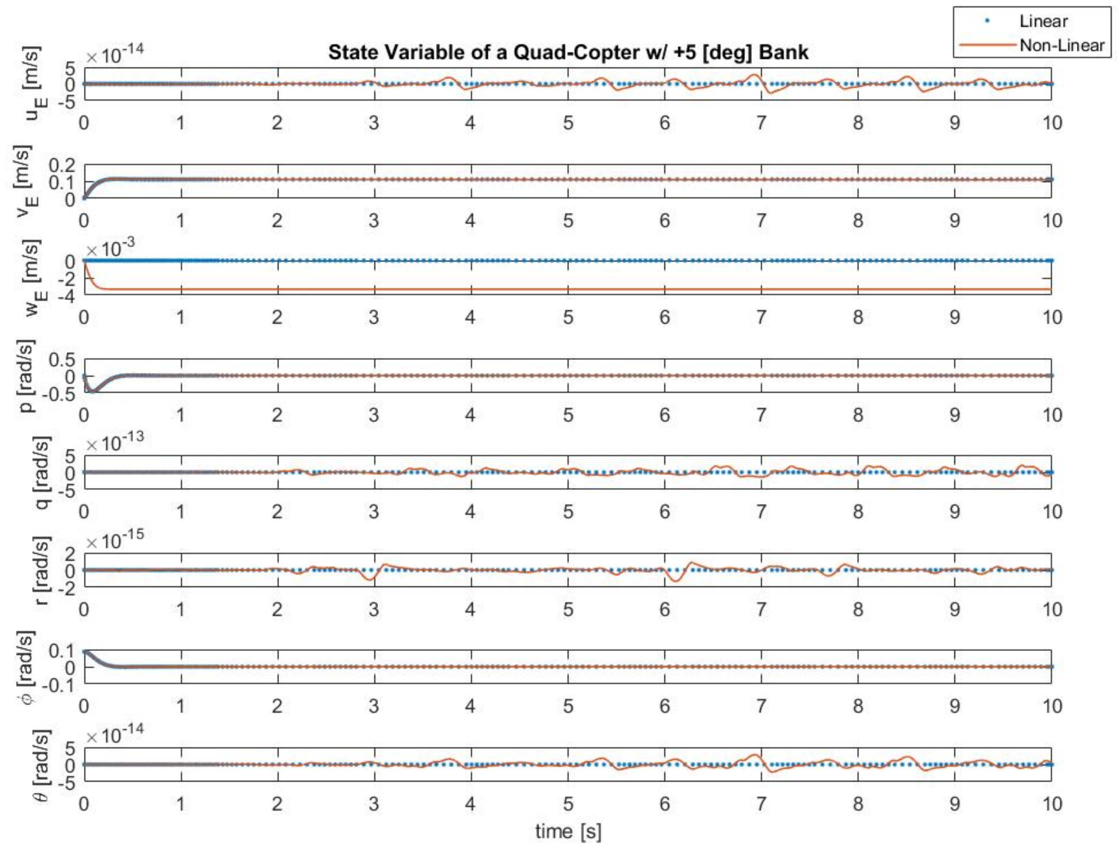


Figure 7. Deviation of $+5^\circ$ in Bank

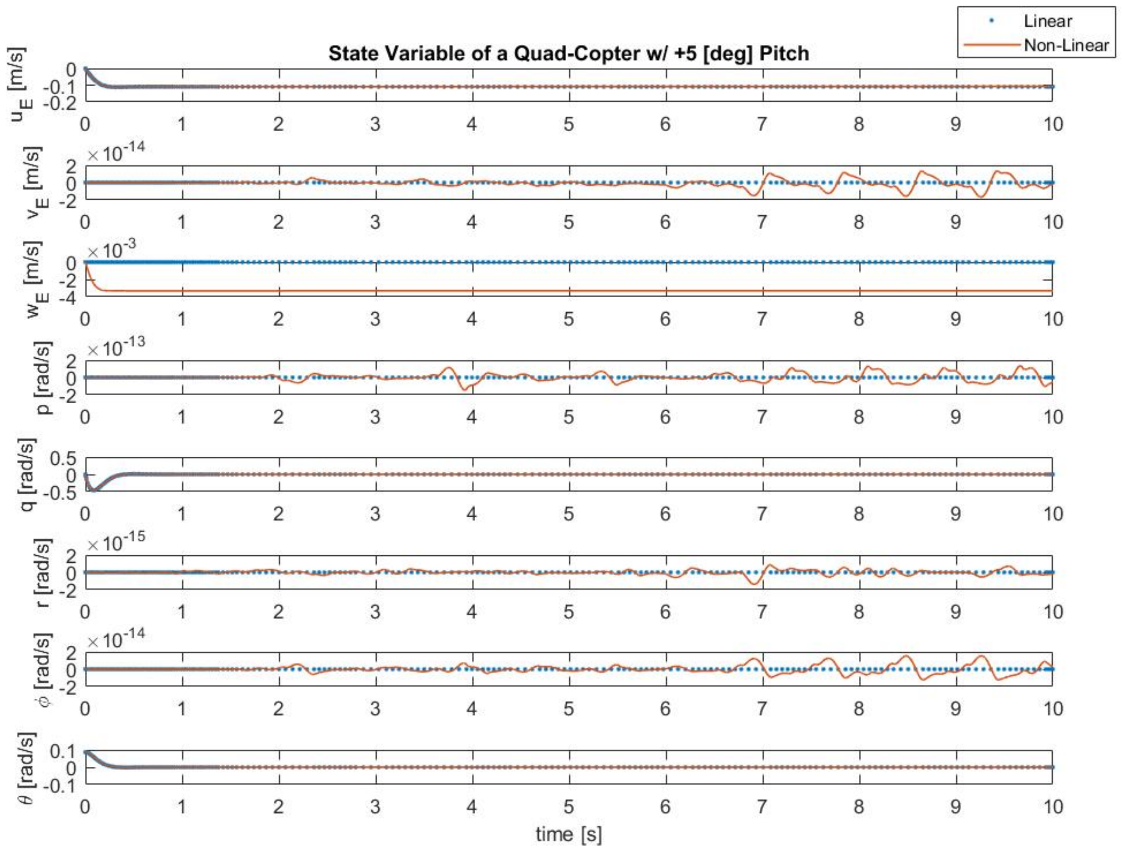


Figure 8. Deviation of $+5^{\circ}$ in Pitch

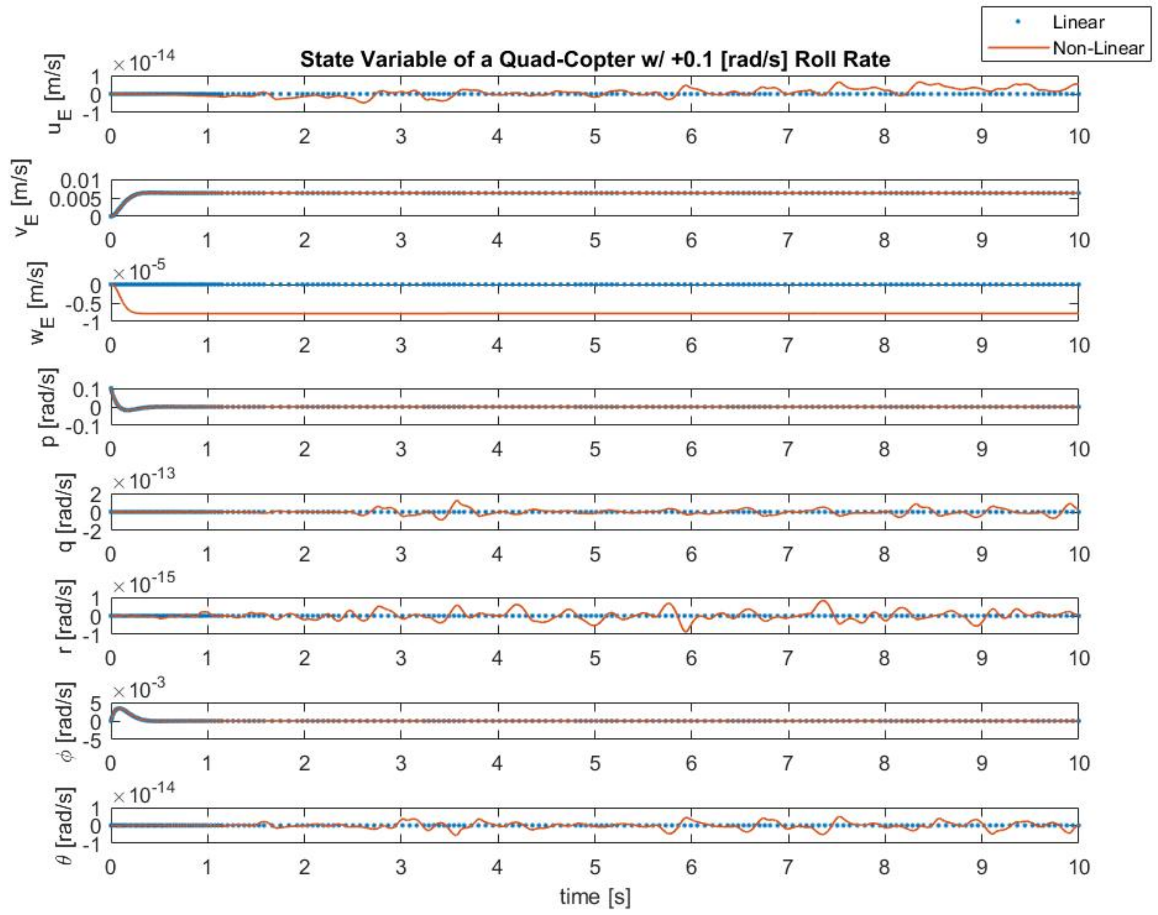


Figure 9. Deviation of $+0.1[\frac{rad}{s}]$ in Roll Rate

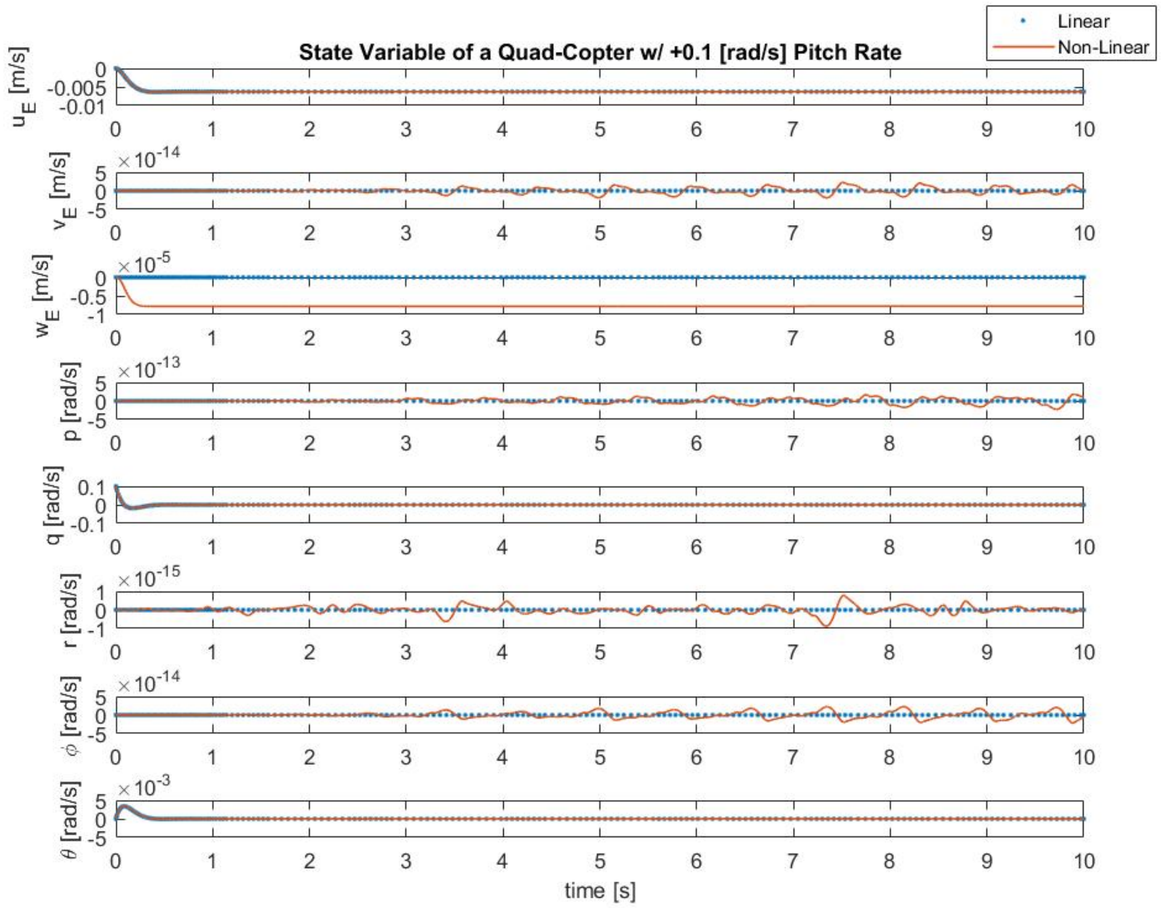


Figure 10. Deviation of $+0.1[\frac{rad}{s}]$ in Pitch Rate

When implementing the linearized feedback control design to the non-linear model, there were very slight differences in response. While all the state variables remained with in the linearized model's result, there were slight variations in the nosiness of the state variables. Granted the largest variation between the linearized and non-linearized response for any of the state variables was on the order of magnitude of 10^{-3}, there were still small variations. These small variations most likely occur due to the coupled nature of the variables as they change, causing small changes in each other on a scale that the precision of ODE45 cannot account for. It should also be noted that even though the linearized model does not directly account for aerodynamic effects, the results show to be nearly unaffected.

IV. Rolling Spider Experimental Data w/ Implemented Gains

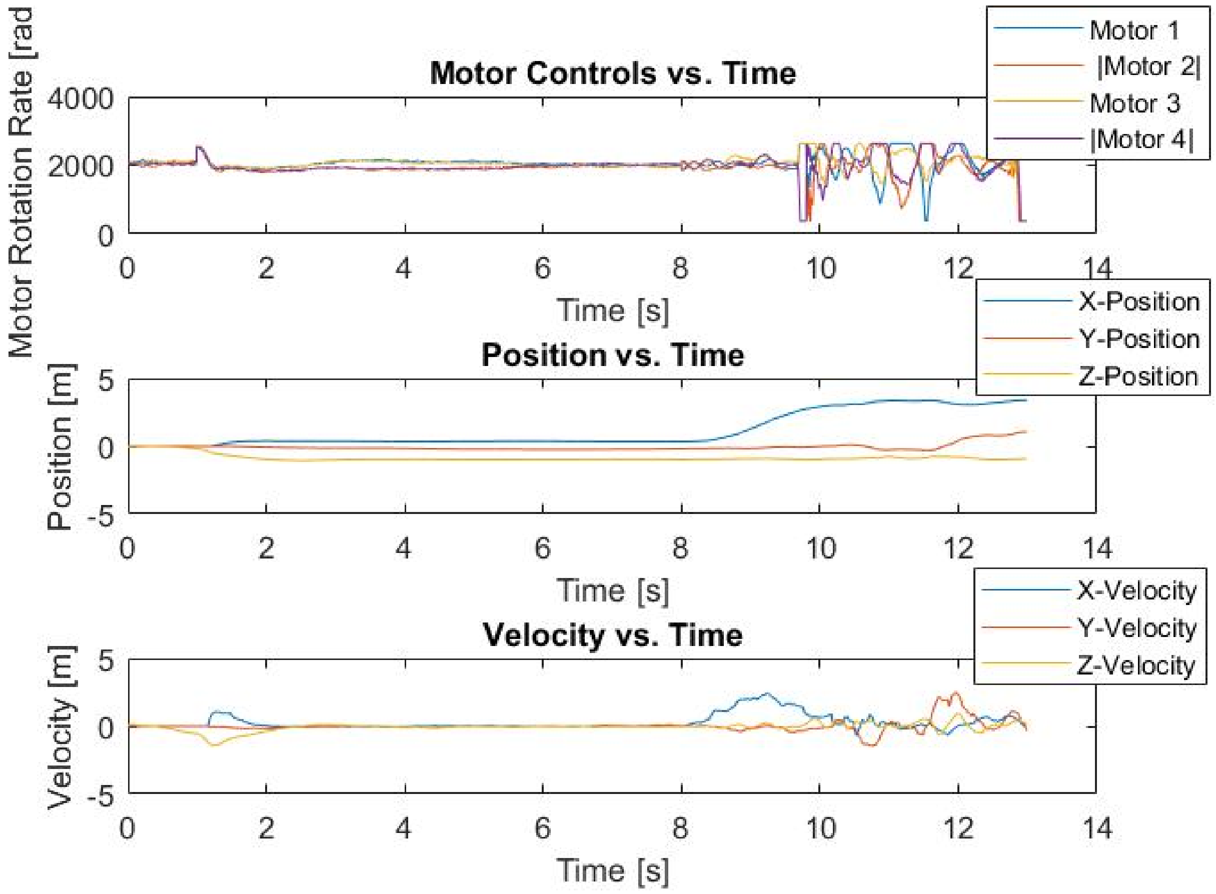


Figure 11. Translation of Rolling Spider

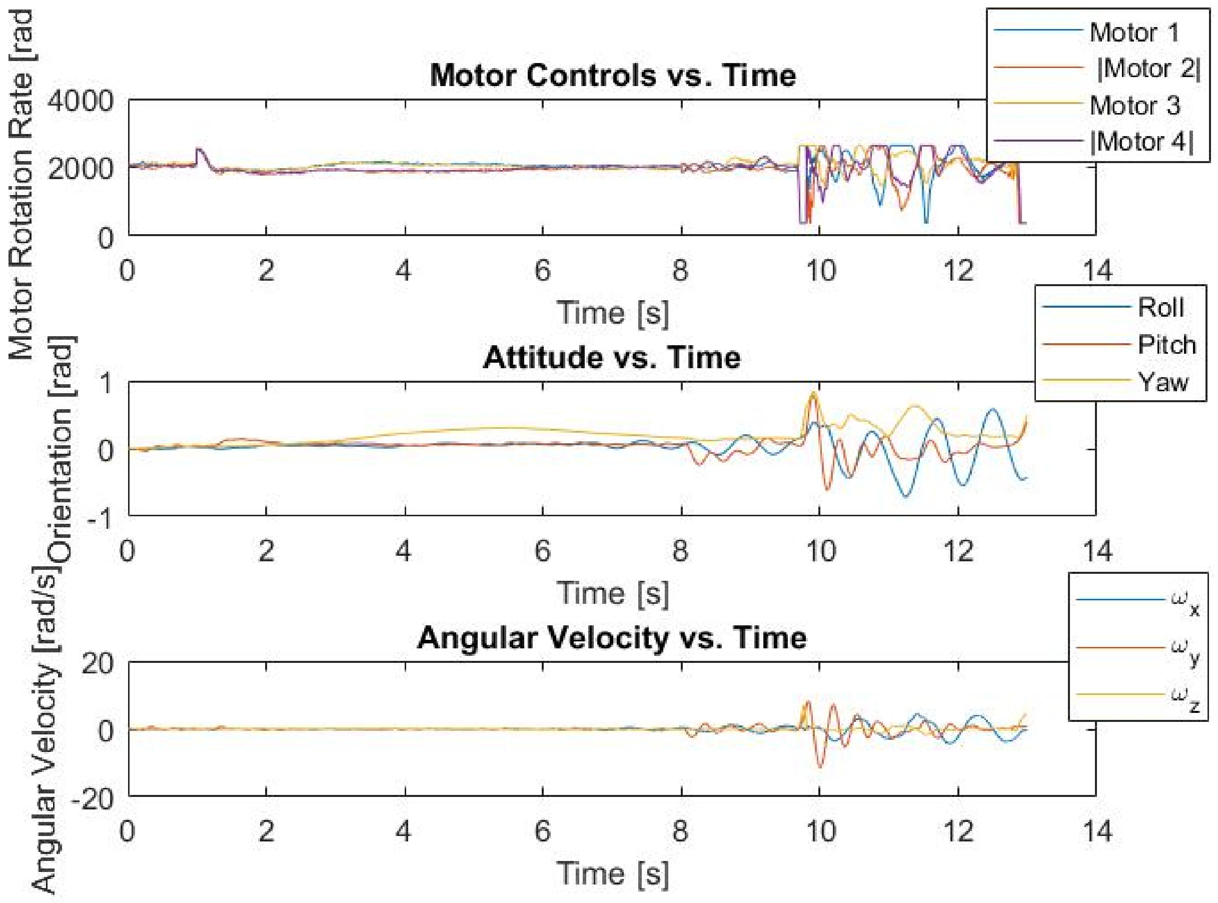


Figure 12. Attitude of Rolling Spider

When comparing the data between the experimental model and the simulations from MATLAB, it is evident that the response of the real life quad copter to deviations in bank and elevation are to emulate. When the quad copter was given a change in the elevation angle, and the gains we derived are applied, the copter corrects back and forth until it eventually hits the net. If the net were larger, the copter would eventually go back to a fixed attitude, with some translation. This was partially modeled by the experimental data as it is noticed the copter corrects itself as it translates. Before hitting the net, the copter is noticed to have angular velocities that slowly emanate back to zero. This shows that while the experimental case varies from the simulations, the overall trends remain apparent.

A. Appendix A - MATLAB Code

1. Main Function for ODE45 to Call for Linear Model

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Author: Jack Lambert

% Purpose: This function gives ODE45 all the differntial equations to
% iterate through over each time step for the Linear Model
% Date Modefied: 2/18/18
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [dydt] = Linear(t,y)
%% Constants
mass = 68/1000; % [kg]
L_arm = 6/100; % [m]
```

```
eta = 1*10^(-3); % Aerodynamic Coefficient for drag [N /(m/s)^2]
zeta = 3*10^(-3);  % Aerodynamic Coefficient for drag [N /(m/s)^2]
alpha = 2*10^(-6); % Aerodynamic Coefficient for drag [N /(rad/s)^2]
beta = 1*10^(-6); % Aerodynamic Coefficient for drag [N /(rad/s)^2]
Ix = 6.8*10^(-5); % MOI about x-axis [kg*m^2]
Iy = 9.2*10^(-5); % MOI about x-axis [kg*m^2]
Iz = 1.35*10^(-4); % MOI about x-axis [kg*m^2]
R = sqrt(2)/2*L_arm; % Distance to COG [m]
k = 0.0024; % [m]
g = 9.81; % [m/s^2]
%% Derivatives to be Integrated
% Translational Motion
delta_xE = y(1); % N - location
delta_yE = y(2); % E - location
delta_zE = y(3); % -D - location
delta_u = y(4); % u - component of velocity
delta_v = y(5); % v compenent of velocity
delta_w = y(6); % w component of velocity
% Rotational Motion
delta_phi = y(7); % Bank [rad]
delta_theta = y(8); % Pitch [rad]
delta_psi = y(9); % Azimuth [rad]
delta_p = y(10); % Roll Rate [rad/s]
delta_q = y(11); % Pitch Rate [rad/s]
delta_r = y(12); % Yaw Rate [rad/s]
%% Adding Feedback Control to Attitude

% Control Constants
k1 = 0.00136; % Derivative control constant for pitch Rate
k2 = 0.0106; % Proportional control constant for phi
k3 = 0.00184; % Derivative control constant for roll Rate
k4 = 0.0144;  % Proportional control constant for theta
k5 = 0.0012; % Derivative Control of yaw rate

% Controls needed for Rotation Control
bL = [(-k1*delta_p - k2*delta_phi), (-k3*delta_q - k4*delta_theta), (-k5*delta_r), 0];

% Geometry of quadcopter
AL = [-R R R -R;
    -R -R R R;
    -k k -k k;
    -1 -1 -1 -1];

% Computing forces based on Control gains and Forces
delta_f = AL\bL';
%% Translation Equations

% Transformation matrix expanded for position in inertial to position in
% body
dydt(1) = delta_u*cos(delta_theta)*cos(delta_psi)+ delta_v*(sin(delta_phi)...
    *sin(delta_theta)*cos(delta_psi)-cos(delta_phi)*sin(delta_psi))...
    + delta_w*(cos(delta_phi)*sin(delta_theta)*cos(delta_psi)+...
    sin(delta_phi)*sin(delta_psi)); % X_E [m]
dydt(2) = delta_u*cos(delta_theta)*sin(delta_psi)+ delta_v*(sin(delta_phi)...
    *sin(delta_theta)*sin(delta_psi)+cos(delta_phi)*cos(delta_psi))...
    + delta_w*(cos(delta_phi)*sin(delta_theta)*sin(delta_psi)-...
    sin(delta_phi)*cos(delta_psi)); % Y_E [m]
dydt(3) = -delta_u*sin(delta_theta)+delta_v*sin(delta_phi)*cos(delta_theta)...
    +delta_w*cos(delta_phi)*cos(delta_theta); % Z_E [m]

dydt(4) = -g*delta_theta; % u_E [m/s]
dydt(5) = g*delta_phi; % v_E [m/s]
dydt(6) = AL(4,:)*delta_f*(1/mass); % w_E [m/s]

%% Moments to Rotations
dydt(7) = delta_p; % Phi [rad]
dydt(8) = delta_q; % Theta [rad]
dydt(9) = delta_r; % Psi [rad]
dydt(10) = AL(1,:)*delta_f*(1/Ix); % p [rad/s]
dydt(11) = AL(2,:)*delta_f*(1/Iy); % q [rad/s]
dydt(12) = AL(3,:)*delta_f*(1/Iz); % r [rad/s]

dydt = dydt';
```

2. *Main Function for ODE45 to call for Non-Linear Model*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Author: Jack Lambert

% Purpose: This function gives ODE45 all the differntial equations to
% iterate through over each time step for the Non-Linear Model.
% Date Modefied: 2/18/18
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [dydt] = NonLinear(t,y)
%% Contstants
mass = 68/1000; % [kg]
L_arm = 6/100; % [m]
eta = 1*10^(-3); % Aerodynamic Coefficient for drag [N /(m/s)^2]
zeta = 3*10^(-3);  % Aerodynamic Coefficient for drag [N /(m/s)^2]
alpha = 2*10^(-6); % Aerodynamic Coefficient for drag [N /(rad/s)^2]
beta = 1*10^(-6); % Aerodynamic Coefficient for drag [N /(rad/s)^2]
Ix = 6.8*10^(-5); % MOI about x-axis [kg*m^2]
Iy = 9.2*10^(-5); % MOI about x-axis [kg*m^2]
Iz = 1.35*10^(-4); % MOI about x-axis [kg*m^2]
R = sqrt(2)/2*L_arm; % Distance to COG [m]
k = 0.0024; % [m]
g = 9.81; % [m/s^2]
%% Forces about each Motor
f1 = (mass*g)/4; % Force for steady Level Flight about Motor 1 [N]
f2 = (mass*g)/4; % Force for steady Level Flight about Motor 2 [N]
f3 = (mass*g)/4; % Force for steady Level Flight about Motor 3 [N]
f4 = (mass*g)/4; % Force for steady Level Flight about Motor 4 [N]
%% Derivatives to be Integrated
% Translational Motion
dx = y(1); % N - location
dy = y(2); % E - location
dz = y(3); % -D - location
u = y(4); % u - component of velocity
v = y(5); % v compenent of velocity
w = y(6); % w component of velocity
% Rotational Motion
phi = y(7); % Attitude Euler Angles
theta = y(8); % Attitude Euler Angles
psi = y(9); % Attitude Euler Angles
p = y(10); % Angular velocity about the x-axis [rad/s]
q = y(11); % Angular Velocity about the y-axis [rad/s]
r = y(12); % Angular Velocity about the z-axis [rad/s]

%% Adding Feedbck Control to Attitude

% Control Constants
k1 = 0.00136; % Derivative control constant for pitch Rate
k2 = 0.0106; % Proportional control constant for phi
k3 = 0.00184; % Derivative control constant for roll Rate
k4 = 0.0144;  % Proportional control constant for theta
k5 = 0.0012; % Derivative Control of yaw rate

% Controls needed for Rotation Control
b = [-k1*p - k2*phi, -k3* q- k4*theta, -k5*r, -mass*g];

% Geometry of quadcopter
A = [-R R R -R;
    -R -R R R;
    -k k -k k;
    -1 -1 -1 -1];
% Computing forces based on Control gains and Forces
```

```
uf = A\b';
%% Forces to Acceleration
% Aerodynimc Forces
A_c = [0 0 A(4,:)*uf]; % Control Forces
A_a = [-eta*u^2*sign(u) -eta*v^2*sign(v) -zeta*w^2*sign(w)]; % Aerodynamics Forces
A_b = A_c + A_a; % Combined Forces
% Kinematic Equations
dydt(4) = (A_b(1)-mass*g*sin(theta))/mass;
dydt(5) = (A_b(2)+mass*g*cos(theta)*sin(phi))/mass;
dydt(6) = (A_b(3)+mass*g*cos(theta)*cos(phi))/mass;

dydt(1) = u*cos(theta)*cos(psi)+ v*(sin(phi)*sin(theta)*cos(psi)-cos(phi)*sin(psi))...
    + w*(cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi));
dydt(2) = u*cos(theta)*sin(psi)+ v*(sin(phi)*sin(theta)*sin(psi)+cos(phi)*cos(psi))...
    + w*(cos(phi)*sin(theta)*sin(psi)-sin(phi)*cos(psi));
dydt(3) = -u*sin(theta)+v*sin(phi)*cos(theta)+w*cos(phi)*cos(theta);


%% Moments to Rotations
G_a = [-(alpha*p^2)*sign(p) -(alpha*q^2)*sign(q) -(beta*r^2)*sign(r)]; % Aerodynamic Moments
G_c = [A(1,:)*uf, A(2,:)*uf, A(3,:)*uf]; % Control Moments
G_b = G_a + G_c;
%%  Attititude

dydt(7) = p + (q*sin(phi)+r*cos(phi))*tan(theta);
dydt(8) = q*cos(phi)-r*sin(phi);
dydt(9) = (q*sin(phi)+r*cos(phi))*sec(theta);
dydt(10) = (G_b(1)+q*r*(Iy - Iz))/Ix;
dydt(11) = (G_b(2)+r*p*(Iz-Ix))/Iy;
dydt(12) = (G_b(3)+p*q*(Ix-Iy))/Iz;

dydt = dydt';
```

3. *Main Function that Plots all State Variables*

```
%% Author: Jack Lambert
% University of Colorado Engineering Department
%
% Purpose:  This codes purpose is to call the linear, non-linear, and
% initial condition functions in order to plot the simulated state
% variables for the different cases. This code calls the linear and non-
% linear functions using ODE45, which computes the differential equations
% for each of the state variables. This code plots the linear and
% non-linear cases state variables separately with respect to time, then
% compares them directly
% Date Modefied: 2/18/18
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ODE45 Variable Allocation
%                      x = z(1); % N - location
%                      dy = z(2); % E - location
%                      dz = z(3); % -D - location
%                      u = z(4); % u - component of velocity
%                      v = z(5); % v compenent of velocity
%                      w = z(6); % w component of velocity
%                      % Rotational Motion
%                      phi = z(7); % Attitude Euler Angles
%                      theta = z(8); % Attitude Euler Angles
%                      psi = z(9); % Attitude Euler Angles
%                      p = z(10); % Angular velocity about the x-axis [rad/s]
%                      q = z(11); % Angular Velocity about the y-axis [rad/s]
%                      r = z(12); % Angular Velocity about the z-axis [rad/s]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Inital Conditions:
%                      i = 1 ----> +5 [deg] Bank
%                      i = 2 ----> +5 [deg] Pitch
%                      i = 3 ----> +5 [deg] Roll Rate
%                      i = 4 ----> +5 [deg] Pitch Rate
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all;
%% Initial Condition function
[conditionL,conditionNL] = InitialConditions();
% Time for ODE45 to numerically integrate over (Linear barely Varies)
timeNL = [0:0.001:10];
timeL = [0 10];
%% For title as conditions are change
string = ["+5 [deg] Bank","+5 [deg] Pitch","+0.1 [rad/s] Roll Rate",...
    "+0.1 [rad/s] Pitch Rate"];
%% Non Linear By Itself
for i = 1:4
    % Calling ODE45
    [tNL,zNL] = ode45('NonLinear',timeNL,conditionNL{i});
    % Plotting Conditions
    figure
    % U_E vs time
    subplot(8,1,1)
    plot(tNL ,zNL(:,4),'Linewidth',1)
    tit = sprintf('%s %s','State Variable of a Quad-Copter w/',string(i));
    title(tit)
    ylabel('u_E [m/s]')

    % V_E vs time
    subplot(8,1,2)
    plot(tNL ,zNL(:,5),'Linewidth',1)
    ylabel('v_E [m/s]')

    % W_E vs time
    subplot(8,1,3)
    plot(tNL ,zNL(:,6),'Linewidth',1)
    ylabel('w_E [m/s]')

    % p vs time
    subplot(8,1,4)
    plot(tNL ,zNL(:,10),'Linewidth',1)
    ylabel('p [rad/s]')

    % q vs time
    subplot(8,1,5)
    plot(tNL ,zNL(:,11),'Linewidth',1)
    ylabel('q [rad/s]')

    % r vs time
    subplot(8,1,6)
    plot(tNL ,zNL(:,12),'Linewidth',1)
    ylabel('r [rad/s]')

    % Phi vs time
     subplot(8,1,7)
    plot(tNL ,zNL(:,7),'Linewidth',1)
    ylabel('\phi [rad]')

    % Theta vs time
    subplot(8,1,8)
    plot(tNL ,zNL(:,8),'Linewidth',1)
    ylabel('\theta [rad]')
    xlabel('Time [s]')
end

%% Linear By Itself
for i = 1:4
    % Calling ODE45
    [tL,zL] = ode45('Linear',timeL,conditionL{i});

    % Plotting Conditions
    figure
    % U_E vs time
    subplot(8,1,1)
    plot(tL ,zL(:,4),'Linewidth',1)
    tit = sprintf('%s %s','State Variable of a Quad-Copter w/',string(i));
    title(tit)
    ylabel('u_E [m/s]')

    % V_E vs time
    subplot(8,1,2)
    plot(tL ,zL(:,5),'Linewidth',1)
    ylabel('v_E [m/s]')

    % W_E vs time
    subplot(8,1,3)
    plot(tL ,zL(:,6),'Linewidth',1)
    ylabel('w_E [m/s]')

    % p vs time
    subplot(8,1,4)
    plot(tL ,zL(:,10),'Linewidth',1)
    ylabel('p [rad/s]')

    % q vs time
    subplot(8,1,5)
    plot(tL ,zL(:,11),'Linewidth',1)
    ylabel('q [rad/s]')

    % r vs time
    subplot(8,1,6)
    plot(tL ,zL(:,12),'Linewidth',1)
    ylabel('r [rad/s]')

    % Phi vs time
     subplot(8,1,7)
    plot(tL ,zL(:,7),'Linewidth',1)
    ylabel('\phi [rad]')

    % Theta vs time
    subplot(8,1,8)
    plot(tL ,zL(:,8),'Linewidth',1)
    ylabel('\theta [rad]')
    xlabel('Time [s]')

end
%% Trajectory Linear vs Non-Linear
for i = 1:4
    [tNL,zNL] = ode45('NonLinear',timeNL,conditionNL{i});
    [tL,zL] = ode45('Linear',timeL,conditionL{i});

    figure
    plot3(zNL(:,1),zNL(:,2),-zNL(:,3),'-o')
    hold on
    plot3(zL(:,1),zL(:,2),-zL(:,3),'-o')
    tit = sprintf('%s %s','Trajectory of Quad-Copter w/',string(i));
    title(tit)
    xlabel('N Displacement [m]')
    ylabel('E Displacement [m]')
    zlabel('-D Displacement [m]')
    legend('Non-Linear','Linear')
    axis equal
end

%% Linear vs. Non-Linear ( State Variables vs Time)
for i = 1:4
    % Calling ODE45
    [tL,zL] = ode45('Linear',timeL,conditionL{i});
    [tNL,zNL] = ode45('NonLinear',timeNL,conditionNL{i});

    % Plotting Conditions
    figure
    % U_E vs time
    subplot(8,1,1)
    plot(tL ,zL(:,4),'.','Linewidth',1)
    tit = sprintf('%s %s','State Variable of a Quad-Copter w/',string(i));
    title(tit)
    hold on
    plot(tNL ,zNL(:,4),'Linewidth',1)
    hold off
    ylabel('u_E [m/s]')
    legend('Linear','Non-Linear')

    % V_E vs time
    subplot(8,1,2)
    plot(tL ,zL(:,5),'.','Linewidth',1)
    hold on
    plot(tNL ,zNL(:,5),'Linewidth',1)
    hold off
    ylabel('v_E [m/s]')

    % W_E vs time
    subplot(8,1,3)
    plot(tL ,zL(:,6),'.','Linewidth',1)
    hold on
    plot(tNL ,zNL(:,6),'Linewidth',1)
    hold off
    ylabel('w_E [m/s]')

    % p vs time
    subplot(8,1,4)
    plot(tL ,zL(:,10),'.','Linewidth',1)
    hold on
    plot(tNL ,zNL(:,10),'Linewidth',1)
    hold off
    ylabel('p [rad/s]')

    % q vs time
    subplot(8,1,5)
    plot(tL ,zL(:,11),'.','Linewidth',1)
    hold on
    plot(tNL ,zNL(:,11),'Linewidth',1)
    hold off
    ylabel('q [rad/s]')

    % r vs time
    subplot(8,1,6)
    plot(tL ,zL(:,12),'.','Linewidth',1)
    hold on
    plot(tNL ,zNL(:,12),'Linewidth',1)
    hold off
    ylabel('r [rad/s]')

    % Phi vs time
     subplot(8,1,7)
    plot(tL ,zL(:,7),'.','Linewidth',1)
    hold on
    plot(tNL ,zNL(:,7),'Linewidth',1)
    hold off
    ylabel('\phi [rad/s]')

    % Theta vs time
    subplot(8,1,8)
    plot(tL ,zL(:,8),'.','Linewidth',1)
    hold on
    plot(tNL ,zNL(:,8),'Linewidth',1)
    hold off
    ylabel('\theta [rad/s]')
    xlabel('time [s]')

end
```

4. Function that Calls in Initial Conditions

```
%% Author: Jack Lambert

% Purpose: This code gives the intial conditions for both the linear and
% non-linear models for ode45 to compute the diffeential equations. This
% codes purpose is to organize
% Date Modefied: 2/18/18
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [conditionL,conditionNL] = InitialConditions()
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Linear Case
% Pertubations
delta_xE = [0 0 0 0]; % N - location [m]
delta_yE = [0 0 0 0]; % E - location [m]
delta_zE = [0 0 0 0]; % -D - location [m]
delta_u = [0 0 0 0]; % u - component of velocity [m/s]
delta_v = [0 0 0 0]; % v compenent of velocity [m/s]
delta_w = [0 0 0 0]; % w component of velocity [m/s]
delta_phi = [5*(pi/180) 0 0 0]; % Phi Euler Angle [rad]
delta_theta = [0 5*(pi/180) 0 0]; % Theta Euler Angle [rad]
delta_psi = [0 0 0 0]; % Psi Euler Angle [rad]
delta_p = [0 0 0.1 0]; % Angular velocity about the x-axis [rad/s]
delta_q = [0 0 0 0.1]; % Angular Velocity about the y-axis [rad/s]
delta_r = [0 0 0 0]; % Angular Velocity about the z-axis [rad/s]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Initial Conditions for Non-Linear Case
c1 = [0 0 0 0]; % N - location [m]
c2 = [0 0 0 0]; % E - location [m]
c3 = [0 0 0 0]; % -D - location [m]
c4 = [0 0 0 0]; % u - component of velocity [m/s]
c5 = [0 0 0 0]; % v compenent of velocity [m/s]
c6 = [0 0 0 0]; % w component of velocity [m/s]
c7 = [5*(pi/180) 0 0 0]; % Phi Euler Angle [rad]
c8 = [0 5*(pi/180) 0 0]; % Theta Euler Angle [rad]
c9 = [0 0 0 0]; % Psi Euler Angle [rad]
c10 = [0 0 0.1 0]; % Roll Rate [rad/s]
c11 = [0 0 0 0.1]; % Pitch rate [rad/s]
c12 = [0 0 0 0]; % Yaw Rate [rad/s]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Running through each of the Cases for the Variations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Linear Case
for i = 1:4
    conditionL{i}= [(delta_xE(i)) (delta_yE(i)) (delta_zE(i))...
        (delta_u(i)) (delta_v(i)) (delta_w(i))...
        (delta_phi(i)) (delta_theta(i)) (delta_psi(i))...
        (delta_p(i)) (delta_q(i)) (delta_r(i))];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Non-Linear Case
for i = 1:4
    conditionNL{i}= [c1(i) c2(i) c3(i) c4(i) c5(i) c6(i) c7(i)...
        c8(i) c9(i) c10(i) c11(i) c12(i)];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

5. *Rolling Spider Experimental Data*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Author: Jack Lambert

% Purpose: This code takes in the experimental data from the rolling
% spiders data and plots the results for the copter translation and
% rotation over time with respect to the controls in which we set the gains
% for
% Date Modefied: 2/18/18
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load('RSdata_Drone09_1405.mat')
%% Motor Commands
timeMotor = rt_motor.time(:);
Motor1 = (rt_motor.signals.values(:,1)*13840.4).^(1/2); % Motor Rotation Rate [Rad/s]
Motor2 = (abs(rt_motor.signals.values(:,2)*13840.4)).^(1/2); % Motor Rotation Rate [Rad/s]
Motor3 = (rt_motor.signals.values(:,3)*13840.4).^(1/2); % Motor Rotation Rate [Rad/s]
Motor4 = (abs(rt_motor.signals.values(:,4)*13840.4)).^(1/2); % Motor Rotation Rate [Rad/s]
%% Translation
% Position
timeest = rt_estim.time(:);
xdata = rt_estim.signals.values(:,1); % X-Position [m]
ydata = rt_estim.signals.values(:,2); % Y-Position [m]
zdata = rt_estim.signals.values(:,3); % Z-Position [m]
% Velocity
Vx = rt_estim.signals.values(:,7); % X-Velocity [m/s]
Vy = rt_estim.signals.values(:,8); % Y-Velocity [m/s]
Vz = rt_estim.signals.values(:,9); % Z-Velocity [m/s]
%% Rotation
% Attitude
yaw = rt_estim.signals.values(:,4); % [Rad]
pitch = rt_estim.signals.values(:,5); % [Rad]
roll = rt_estim.signals.values(:,6); % [Rad]
% Angular Rates
p = rt_estim.signals.values(:,10); % Body Fixed frame rotation about x-axis [Rad/s]
q = rt_estim.signals.values(:,11); % Body Fixed frame rotation about y-axis[Rad/s]
r = rt_estim.signals.values(:,12); % Body Fixed frame rotation about z-axis [Rad/s]
%% Plots of Translation vs. Time in Correlation to Motor Controls
% Motor Controls
figure(1)
subplot(3,1,1)
plot(timeMotor,Motor1)
hold on
plot(timeMotor,Motor2)
plot(timeMotor,Motor3)
plot(timeMotor,Motor4)
title('Motor Controls vs. Time')
xlabel('Time [s]')
ylabel('Motor Rotation Rate [rad/s]')
legend('Motor 1',' |Motor 2|','Motor 3','|Motor 4|')
% Position
subplot(3,1,2,'replace')
plot(timeest,xdata)
hold on
plot(timeest,ydata)
plot(timeest,zdata)
hold off
title('Position vs. Time')
ylabel('Position [m]')
xlabel('Time [s]')
legend('X-Position','Y-Position','Z-Position')
% Velocity
subplot(3,1,3,'replace')
plot(timeest,Vx)
hold on
plot(timeest,Vy)
plot(timeest,Vz)
hold off
title('Velocity vs. Time')
ylabel('Velocity [m]')
xlabel('Time [s]')
legend('X-Velocity','Y-Velocity','Z-Velocity')

%% Plots of Rotation vs. Time in Correlation to Motor Controls
% Motor Controls
figure(2)
subplot(3,1,1)
plot(timeMotor,Motor1)
hold on
```

```
plot(timeMotor,Motor2)
plot(timeMotor,Motor3)
plot(timeMotor,Motor4)
title('Motor Controls vs. Time')
xlabel('Time [s]')
ylabel('Motor Rotation Rate [rad/s]')
legend('Motor 1',' |Motor 2|','Motor 3','|Motor 4|')
% Attitude
subplot(3,1,2,'replace')
plot(timeest,roll)
hold on
plot(timeest,pitch)
plot(timeest,yaw)
hold off
title('Attitude vs. Time')
ylabel('Orientation [rad]')
xlabel('Time [s]')
legend('Roll','Pitch','Yaw')
% Anglar Velocity in Body fixed Frame
subplot(3,1,3,'replace')
plot(timeest,p)
hold on
plot(timeest,q)
plot(timeest,r)
hold off
title('Angular Velocity vs. Time')
ylabel('Angular Velocity [rad/s]')
xlabel('Time [s]')
legend('\omega_{x}','\omega_{y}','\omega_{z}')
```