University of Colorado - Boulder

ASEN 3801

Lab 5 Assignment

Authors:
Hayden Gebhardt
Zack Goldberg
Fernando Gonzalez-Clara
Wilson Wen

Instructor:
Eric Frew

College of Engineering & Applied Science
UNIVERSITY OF COLORADO BOULDER

Submitted: February 9, 2026

Contents

I. Results

A. Problem 1

Please see MATLAB function "PlotSimulation.m" in the appendix: III.B.

We were assigned to update the PlotSimulation function from the previous lab so the subplots for the control input variables show elevator, aileron and rudder in degrees and throttle as a fraction from 0 to 1.

In order to accomplish this, the normalized throttle was determined according to the condition:

$$\boldsymbol{T_{normalized}} = \left\{ \begin{array}{ll} \frac{\boldsymbol{\Delta t} - \min(\boldsymbol{\Delta t})}{\max(\boldsymbol{\Delta t}) - \min(\boldsymbol{\Delta t})}, & \text{if } \max(\boldsymbol{\Delta t}) \neq 0 \text{ and } \max(\boldsymbol{\Delta t}) \neq \min(\boldsymbol{\Delta t}) \\ \boldsymbol{\Delta t}, & \text{else} \end{array} \right\}$$

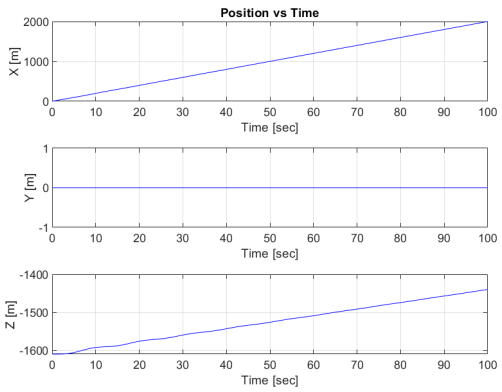
Where $\boldsymbol{\Delta t}$ is the vector of throttle inputs. The intent of this condition was to avoid a divide-by-zero error.

The elevator, aileron, and rudder inputs were simply converted to degrees through multiplication by the conversion factor, $\frac{\pi}{180}$.

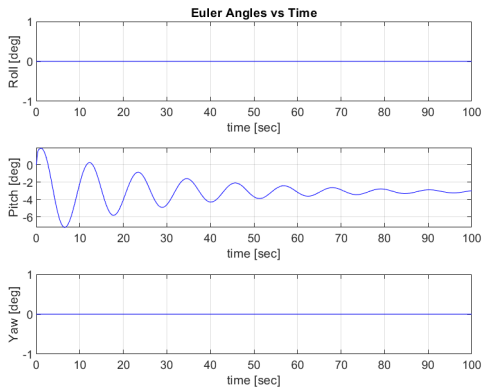
B. Problem 2

Please see MATLAB function "AircraftEOM.m" in the appendix: III.C.

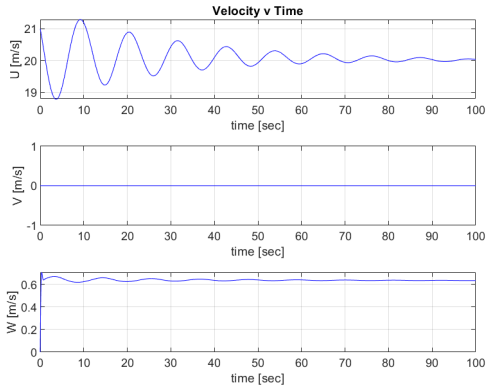
1. Problem 2.1



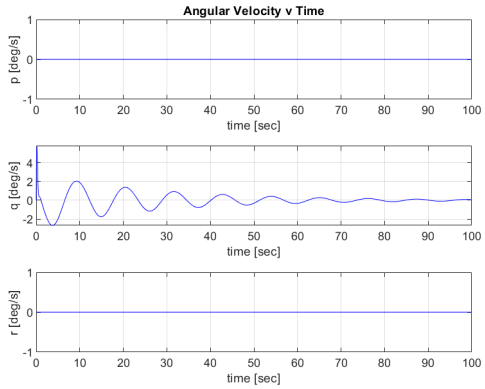
Aircraft Inertial Position vs. Time



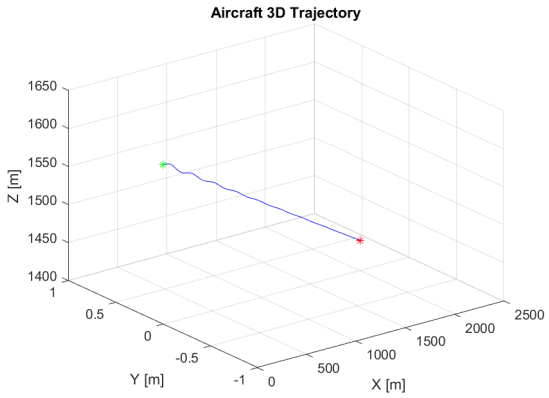
Euler Angles vs. Time



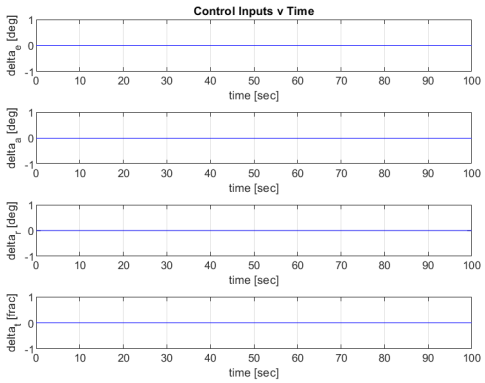
Aircraft Velocity vs. Time



Angular Velocity vs. Time



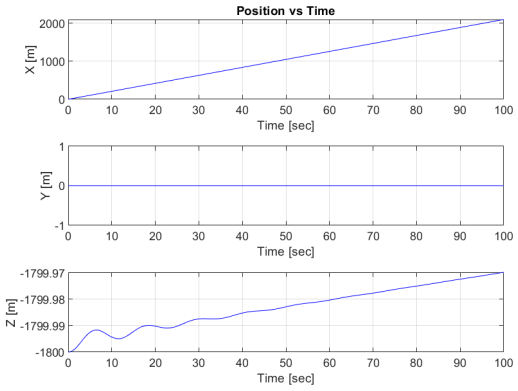
Aircraft 3D Trajectory



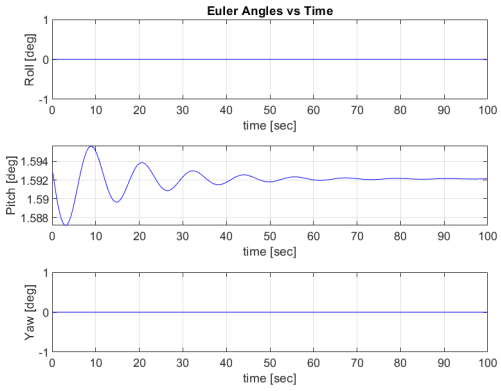
Control Inputs vs. Time

As we look at the motion of this aircraft with the aircraft's initial state and control inputs, we can first see that the Position of the aircraft has non-negligible variation concerning Y-Axis. The aircraft increases linearly in the X-direction and Z-position to time. Within the Roll angle and Yaw angle, there is a constant zero suggesting stable flight within these parameters. However, when we look at the Pitch angle, it exhibits oscillatory behavior, indicating periodic changes in the aircraft's pitch. The oscillations appear to dampen over time, suggesting a system that's returning to a steady state. Now the Y-axis velocity(V) is almost constant at zero, aligning with the negligible movement in the Y position and indicating no side-slip. The Z-axis velocity(W) is relatively constant, which corresponds with the nearly linear Z position, suggesting steady vertical motion. The X-axis velocity(U) shows an initial spike and then oscillates before stabilizing around a constant value, suggesting a disturbance at the beginning that the system overcomes to achieve steady forward velocity. The p and r angular velocity is also nearly constant at zero, indicating no significant rotational motion around the X and Z axis. The q angular velocity shows a dampened oscillatory pattern, which correlates with the pitch angle behavior, suggesting a disturbance in pitch that the aircraft stabilizes from over time. When analyzing the oscillatory pattern regarding the Pitch angle, X-axis velocity, and q angular velocity time history, the aircraft is not considered to be in trim state. To achieve trim condition, various aerodynamics forces and moments are required to be in equilibrium; However, a zero control inputs indicates the absence of active control to counteract external disturbances thus maintain its trim condition.

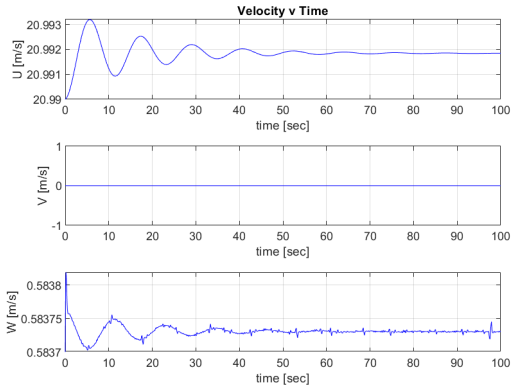
2. Problem 2.2



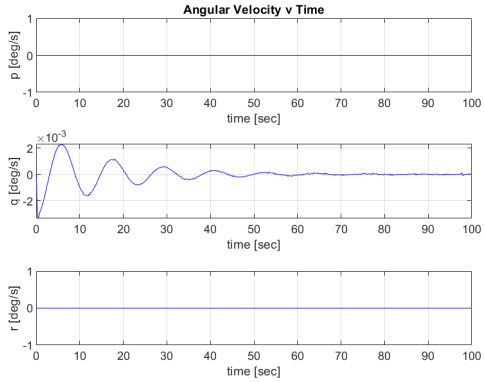
Aircraft Inertial Position vs. Time



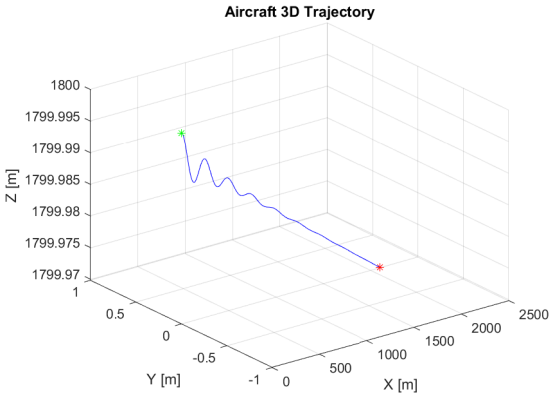
Euler Angles vs. Time



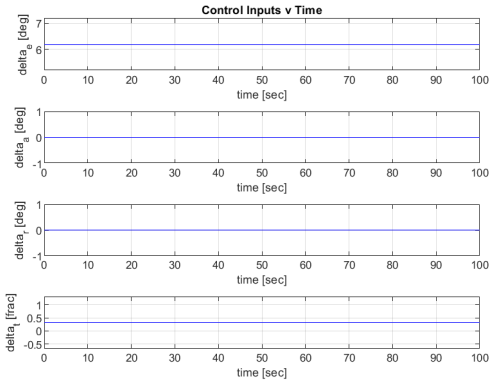
Aircraft Velocity vs. Time



Angular Velocity vs. Time



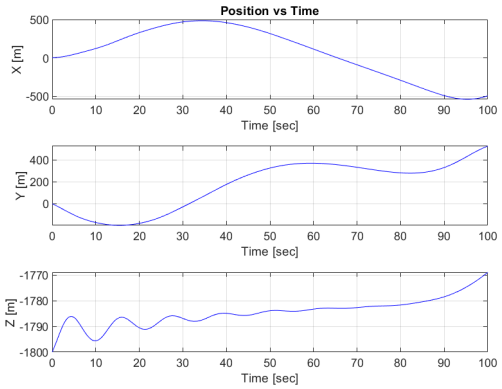
Aircraft 3D Trajectory



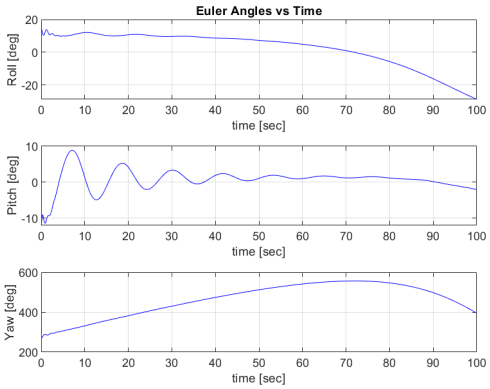
Control Inputs vs. Time

As we see the aircraft's initial state and control inputs from the second set of values, we see the aircraft has a constant forward movement with consistent velocity. There is along the Y-axis, indicating the aircraft maintains a constant lateral position, neither moving left nor right. The aircraft also shows a relatively fixed position on the Z-axis, suggesting it is maintaining a consistent altitude with very minor fluctuations. The roll angle and the angular velocity associated with the roll(p) are constant at zero, indicating the aircraft is not tilting to either side. It maintains a level orientation relative to its lateral axis. The pitch angle and its angular velocity show very slight variations, suggesting the aircraft is mostly level in its longitudinal axis with minimal nose-up or nose-down movement but ultimately maintains a steady stable flight near the end of its time flight. It can also be seen that the oscillatory pattern regarding these two plots are inversely symmetric, indicating the control inputs are dampening the disturbances. The yaw angle and its angular velocity are constant at zero, indicating the aircraft is not turning left or right and maintains a steady direction.

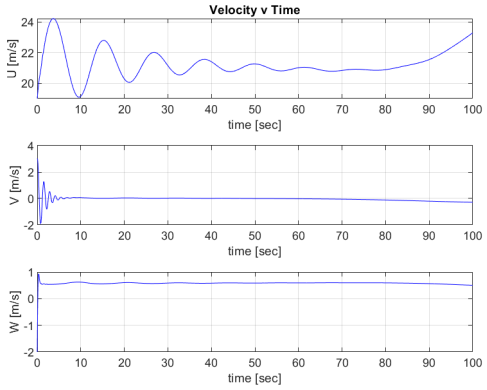
3. Problem 2.3



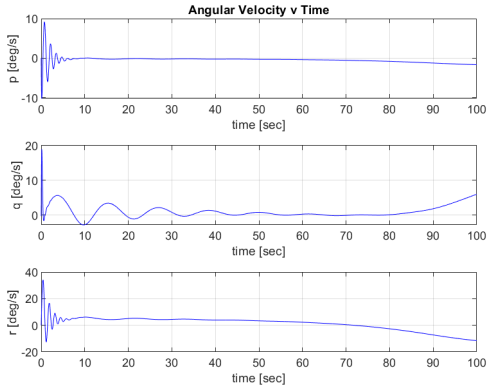
Aircraft Inertial Position vs. Time



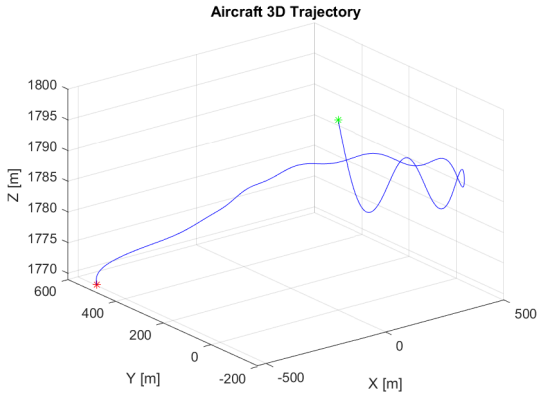
Euler Angles vs. Time



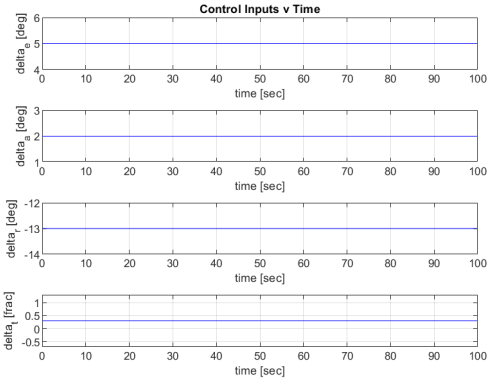
Aircraft Velocity vs. Time



Angular Velocity vs. Time



Aircraft 3D Trajectory



Control Inputs vs. Time

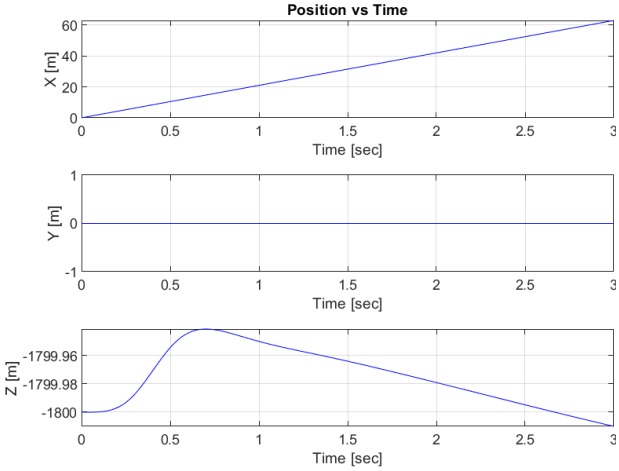
As we see the aircraft's initial state and control inputs from the second set of values given we can see that within the x-axis, the aircraft has a constant forward motion. The velocity in this axis is seen as an oscillation, suggesting changes in speed or acceleration and deceleration, with a steady state speed as it goes to time. The aircraft is rotating

about its longitudinal axis, indicating banking or tilting motions. The aircraft is moving along the Y-axis, suggesting lateral movements such as drifting or sidestepping. The velocity in the Y direction is also variable, indicating changes in the rate of lateral movement. The aircraft is rotating about its lateral axis, indicating pitch movements like nose-up or nose-down adjustments. Now in the Z-axis, the position points to changes in altitude, which in this case is descending. The velocity in the axis is variable, showing oscillation in the rate of ascent and descent. The aircraft exhibits rotation about its vertical axis, indicating yaw movements such as turning left or right. In the end, the aircraft's motion is dynamic across all three axes, with varying velocities and rotational movements. It does not follow a straight, level flight path but instead shows a complex motion pattern involving changes in direction, speed, and orientation.

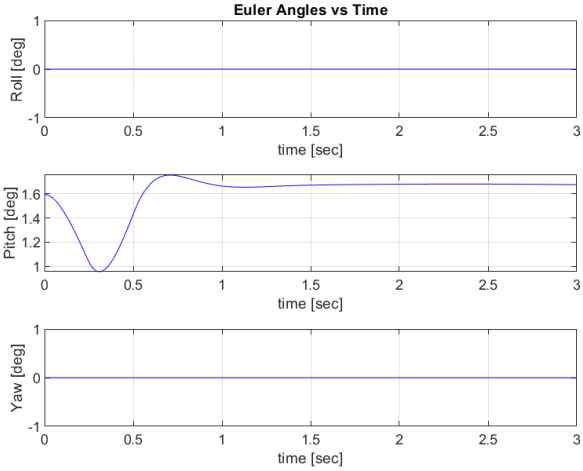
C. Problem 3

Please see MATLAB function "AircraftEOMDoublet.m" in the appendix: III.A.

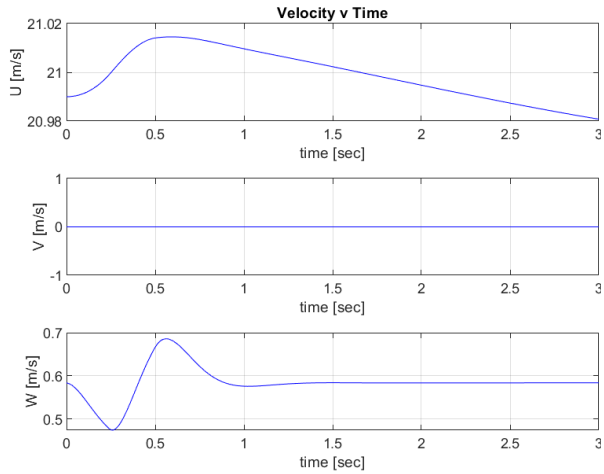
1. Problem 3.1



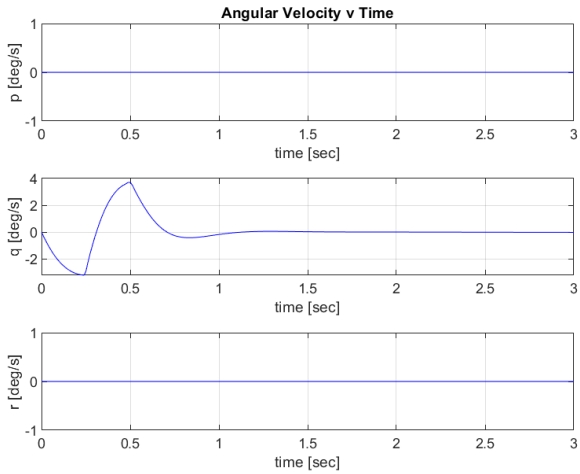
Aircraft Inertial Position vs. Time



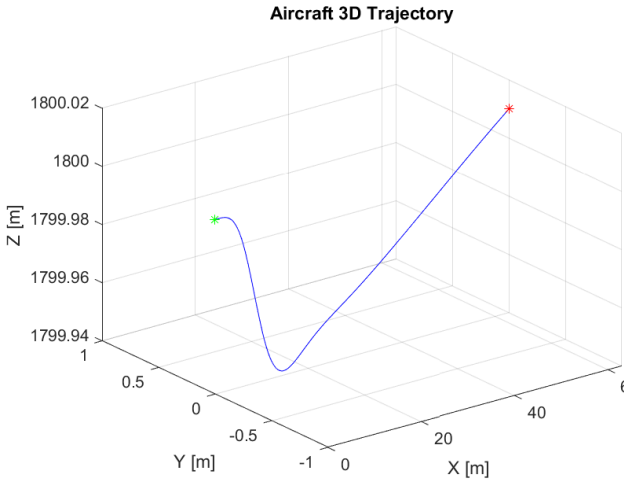
Euler Angles vs. Time



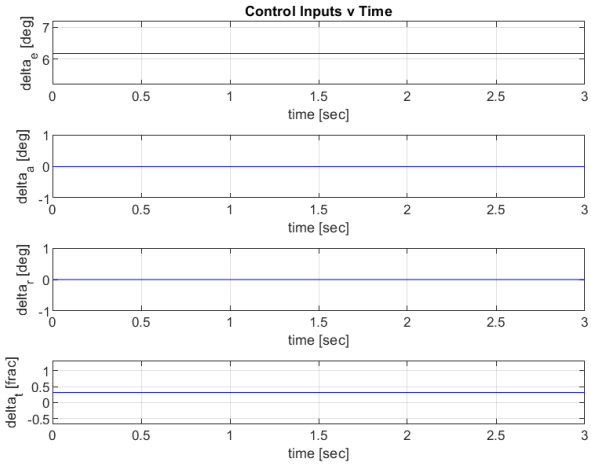
Aircraft Velocity vs. Time



Aircraft Angular Velocity vs. Time



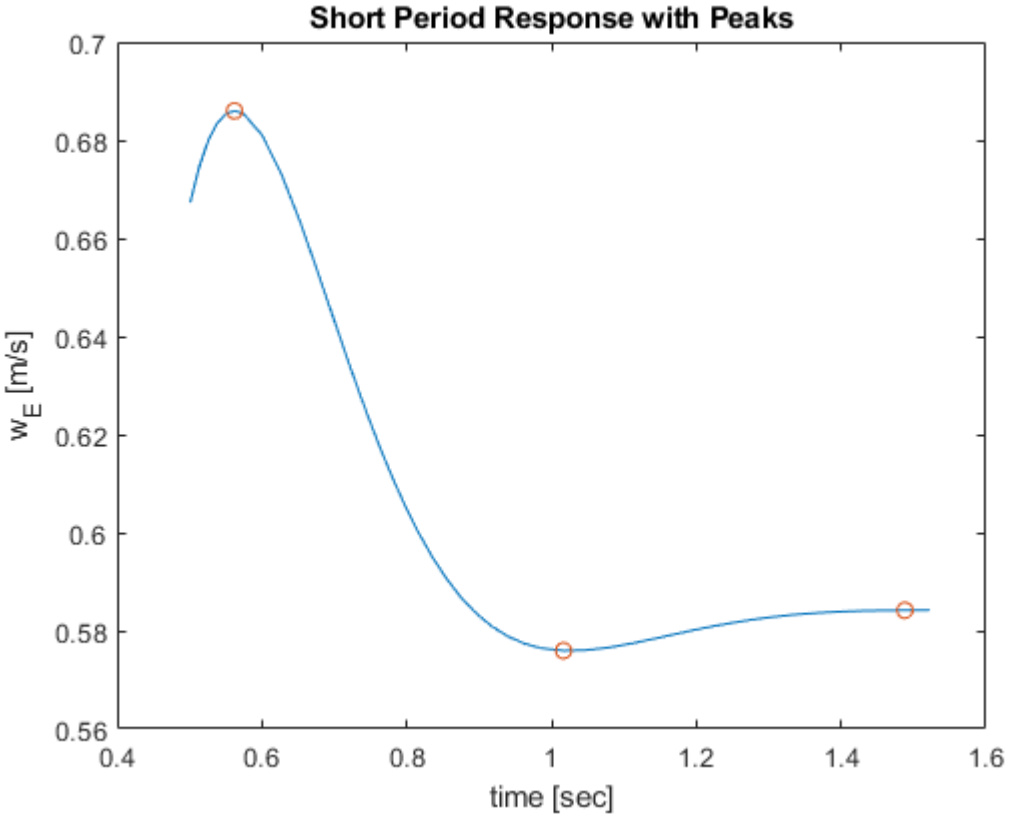
Aircraft 3D Trajectory



Aircraft Control Inputs vs. Time

2. Short Period Response

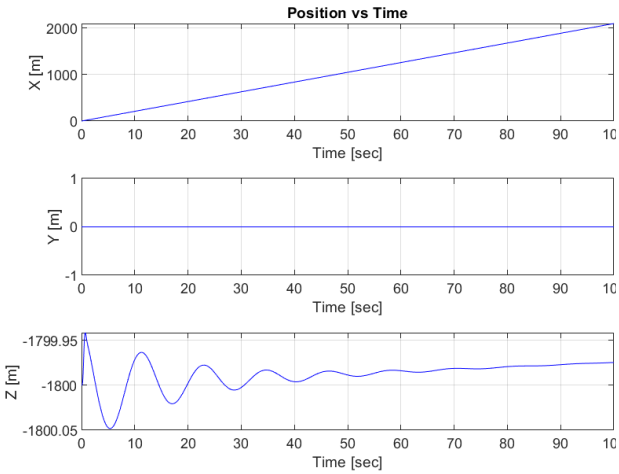
Using the data for the short period section of the aircraft's response to a doublet perturbation, we calculated the appropriate natural frequency and damping ratio:



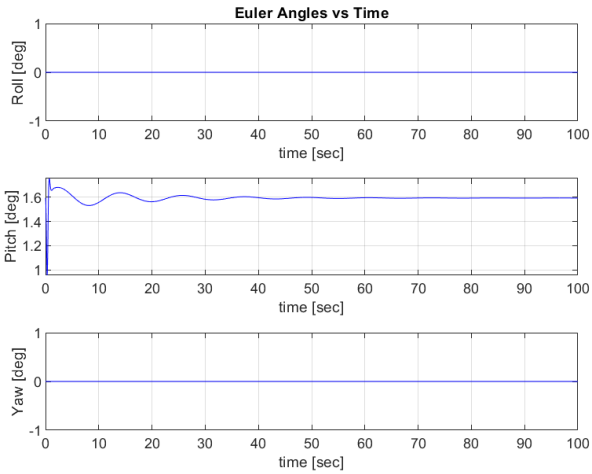
Trimmed Short Period Response with Peaks

Term	Value	Units
Natural Frequency (ω_n)	**6.7857**	rad/s
Damping Ratio (ζ)	**0.017**	Unitless

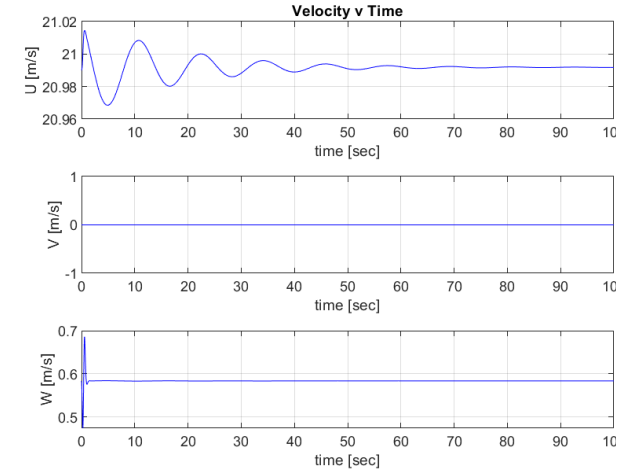
3. Problem 3.2



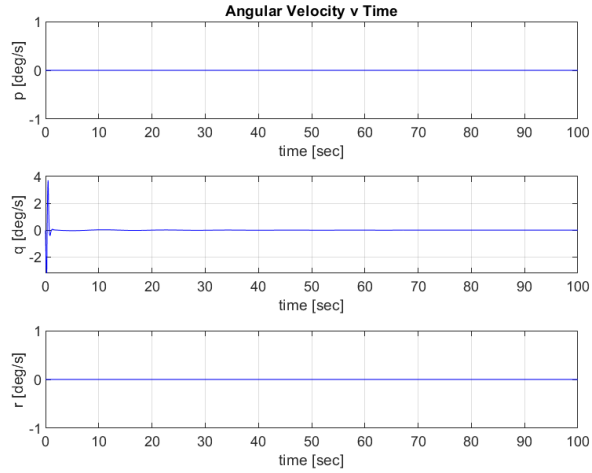
Aircraft 3D Trajectory



Aircraft Control Inputs vs. Time



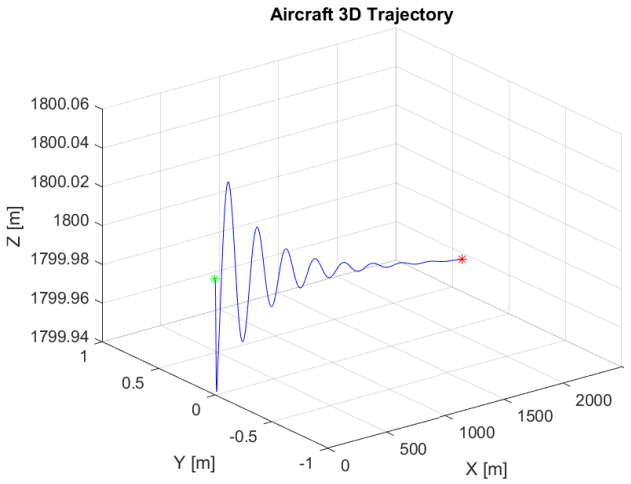
Aircraft 3D Trajectory



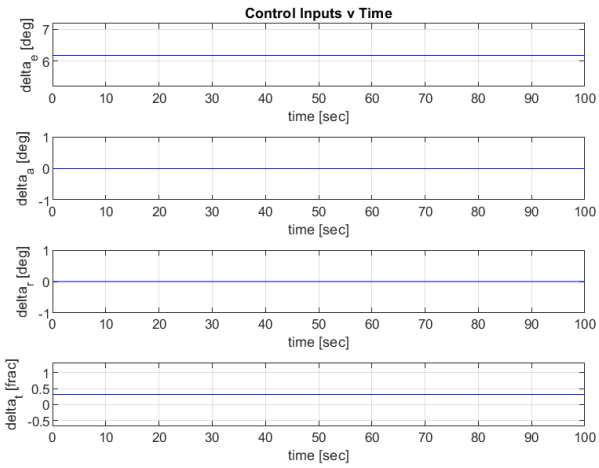
Aircraft Control Inputs vs. Time

4. Phugoid Response

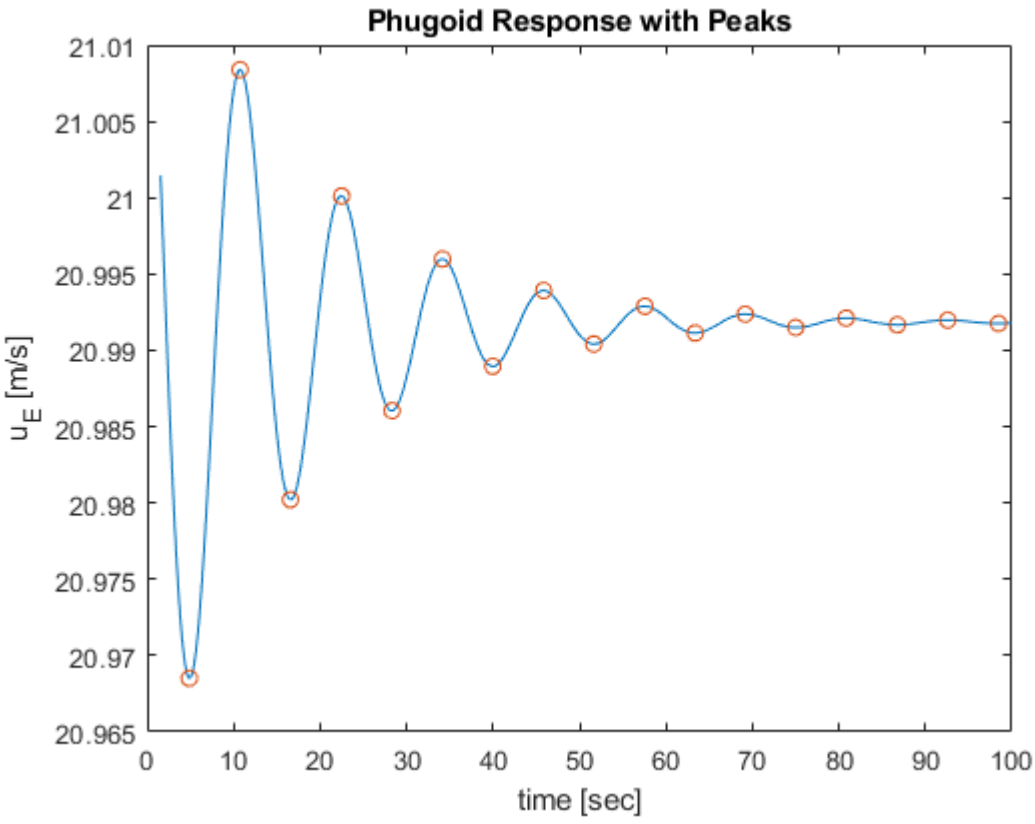
Using the data for the Phugoid section of the aircraft's response to a doublet perturbation, we calculated the appropriate natural frequency and damping ratio:



Aircraft 3D Trajectory



Aircraft Control Inputs vs. Time



Trimmed Phugoid Response with Peaks

II. Acknowledgements

Term	Value	Units
Natural Frequency (ω_n)	**0.5372**	rad/s
Damping Ratio (ζ)	**0.00002074**	Unitless

	Plan	Model	Experiment	Results	Report	Code	ACK
Zack	2	1	1	2	1	2	X
Hayden	1	1	1	2	2	2	X
Fernando	1	1	1	1	1	1	X
Wilson	0	1	1	1	1	0	X

III. Appendix - MATLAB Code

A. Main Function

```
% ASEN 3801 - Lab 5
% Hayden Gebhardt
% Zackary Goldberg
% Fernando Gonzalez-Clara

%% Housekeeping

clc; clear; close all;

%% Variable Declarations

ttwistor; % call aircraft parameters

%% Problem 1

% PlotSimulation(time, aircraft_state_array, control_input_array, col);

%% Problem 2

time = [0 100];

% 1 initial state and control inputs
X_0 = [0 0 -1609.34 0 0 0 21 0 0 0 0 0]';
u_0 = [0 0 0 0]';
wind_inertial = [0 0 0]';

[t, x] = ode45(@(t, aircraft_state) AircraftEOM(t, aircraft_state, u_0,
    wind_inertial, aircraft_parameters), time, X_0);
control_input_array = u_0' .* ones(length(t), 1);

PlotSimulation(t, x, control_input_array, 'b-');

% 2 initial state and control inputs
X_0 = [0 0 -1800 0 0.02780 0 20.99 0 0.5837 0 0 0]';
u_0 = [0.1079 0 0 0.3182]';

[t, x] = ode45(@(t, aircraft_state) AircraftEOM(t, aircraft_state, u_0,
    wind_inertial, aircraft_parameters), time, X_0);
```

```
control_input_array = u_0' .* ones(length(t), 1);

PlotSimulation(t, x, control_input_array, 'b-');

% 3 initial state and control inputs
a = convangvel([0.08 -0.2 0],'deg/s', 'rad/s');
X_0 = [0 0 -1800 deg2rad(15) deg2rad(-12) deg2rad(270) 19 3 -2 a(1) a(2) a(3)
    ]';
u_0 = [deg2rad(5) deg2rad(2) deg2rad(-13) 0.3]';

[t, x] = ode45(@(t, aircraft_state) AircraftEOM(t, aircraft_state, u_0,
    wind_inertial, aircraft_parameters), time, X_0);
control_input_array = u_0' .* ones(length(t), 1);

PlotSimulation(t, x, control_input_array, 'b-');

%% Save figures
SaveFigures(1:18, "Figures/Problem2", "Figure_*FIGURE*");

%% Problem 3

time = [0 3];

% 1 short period behavior
X_0 = [0 0 -1800 0 0.02780 0 20.99 0 0.5837 0 0 0]';
u_0 = [0.1079 0 0 0.3182]';
wind_inertial = [0 0 0]';

doublet_size = deg2rad(15);
doublet_time = 0.25;

[t, x] = ode45(@(t, aircraft_state) AircraftEOMDoublet(t, aircraft_state, u_0,
     doublet_size, doublet_time, wind_inertial, aircraft_parameters), time,
    X_0);
control_input_array = u_0' .* ones(length(t), 1);

PlotSimulation(t, x, control_input_array, 'b-');

t_SP = t(55:103, :)';
x_SP = x(55:103, 9)';

[ypeaks_SP, xpeaks_SP, indices_SP] = findMaxAndMin(x_SP, t_SP);

% figure();
% plot(t_SP, x_SP)
% hold on;
% plot(xpeaks_SP, ypeaks_SP, 'o')
% hold off;

periods_SP = 2 * mean(diff(t_SP(indices_SP)));
meanPeriod_SP = mean(periods_SP);

omega_n_SP = 2 * pi / meanPeriod_SP;

```

```
amplitudes_SP = ypeaks_SP;
damping_ratios_SP = zeros(size(amplitudes_SP));

for i = 1:length(amplitudes_SP)-1
    damping_ratios_SP(i) = -log(amplitudes_SP(i+1)/amplitudes_SP(i)) / sqrt(pi
        ^2 + (log(amplitudes_SP(i+1)/amplitudes_SP(i)))^2);
end

mean_damping_ratio_SP = abs(mean(damping_ratios_SP));

sprintf('Estimated Short Period Natural Frequency: %.4f', omega_n_SP)
sprintf('Estimated Short Period Damping Ratio: %.4f', mean_damping_ratio_SP)

% 2 phugoid behavior

time = [0 100];

[t, x] = ode45(@(t, aircraft_state) AircraftEOMDoublet(t, aircraft_state, u_0,
    doublet_size, doublet_time, wind_inertial, aircraft_parameters), time,
    X_0);
control_input_array = u_0' .* ones(length(t), 1);

PlotSimulation(t, x, control_input_array, 'b-');

t_PH = t(104:end, :)';
x_PH = x(104:end, 7)';

[ypeaks_PH, xpeaks_PH, indices_PH] = findMaxAndMin(x_PH, t_PH);

% figure();
% plot(t_PH, x_PH)
% hold on;
% plot(xpeaks_PH, ypeaks_PH, 'o')
% hold off;

periods_PH = 2 * mean(diff(t_PH(indices_PH)));
meanPeriod_PH = mean(periods_PH);

omega_n_PH = 2 * pi / meanPeriod_PH;

amplitudes_PH = ypeaks_PH;
damping_ratios_PH = zeros(size(amplitudes_PH));

for i = 1:length(amplitudes_PH)-1
    damping_ratios_PH(i) = -log(amplitudes_PH(i+1)/amplitudes_PH(i)) / sqrt(pi
        ^2 + (log(amplitudes_PH(i+1)/amplitudes_PH(i)))^2);
end

mean_damping_ratio_PH = abs(mean(damping_ratios_PH));

sprintf('Estimated Phugoid Natural Frequency: %.4f', omega_n_PH)
sprintf('Estimated Phugoid Damping Ratio: %.10f', mean_damping_ratio_PH)

% Save figures
```

```
137 SaveFigures(19:30, "Figures/Problem3", "Figure_*FIGURE*");
```

B. PlotSimulation

```
1  % Zack Goldberg, Fernando Gonzalez Clara, Hayden Gebhardt, Wilson Wen
2  % ASEN 3801
3  % SaveFigures
4  % Created: 12/1/23
5  function PlotSimulation(time, aircraft_state_array, control_input_array, col)
6
7      % Convert aileron, rudder, elevator from rad to deg
8      control_input_array(:, 1:3) = rad2deg(control_input_array(:, 1:3));
9
10     figure();
11
12     % Figure: XYZ-position vs time
13     subplot(311);
14     plot(time, aircraft_state_array(:,1),col);hold on;
15     title('Position vs Time');
16     grid on;
17     xlabel('Time [sec]');
18     ylabel('X [m]');
19
20     subplot(312);
21     plot(time, aircraft_state_array(:,2),col);hold on;
22     grid on;
23     xlabel('Time [sec]');
24     ylabel('Y [m]')
25
26     subplot(313);
27     plot(time, aircraft_state_array(:,3),col);hold on;
28     grid on;
29     xlabel('Time [sec]');
30     ylabel('Z [m]')
31
32     % Figure: Euler Angles vs Time
33     figure();
34
35     subplot(311);
36     plot(time, rad2deg(aircraft_state_array(:,4)),col);hold on;
37     title('Euler Angles vs Time');
38     grid on;
39     xlabel('time [sec]');
40     ylabel('Roll [deg]')
41
42     subplot(312);
43     plot(time, rad2deg(aircraft_state_array(:,5)),col);hold on;
44     grid on;
45     xlabel('time [sec]');
46     ylabel('Pitch [deg]')
47
48     subplot(313);
49     plot(time, rad2deg(aircraft_state_array(:,6)),col);hold on;
50     grid on;
```

```
51      xlabel('time [sec]');
52      ylabel('Yaw [deg]')
53
54      % Figure: Body (UVW) Velocity vs Time
55      figure();
56
57      subplot(311);
58      plot(time, aircraft_state_array(:,7),col);hold on;
59      title('Velocity vs Time');
60      grid on;
61      xlabel('time [sec]');
62      ylabel('U [m/s]')
63
64      subplot(312);
65      plot(time, aircraft_state_array(:,8),col);hold on;
66      grid on;
67      xlabel('time [sec]');
68      ylabel('V [m/s]')
69
70      subplot(313);
71      plot(time, aircraft_state_array(:,9),col);hold on;
72      grid on;
73      xlabel('time [sec]');
74      ylabel('W [m/s]')
75
76      % Figure: Angular Velocity (pqr) vs times
77      figure();
78
79      subplot(311);
80      plot(time, rad2deg(aircraft_state_array(:,10)),col);hold on;
81      title('Angular Velocity vs Time');
82      grid on;
83      xlabel('time [sec]');
84      ylabel('p [deg/s]')
85
86      subplot(312);
87      plot(time, rad2deg(aircraft_state_array(:,11)),col);hold on;
88      grid on;
89      xlabel('time [sec]');
90      ylabel('q [deg/s]')
91
92      subplot(313);
93      plot(time, rad2deg(aircraft_state_array(:,12)),col);hold on;
94      grid on;
95      xlabel('time [sec]');
96      ylabel('r [deg/s]')
97
98      % Figure: Aircraft 3D Trajectory
99      figure();
100     plot3(aircraft_state_array(:,1),aircraft_state_array(:,2),-
            aircraft_state_array(:,3),col);hold on;
101     grid on;
102     xlabel('X [m]');
103     ylabel('Y [m]');
```

```
104         zlabel('Z [m]');
105         title('Aircraft 3D Trajectory');
106         hold on;
107         plot3(aircraft_state_array(1,1),aircraft_state_array(1,2),-
                aircraft_state_array(1,3),'g*');
108         plot3(aircraft_state_array(end,1),aircraft_state_array(end,2),-
                aircraft_state_array(end,3),'r*');
109
110         delta_t = control_input_array(:,4);
111
112         % Normalize the throttle between zero and one
113         if max(delta_t) ~= 0 && max(delta_t) ~= min(delta_t)
114
115             T_normalized = (delta_t - min(delta_t)) ./ (max(delta_t) - min(delta_t
                    ));    % normalizing throttle as a fraction from 0 to 1
116         else
117
118             T_normalized = delta_t * ones(1, length(control_input_array(:, 4)));
119         end
120
121         % Figure: Control Inputs (delta_e, delta_a, delta_r, delta-t) vs Time
122         figure();
123
124         subplot(411);
125         plot(time, control_input_array(:,1),col);hold on;
126         title('Control Inputs vs Time');
127         grid on;
128         xlabel('time [sec]');
129         ylabel('delta_e [deg]')
130
131         subplot(412);
132         plot(time, control_input_array(:,2),col);hold on;
133         grid on;
134         xlabel('time [sec]');
135         ylabel('delta_a [deg]')
136
137         subplot(413);
138         plot(time, control_input_array(:,3),col);hold on;
139         grid on;
140         xlabel('time [sec]');
141         ylabel('delta_r [deg]')
142
143         subplot(414);
144         plot(time, T_normalized,col);hold on;
145         grid on;
146         xlabel('time [sec]');
147         ylabel('delta_t [frac]')
148
149 end
```

C. AircraftEOM

```
1 % Hayden Gebhardt
2 % ASEN 3801
```

```
% AircraftEOM
% Created: 12/1/23
function xdot = AircraftEOM(time, aircraft_state, aircraft_surfaces,
    wind_inertial, aircraft_parameters)

    x = aircraft_state(1);
    y = aircraft_state(2);
    z = aircraft_state(3);

    phi = aircraft_state(4);
    theta = aircraft_state(5);
    psi = aircraft_state(6);

    u_E = aircraft_state(7);
    v_E = aircraft_state(8);
    w_E = aircraft_state(9);

    p = aircraft_state(10);
    q = aircraft_state(11);
    r = aircraft_state(12);

    % Physical properties
    height = -aircraft_state(3);
    g = 9.81;
    m = aircraft_parameters.m;

    density = stdatmo(height);

    [aero_forces, aero_moments] = AeroForcesAndMoments(aircraft_state,
        aircraft_surfaces, wind_inertial, density, aircraft_parameters);

    L = aero_moments(1);
    M = aero_moments(2);
    N = aero_moments(3);

    % EOM

    x_dot = [cos(theta) * cos(psi), sin(phi) * sin(theta) * cos(psi) - cos(phi
        ) * sin(psi), cos(phi) * sin(theta) * cos(psi) + sin(phi) * sin(psi)];
    y_dot = [cos(theta) * sin(psi), sin(phi) * sin(theta) * sin(psi) + cos(phi
        ) * cos(psi), cos(phi) * sin(theta) * sin(psi) - sin(phi) * cos(psi)];
    z_dot = [-sin(theta), sin(phi) * cos(theta), cos(phi) * cos(theta)];

    v_dot = [x_dot ; y_dot; z_dot] * [u_E; v_E; w_E];

    phi_dot =   [1, sin(phi) * tan(theta), cos(phi) * tan(theta)];
    theta_dot = [0, cos(phi), -sin(phi)];
    psi_dot =   [0, sin(phi) * sec(theta), cos(phi) * sec(theta)];

    euler_rates = [phi_dot; theta_dot; psi_dot] * [p; q; r];

    accel = [(r * v_E) - (q * w_E); (p * w_E) - (r * u_E); (q * u_E) - (p *
        v_E)];
    grav = g.*[-sin(theta); cos(theta).*sin(phi); cos(theta).*cos(phi)];
```

```
    aero = (1/m) * aero_forces;

    vel_rates = accel + grav + aero;

    gamma = aircraft_parameters.Ix * aircraft_parameters.Iz -
        aircraft_parameters.Ixz^2;
    gamma_1 = (aircraft_parameters.Ixz * (aircraft_parameters.Ix -
        aircraft_parameters.Iy + aircraft_parameters.Iz)) / gamma;
    gamma_2 = (aircraft_parameters.Iz * (aircraft_parameters.Iz -
        aircraft_parameters.Iy) + aircraft_parameters.Ixz^2) / gamma;
    gamma_3 = aircraft_parameters.Iz / gamma;
    gamma_4 = aircraft_parameters.Ixz / gamma;
    gamma_5 = (aircraft_parameters.Iz - aircraft_parameters.Ix) /
        aircraft_parameters.Iy;
    gamma_6 = aircraft_parameters.Ixz / aircraft_parameters.Iy;
    gamma_7 = (aircraft_parameters.Ix * (aircraft_parameters.Ix -
        aircraft_parameters.Iy) + aircraft_parameters.Ixz^2) / gamma;
    gamma_8 = aircraft_parameters.Ix / gamma;

    p_dot = (gamma_1 * p * q - gamma_2 * q * r) + (gamma_3 * L + gamma_4 * N);
    q_dot = (gamma_5 * p * r - gamma_6 * (p^2 - r^2)) + ((1/
        aircraft_parameters.Iy) * M);
    r_dot = (gamma_7 * p * q - gamma_1 * q * r) + (gamma_4 * L + gamma_8 * N);

    angular_rates = [p_dot; q_dot; r_dot];

    xdot = [v_dot; euler_rates; vel_rates; angular_rates];

end
```

D. AircraftEOMDoublet

```
% Hayden Gebhardt
% ASEN 3801
% AircraftEOMDoublet
% Created: 12/1/23
function xdot = AircraftEOMDoublet(time, aircraft_state, aircraft_surfaces,
    doublet_size, doublet_time, wind_inertial, aircraft_parameters)

    % Unpack state vector
    x_E = aircraft_state(1);
    y_E = aircraft_state(2);
    z_E = aircraft_state(3);

    phi = aircraft_state(4);
    theta = aircraft_state(5);
    psi = aircraft_state(6);

    u_E = aircraft_state(7);
    v_E = aircraft_state(8);
    w_E = aircraft_state(9);

    p = aircraft_state(10);
    q = aircraft_state(11);
```

```
    r = aircraft_state(12);

    % Physical properties
    height = -aircraft_state(3);
    g = 9.81;
    m = aircraft_parameters.m;

    density = stdatmo(height);

    % Aero Forces and Moments
    if any(time > 0 & time <= doublet_time)

        aircraft_surfaces(1) = aircraft_surfaces(1) + doublet_size;

    elseif any(time > doublet_time & time <= 2 * doublet_time)

        aircraft_surfaces(1) = aircraft_surfaces(1) - doublet_size;

    else

        aircraft_surfaces(1) = aircraft_surfaces(1);
    end

    % Compute aero forces and moments
    [aero_forces, aero_moments] = AeroForcesAndMoments(aircraft_state,
        aircraft_surfaces, wind_inertial, density, aircraft_parameters);

    % Unpack aero moments
    L = aero_moments(1);
    M = aero_moments(2);
    N = aero_moments(3);

    % EOM

    x_dot = [cos(theta) * cos(psi), sin(phi) * sin(theta) * cos(psi) - cos(phi
        ) * sin(psi), cos(phi) * sin(theta) * cos(psi) + sin(phi) * sin(psi)];
    y_dot = [cos(theta) * sin(psi), sin(phi) * sin(theta) * sin(psi) + cos(phi
        ) * cos(psi), cos(phi) * sin(theta) * sin(psi) - sin(phi) * cos(psi)];
    z_dot = [-sin(theta), sin(phi) * cos(theta), cos(phi) * cos(theta)];

    v_dot = [x_dot ; y_dot; z_dot] * [u_E; v_E; w_E];

    phi_dot =   [1, sin(phi) * tan(theta), cos(phi) * tan(theta)];
    theta_dot = [0, cos(phi), -sin(phi)];
    psi_dot =   [0, sin(phi) * sec(theta), cos(phi) * sec(theta)];

    euler_rates = [phi_dot; theta_dot; psi_dot] * [p; q; r];

    accel = [(r * v_E) - (q * w_E); (p * w_E) - (r * u_E); (q * u_E) - (p *
        v_E)];
    grav = g.*[-sin(theta); cos(theta).*sin(phi); cos(theta).*cos(phi)];
    aero = (1/m) * aero_forces;

    vel_rates = accel + grav + aero;
```

```
72
73      % Compute gammas
74      gamma = aircraft_parameters.Ix * aircraft_parameters.Iz -
            aircraft_parameters.Ixz^2;
75      gamma_1 = (aircraft_parameters.Ixz * (aircraft_parameters.Ix -
            aircraft_parameters.Iy + aircraft_parameters.Iz)) / gamma;
76      gamma_2 = (aircraft_parameters.Iz * (aircraft_parameters.Iz -
            aircraft_parameters.Iy) + aircraft_parameters.Ixz^2) / gamma;
77      gamma_3 = aircraft_parameters.Iz / gamma;
78      gamma_4 = aircraft_parameters.Ixz / gamma;
79      gamma_5 = (aircraft_parameters.Iz - aircraft_parameters.Ix) /
            aircraft_parameters.Iy;
80      gamma_6 = aircraft_parameters.Ixz / aircraft_parameters.Iy;
81      gamma_7 = (aircraft_parameters.Ix * (aircraft_parameters.Ix -
            aircraft_parameters.Iy) + aircraft_parameters.Ixz^2) / gamma;
82      gamma_8 = aircraft_parameters.Ix / gamma;
83
84      % Compute angular rates
85      p_dot = (gamma_1 * p * q - gamma_2 * q * r) + (gamma_3 * L + gamma_4 * N);
86      q_dot = (gamma_5 * p * r - gamma_6 * (p^2 - r^2)) + ((1/
            aircraft_parameters.Iy) * M);
87      r_dot = (gamma_7 * p * q - gamma_1 * q * r) + (gamma_4 * L + gamma_8 * N);
88
89      % Package angular rates
90      angular_rates = [p_dot; q_dot; r_dot];
91
92      % Package xdot vector
93      xdot = [v_dot; euler_rates; vel_rates; angular_rates];
94
95  end
```

E. Wrappers and Miscellaneous Functions

```
1  % Zack Goldberg
2  % ASEN 3801
3  % findMaxAndMin
4  % Created: 12/13/23
5  function [yvalues, xvalues, indices] = findMaxAndMin(y, x)
6
7  % Find all maxima
8  [MaximaYValues, MaximaXValues] = findpeaks(y, x);
9
10 % Find maxima indices
11 [~, MaximaIndices] = findpeaks(y);
12
13 % Find all minima
14 [MinimaYValues, MinimaXValues] = findpeaks(-y, x);
15
16 % Find minima indices
17 [~, MinimaIndices] = findpeaks(-y);
18
19 % Sort the x values in order
20 [xvalues, order] = sort([MaximaXValues, MinimaXValues], "ascend");
21
```

```
% Sort the y values and indices in the same order as the x values
yvalues = [MaximaYValues, MinimaYValues];
yvalues = yvalues(order);

indices = [MaximaIndices, MinimaIndices];
indices = indices(order);

end
```

```
% Zack Goldberg
% ASEN 3801
% SaveFigures
% Created: 12/1/23
function SaveFigures(fignums, dir, figname)

    % Create the appropriate directory if it does not exist
    if ~isfolder(dir)
        mkdir(dir)
    end

    % Save the figure with an appropriate filename
    for k = fignums
        filename = strrep(figname, "*FIGURE*", int2str(k));
        saveas(k, fullfile(dir, filename), 'png');
    end

end
```

```
% Zack Goldberg
% ASEN 3801
% TransformFromInertialToBody
% Created: 12/1/23
function vecbody = TransformFromInertialToBody(vecinertial, attitude)
%
% Ourputs:  vecinertial = 3x1 vector in inertial coordinates
%           attitude = 3x1 or 1x3 euler angles
%
% Inputs:   vecbody = 3x1 vector in body coordinates
%
% This function wraps around the RotationMatrix321 function to provide a
% simple interface to transform a vector from the inertial to the body
% coordinate system.

vecbody = RotationMatrix321(attitude)*vecinertial;

end
```

```
function R321 = RotationMatrix321(attitude)
% Zack Goldberg, Fernando Gonzalez Clara, Jackson Rehl (Lab 4)
% ASEN 3801
% RotationMatrix313
% Created: 9/15/23
%
% Inputs: attitude = 3x1 euler angles from EulerAngles321.m
```

```
8  %
9  % Output: 3x3 rotation matrix in the from
10 %         R321 = [a1 a2 a3; b1 b2 b3; c1 c2 c3];
11 %
12 % Methodology: using the euler angles given in order to algebraically
13 %               compute a 3-2-1 rotation matrix parameterized by those
14 %               angles.
15
16 phi = attitude(1);
17 theta = attitude(2);
18 psi = attitude(3);
19
20 a1 = cos(theta)*cos(psi);
21 a2 = cos(theta)*sin(psi);
22 a3 = -sin(theta);
23
24 b1 = sin(phi)*sin(theta)*cos(psi)-cos(phi)*sin(psi);
25 b2 = sin(phi)*sin(theta)*sin(psi)+cos(phi)*cos(psi);
26 b3 = sin(phi)*cos(theta);
27
28 c1 = cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi);
29 c2 = cos(phi)*sin(theta)*sin(psi)-sin(phi)*cos(psi);
30 c3 = cos(phi)*cos(theta);
31
32 R321 = [a1 a2 a3; b1 b2 b3; c1 c2 c3];
33
34 end
```