University of Colorado - Boulder

ASEN 3128: Aircraft Dynamics

September 25, 2020

ASEN 3128 LAB 2: Simulate EOM

Author:
Luke Engelken[a]

Author:
Connor O'Reilly[b]

Author:
Ajay Dhindsa[c]

Professor:
Professor Eric Frew

[a]SID: 107165781
[b]SID: 107054811
[c]SID: 108423004

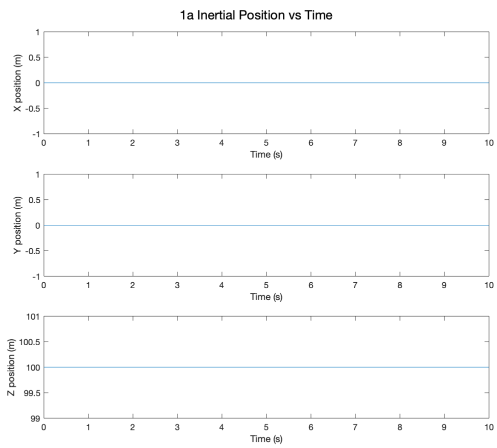
Ann and H.J. Smead
Aerospace Engineering Sciences
UNIVERSITY OF COLORADO BOULDER

Contents

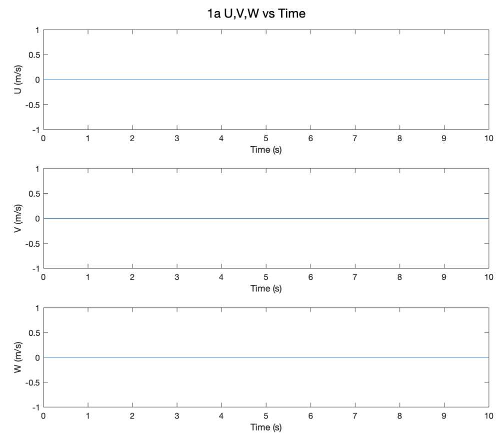
I. Problem 1

In order to determine the trim thrusts for the motors on the quadrotor, a simulation was created to visualize the state vectors of the quadrotor over time. the simulation was creating using ODE45 in Matlab where initial conditions are given in a state vector and the change in the state vector's initial conditions are determined during the ODE45 function call.

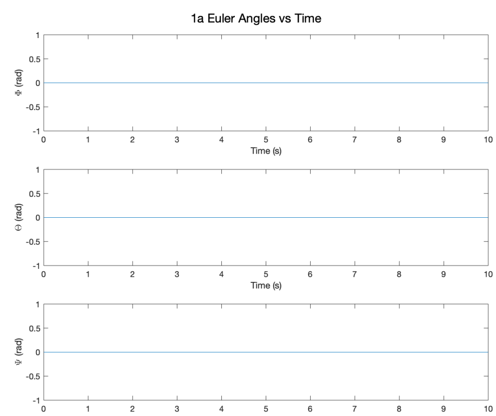
In order to determine the trim thrusts values for steady hovering flight, the quadrotor is placed at an initial height of 100 m off the ground. The motors on the quadrotor are then set to each produce a force equal to one fourth the weight of the quadrotor, such that $F_1 = F_2 = F_3 = F_4 = \frac{m_{quadrotor}g}{4} = 0.1668$ N. Using these initial conditions we have determined the trim thrusts for the motors on the quadrotor for steady hovering flight as shown below in Figures 1 - 4.



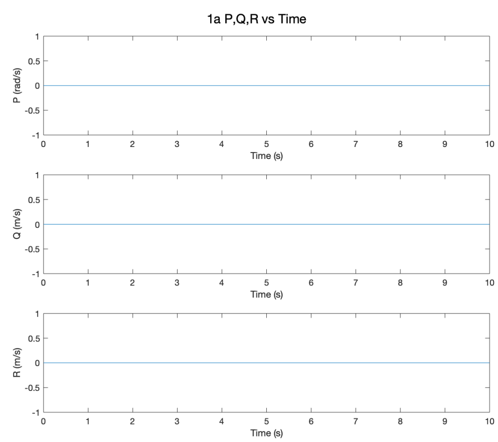
(a) Inertial Position VS Time



(b) Inertial Velocity VS Time



(c) Euler Angles VS Time



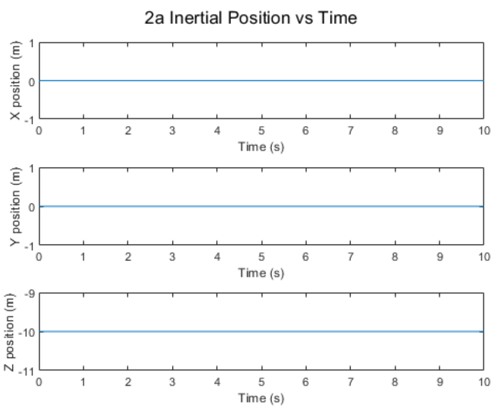
(d) Angular Velocity VS Time

Fig. 1 Quadrotor State Vector Over Time

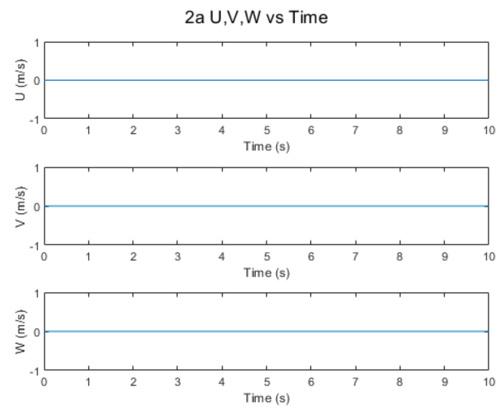
II. Problem 2

A.

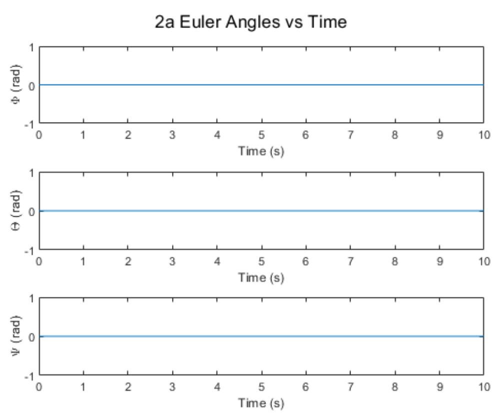
This simulation is calculated after adding in aerodynamics forces and moments into the calculation. As seen below, adding these extra forces does not alter the trim state for steady hover.



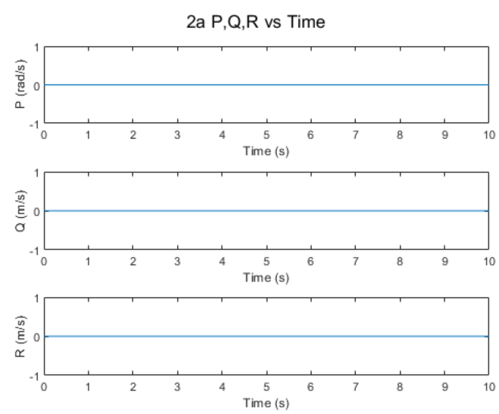
(a) Inertial Position VS Time



(b) Inertial Velocity VS Time



(c) Euler Angles VS Time

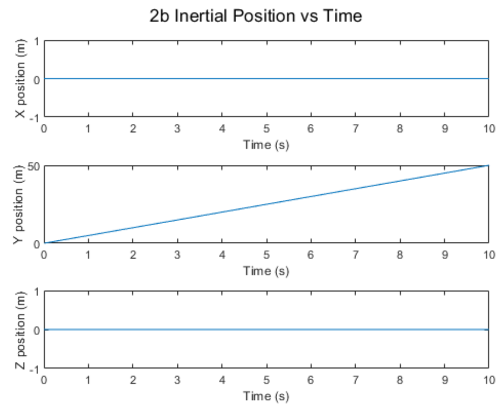


(d) Angular Velocity VS Time

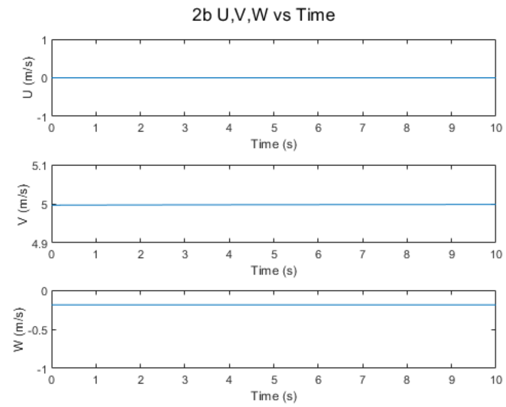
Fig. 2 Quadrotor State Vector Over Time With Aerodynamic Forces

B.

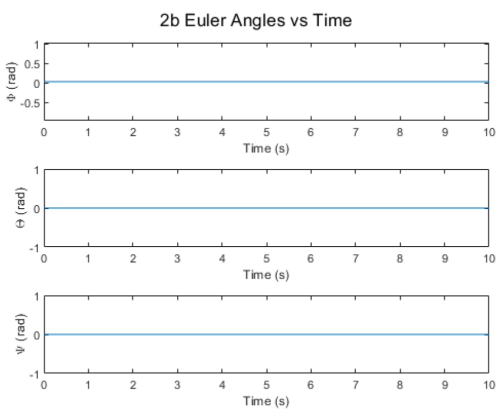
The quadrotor is now translating East with a constant velocity of 5 m/s, while maintaining a yaw angle of 0°. We calculated the trim state to be $\phi = 0.0375$ radians and $F_1 = F_2 = F_3 = F_4 = -0.1669$ Seen below are the resulting trim states given these new initial conditions.



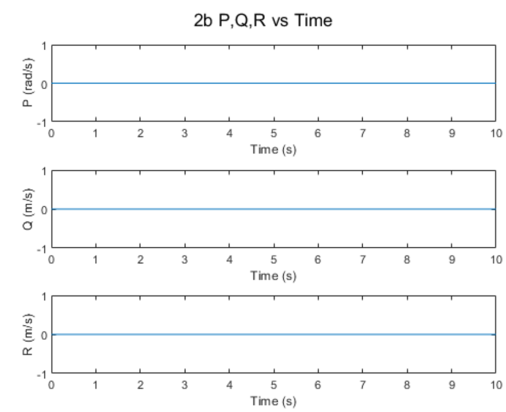
(a) Inertial Position VS Time



(b) Inertial Velocity VS Time



(c) Euler Angles VS Time

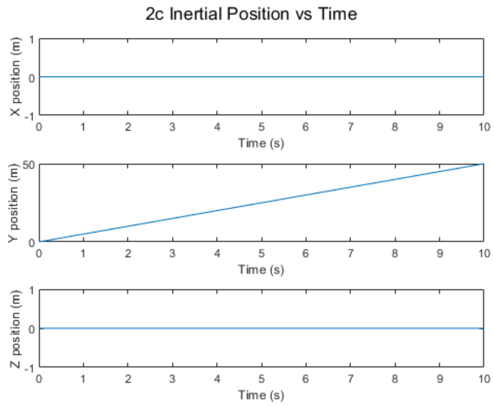


(d) Angular Velocity VS Time

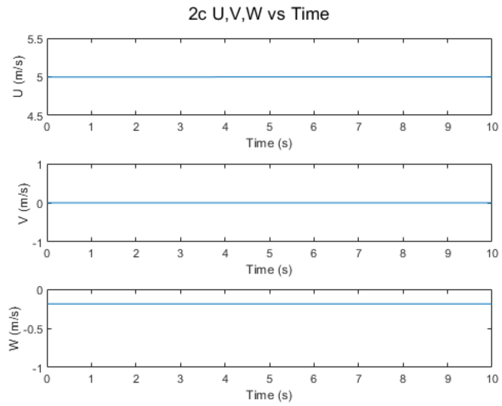
Fig. 3 Quadrotor State Vector Over Time Translating East

C.

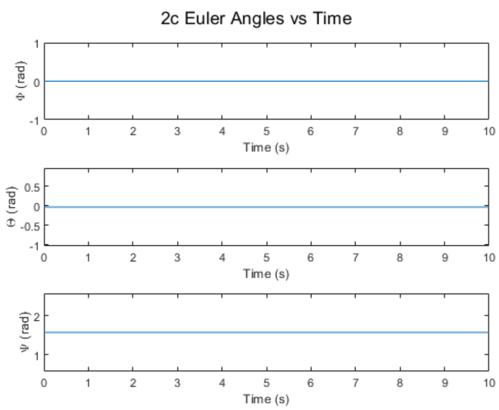
Finally, the quadrotor is now required to be maintaining a yaw of 90°while still translating East at 5 m/s. We calculated the trim state to be $\theta = -0.0375$ radians and $F_1 = F_2 = F_3 = F_4 = -0.1669$ Below is the simulation verifying the trim state needed.



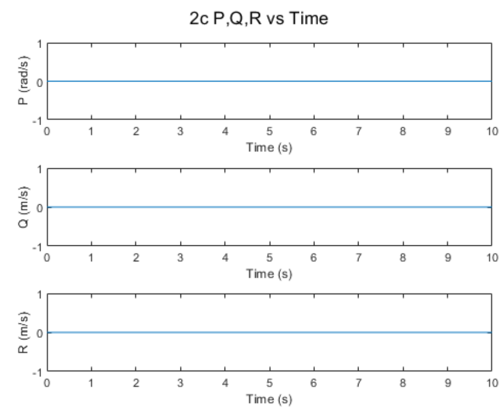
(a) Inertial Position VS Time



(b) Inertial Velocity VS Time



(c) Euler Angles VS Time

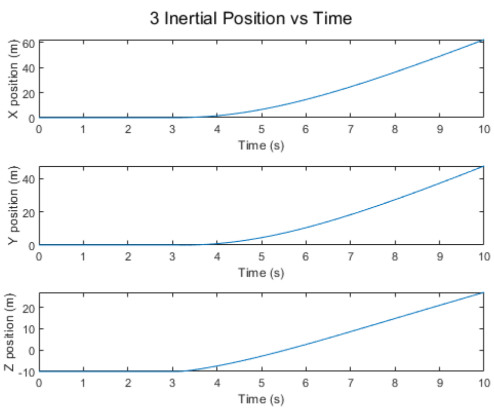


(d) Angular Velocity VS Time

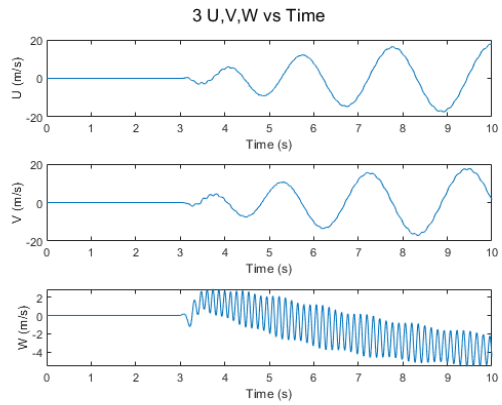
Fig. 4 Quadrotor State Vector Over Time Translating East With Yaw of 90°

III. Problem 3

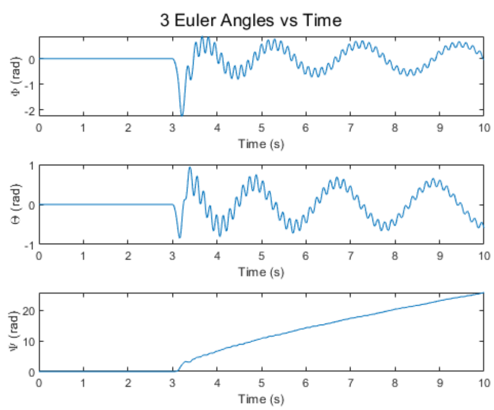
Finally, the question is presented of whether or not steady hovering flight is stable for the quadrotor. In order to determine this, we applied an impulse lasting 0.5 second after 3 seconds of steady hovering in order to simulate motor failure to determine the quadrotor's stability. We implemented this by setting f4 = 0 N for 0.5 seconds in our ODE45 function call. Below shows the state vector of the quadrotor over the duration in which the impulse is applied to the motor.



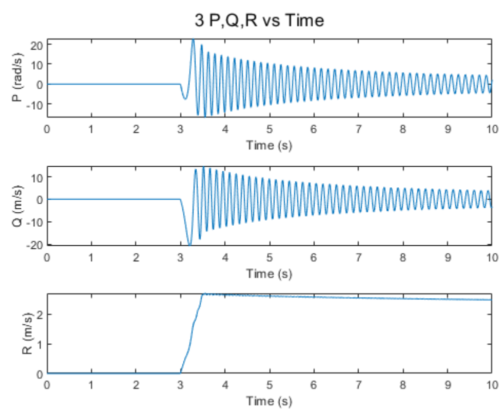
(a) Inertial Position VS Time



(b) Inertial Velocity VS Time



(c) Euler Angles VS Time



(d) Angular Velocity VS Time

Fig. 5 Quadrotor State Vector Over Time Stability Response

Team Member Participation Table

	Plan	Model	Experiment	Results	Report	Code	ACK
Luke	2	1	1	2	1	1	LE
Ajay	1	2	1	1	2	1	AD
Connor	1	1	2	1	1	2	CO

IV. Appendix: MATLAB Code

1. Main

```
% Luke Engelken
% ASEN 3128
% prob1.m
% Created: 9/11/20

%To run, in command window type "prob(<insert quetsion number)" and you
%will see plots
%ex: prob('1a')

function [] = drone(prob)
    %% Declare Constants
    m = 0.068; %mass of drone (kg)
    r = 0.06; %radial distance from cg to motor (m)
    km = 0.0024; %control moment coefficient (N*m/N)
    Ix = 6.8e-5; %x-axis moment of inertia (kg*m^2)
    Iy = 9.2e-5; %y-axis moment of inertia (kg*m^2)
    Iz = 1.35e-3; %z-axis moment of inertia (kg*m^2)
    v = 1e-3; %aerodynamic force coefficient (N/(m/s^2))
    mu = 2e-6; %N*m/(rad/s^2)
    g = 9.81; %m/s^2
    %xstate vector: [x; y; z; phi; theta; psi; xdot; ydot; zdot; phidot;
    %thetadot; psidot] (integral of eom state vector)
    tspan = [0 10];
    switch prob
        case '1a'
            %Prob 1a - steady state hover, no drag
            f1 = g*m/4; %Assume -force up
            f2 = f1;
            f3 = f1;
            f4 = f1;
            pinit = [0; 0; 100; 0; 0; 0; 0; 0; 0; 0; 0; 0];
            [tx,x1] = ode45(@(t,x) objectEOMnodrag(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4),tspan,pinit);
        case '2a'
            %Prob 2a - steady state hover, with drag
            f1 = g*m/4; %Assume -force up
            f2 = f1;
            f3 = f1;
            f4 = f1;
            pinit = [0; 0; -10; 0; 0; 0; 0; 0; 0; 0; 0; 0];
            [tx,x1] = ode45(@(t,x) objectEOM(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4),tspan,pinit);
        case '2b'
            %Prob 2b - constant y velocity = 5m/s, psi = 0
            syms phi
            %eqn = 5*v*5*cos(phi)+m*g*sin(phi)^3+m*g*sin(phi)*cos(phi)^2 == 0;
            eqn = atan(phi) == v*25/(m*g);
            phi = solve(eqn);
            phi = eval(phi(1,1)); %3.1041
            %phi = 2.12;
            pinit = [0; 0; 0; phi; 0; 0; 0; 5*cos(phi); -5*sin(phi); 0; 0; 0];
            Zc = m*g*(cos(pinit(4))+sin(pinit(4))^2/cos(pinit(4))); %-0.667080298221064
            %Zc = 25*v/sin(phi); %-.6681
            %Zc = .6665;
            f1 = Zc/4;
```

```
            f2 = f1;
            f3 = f2;
            f4 = f3;
            [tx,x1] = ode45(@(t,x) objectEOM(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4),tspan,pinit);
        case '2c'
            %Prob 2c - constant y velocity = -5m/s, psi = 90deg
             syms phi
            %eqn = 5*v*5*cos(phi)+m*g*sin(phi)^3+m*g*sin(phi)*cos(phi)^2 == 0;
            eqn = atan(phi) == v*25/(m*g);
            phi = solve(eqn);
            phi = eval(phi(1,1)); %3.1041
            %phi = 2.12;
            pinit = [0; 0; 0; 0; -phi; pi/2; 5*cos(phi); 0; -5*sin(phi); 0; 0; 0];
            Zc = m*g*(cos(pinit(5))+sin(pinit(5))^2/cos(pinit(5))); %-0.667080298221064
            %Zc = 25*v/sin(phi); %-.6681
            %Zc = .6665;
            f1 = Zc/4;
            f2 = f1;
            f3 = f2;
            f4 = f3;
            [tx,x1] = ode45(@(t,x) objectEOM(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4),tspan,pinit);
        case '3'
            %Prob 2a - steady state hover, with drag
            f1 = g*m/4; %Assume -force up
            f2 = f1;
            f3 = f1;
            f4 = f1;
            pinit = [0; 0; -10; 0; 0; 0; 0; 0; 0; 0; 0; 0];
            [tx,x1] = ode45(@(t,x)
                objectEOM_disturb(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4),tspan,pinit);
    end

    xval = x1(:,1);
    yval = x1(:,2);
    zval = x1(:,3);
    phival = x1(:,4);
    thetaval = x1(:,5);
    psival = x1(:,6);
    uval = x1(:,7);
    vval = x1(:,8);
    wval = x1(:,9);
    pval = x1(:,10);
    qval = x1(:,11);
    rval = x1(:,12);

    figure(1)
    drone_plot(x1,tx,'pqr',prob);

    figure(2)
    drone_plot(x1,tx,'uvw',prob);

    figure(3)
    drone_plot(x1,tx,'euler',prob);

    figure(4)
    drone_plot(x1,tx,'xyz',prob);
end
```

2. Plotting

```
function [] = drone_plot(x1,tx,type,prob)
%DRONE_PLOT: Plots the 4 subplots of inertial position, Euler angles,
%body acceleration, and Euler rates
xval = x1(:,1);
    yval = x1(:,2);
    zval = x1(:,3);
    phival = x1(:,4);
    thetaval = x1(:,5);
    psival = x1(:,6);
    uval = x1(:,7);
    vval = x1(:,8);
    wval = x1(:,9);
    pval = x1(:,10);
    qval = x1(:,11);
    rval = x1(:,12);

switch type
    case 'pqr'
        subplot(3,1,1)
        plot(tx,pval);
        xlabel('Time (s)');
        ylabel('P (rad/s)');
        subplot(3,1,2)
        plot(tx,qval);
        xlabel('Time (s)');
        ylabel('Q (m/s)');
        subplot(3,1,3)
        plot(tx,rval);
        xlabel('Time (s)');
        ylabel('R (m/s)');
        sgtitle(sprintf('%s P,Q,R vs Time',prob));
    case 'uvw'
        subplot(3,1,1)
        plot(tx,uval);
        xlabel('Time (s)');
        ylabel('U (m/s)');
        if prob == '2c'
            axis([0 10 4.5 5.5]);
        end
        subplot(3,1,2)
        plot(tx,vval);
        xlabel('Time (s)');
        ylabel('V (m/s)');
        if prob == '2b'
            axis([0 10 4.9 5.1]);
        end
        subplot(3,1,3)
        plot(tx,wval);
        xlabel('Time (s)');
        ylabel('W (m/s)');
        if prob == '2b' | prob == '2c'
            axis([0 10 -1 0]);
        end
        sgtitle(sprintf('%s U,V,W vs Time',prob));
    case 'euler'
        subplot(3,1,1)
        plot(tx,phival);
```

```
        xlabel('Time (s)');
        ylabel('\Phi (rad)');
        subplot(3,1,2)
        plot(tx,thetaval);
        xlabel('Time (s)');
        ylabel('\Theta (rad)');
        subplot(3,1,3)
        plot(tx,psival);
        xlabel('Time (s)');
        ylabel('\Psi (rad)');
        sgtitle(sprintf('%s Euler Angles vs Time',prob));
    case 'xyz'
        subplot(3,1,1)
        plot(tx,xval);
        xlabel('Time (s)');
        ylabel('X position (m)');
        subplot(3,1,2)
        plot(tx,yval);
        xlabel('Time (s)');
        ylabel('Y position (m)');
        subplot(3,1,3)
        plot(tx,zval);
        if prob == '2b' | prob == '2c'
            axis([0 10 -1 1]);
        end
        xlabel('Time (s)');
        ylabel('Z position (m)');
        sgtitle(sprintf('%s Inertial Position vs Time',prob));


end
```

3. State Vector

```
function [xstate] = objectEOM(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4)
%
%Inputs:
% t = time
% x = 12-dimension state vector includes the inertial velocity in
%   inertial coordinates and the inertial position in inertial coordinates
%   [x; y; z; phi; theta; psi; u; v; w; p; q; r]
% m = mass of drone (kg)
% r = radius of frame from motor to cg
% km = control moment coefficient
% Ix,Iy,Iz = moments about axis
% v = drag coefficient
% mu = moment coefficient
% f1,f2,f3,f4 = forces from motors
%
%Outputs:
% xdot = 12-dimension state vector includes inertial velocity in inertial
%    coordinates and the inertial acceleration in inertial coordinates
%    [xdot; ydot; zdot; phidot; thetadot; psidot; udot; vdot; wdot;
%    pdot; qdot; rdot]
%
%Methodology: Use Newton's second law F=ma to calculate the acceleration
%   and velocity at each point in time for ode45 to integrate to find
%   position. Drag, gravity and motor thrust are only forces acting on drone.
```

```
%

g = 9.81;

%Get IV's
x1 = x(1);
y1 = x(2);
z1 = x(3);
phi1 = x(4);
theta1 = x(5);
psi1 = x(6);
u1 = x(7);
v1 = x(8);
w1 = x(9);
p1 = x(10);
q1 = x(11);
r1 = x(12);
Zc = -f1 - f2 - f3 - f4; %sum of 4 motor thrusts in -z direction

%Find Pdot(xdot ydot zdot) (earth fixed)
Pdot = R_eb(phi1,theta1,psi1,'rad')*[u1;v1;w1];

%Find Odot(thetadot phidot psidot)
Odot = T(phi1,theta1,psi1,'rad')*[p1;q1;r1];

%Get magnitude of velocity (B)
V_a = sqrt(u1^2 + v1^2 + w1^2);

%Calculate acceleration (body: Vb = [udot; vdot; wdot] components
%Vb = -w_b*V_b+f_b/m
Vb(1) = (r1*v1-q1*w1)+g*(-sin(theta1))+1/m*(-v*V_a*u1);
Vb(2) = (p1*w1-r1*u1)+g*(cos(theta1)*sin(phi1))+1/m*(-v*V_a*v1);
Vb(3) = (q1*u1-p1*v1)+g*(cos(theta1)*cos(phi1))+1/m*(-v*V_a*w1)+1/m*(Zc);

%Calculate L, M, N
m_a = -mu*sqrt(p1^2+q1^2+r1^2)*[p1; q1; r1];

%Calculate m_ctl
m_ctl(1) = (r/sqrt(2))*(-f1-f2+f3+f4);
m_ctl(2) = (r/sqrt(2))*(f1-f2-f3+f4);
m_ctl(3) = (r/sqrt(2))*(f1-f2+f3-f4);

%Calculate omega_dot(pdot,qdot,rdot)
omega_dot(1) = (Iy-Iz)/(Ix)*q1*r1 + 1/Ix*m_a(1) + 1/Ix*m_ctl(1);
omega_dot(2) = (Iz-Ix)/(Iy)*p1*r1 + 1/Iy*m_a(2) + 1/Iy*m_ctl(2);
omega_dot(3) = (Ix-Iy)/(Iz)*p1*q1 + 1/Iz*m_a(3) + 1/Iz*m_ctl(3);

%Put back into ode45
xstate(1) = Pdot(1); %xdot
xstate(2) = Pdot(2); %ydot
xstate(3) = Pdot(3); %zdot
xstate(4) = Odot(1); %phidot
xstate(5) = Odot(2); %thetadot
xstate(6) = Odot(3); %psidot
xstate(7) = Vb(1); %udot
xstate(8) = Vb(2); %vdot
xstate(9) = Vb(3); %wdot
xstate(10) = omega_dot(1); %pdot
xstate(11) = omega_dot(2); %qdot
```

```
xstate(12) = omega_dot(3); %rdot
xstate = xstate';
end
```

4. State Vector No Drag

```
function [xstate] = objectEOM(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4)
%
%Inputs:
% t = time
% x = 12-dimension state vector includes the inertial velocity in
%   inertial coordinates and the inertial position in inertial coordinates
%   [x; y; z; phi; theta; psi; u; v; w; p; q; r]
% m = mass of drone (kg)
% r = radius of frame from motor to cg
% km = control moment coefficient
% Ix,Iy,Iz = moments about axis
% v = drag coefficient
% mu = moment coefficient
% f1,f2,f3,f4 = forces from motors
%
%Outputs:
% xdot = 12-dimension state vector includes inertial velocity in inertial
%    coordinates and the inertial acceleration in inertial coordinates
%    [xdot; ydot; zdot; phidot; thetadot; psidot; udot; vdot; wdot;
%    pdot; qdot; rdot]
%
%Methodology: Use Newton's second law F=ma to calculate the acceleration
%   and velocity at each point in time for ode45 to integrate to find
%   position. Drag, gravity and motor thrust are only forces acting on drone.
%

g = 9.81;

%Get IV's
x1 = x(1);
y1 = x(2);
z1 = x(3);
phi1 = x(4);
theta1 = x(5);
psi1 = x(6);
u1 = x(7);
v1 = x(8);
w1 = x(9);
p1 = x(10);
q1 = x(11);
r1 = x(12);
Zc = -f1 - f2 - f3 - f4; %sum of 4 motor thrusts in -z direction

%Find Pdot(xdot ydot zdot) (earth fixed)
Pdot = R_eb(phi1,theta1,psi1,'rad')*[u1;v1;w1];

%Find Odot(thetadot phidot psidot)
Odot = T(phi1,theta1,psi1,'rad')*[p1;q1;r1];

%Get magnitude of velocity (B)
V_a = sqrt(u1^2 + v1^2 + w1^2);

```

```
%Calculate acceleration (body: Vb = [udot; vdot; wdot] components
%Vb = -w_b*V_b+f_b/m
Vb(1) = (r1*v1-q1*w1)+g*(-sin(theta1));
Vb(2) = (p1*w1-r1*u1)+g*(cos(theta1)*sin(phi1));
Vb(3) = (q1*u1-p1*v1)+g*(cos(theta1)*cos(phi1))+1/m*(Zc);

%Calculate L, M, N
m_a = -mu*sqrt(p1^2+q1^2+r1^2)*[p1; q1; r1];

%Calculate m_ctl
m_ctl(1) = (r/sqrt(2))*(-f1-f2+f3+f4);
m_ctl(2) = (r/sqrt(2))*(f1-f2-f3+f4);
m_ctl(3) = (r/sqrt(2))*(f1-f2+f3-f4);

%Calculate omega_dot(pdot,qdot,rdot)
omega_dot(1) = (Iy-Iz)/(Ix)*q1*r1 + 1/Ix*m_a(1) + 1/Ix*m_ctl(1);
omega_dot(2) = (Iz-Ix)/(Iy)*p1*r1 + 1/Iy*m_a(2) + 1/Iy*m_ctl(2);
omega_dot(3) = (Ix-Iy)/(Iz)*p1*q1 + 1/Iz*m_a(3) + 1/Iz*m_ctl(3);

%Put back into ode45
xstate(1) = Pdot(1); %xdot
xstate(2) = Pdot(2); %ydot
xstate(3) = Pdot(3); %zdot
xstate(4) = Odot(1); %phidot
xstate(5) = Odot(2); %thetadot
xstate(6) = Odot(3); %psidot
xstate(7) = Vb(1); %udot
xstate(8) = Vb(2); %vdot
xstate(9) = Vb(3); %wdot
xstate(10) = omega_dot(1); %pdot
xstate(11) = omega_dot(2); %qdot
xstate(12) = omega_dot(3); %rdot
xstate = xstate';
end
```

5. State Vector Disturbed

```
function [xstate] = objectEOM(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4)
%
%Inputs:
% t = time
% x = 12-dimension state vector includes the inertial velocity in
%  inertial coordinates and the inertial position in inertial coordinates
%  [x; y; z; phi; theta; psi; u; v; w; p; q; r]
% m = mass of drone (kg)
% r = radius of frame from motor to cg
% km = control moment coefficient
% Ix,Iy,Iz = moments about axis
% v = drag coefficient
% mu = moment coefficient
% f1,f2,f3,f4 = forces from motors
%
%Outputs:
% xdot = 12-dimension state vector includes inertial velocity in inertial
%   coordinates and the inertial acceleration in inertial coordinates
%   [xdot; ydot; zdot; phidot; thetadot; psidot; udot; vdot; wdot;
%   pdot; qdot; rdot]
%
```

```
%Methodology: Use Newton's second law F=ma to calculate the acceleration
%   and velocity at each point in time for ode45 to integrate to find
%   position. Drag, gravity and motor thrust are only forces acting on drone.
%

g = 9.81;

%Get IV's
x1 = x(1);
y1 = x(2);
z1 = x(3);
phi1 = x(4);
theta1 = x(5);
psi1 = x(6);
u1 = x(7);
v1 = x(8);
w1 = x(9);
p1 = x(10);
q1 = x(11);
r1 = x(12);

if (t > 3 && t < 3.5)
    f4 = 0;
end

Zc = -f1 - f2 - f3 - f4; %sum of 4 motor thrusts in -z direction

%Find Pdot(xdot ydot zdot) (earth fixed)
Pdot = R_eb(phi1,theta1,psi1,'rad')*[u1;v1;w1];

%Find Odot(thetadot phidot psidot)
Odot = T(phi1,theta1,psi1,'rad')*[p1;q1;r1];

%Get magnitude of velocity (B)
V_a = sqrt(u1^2 + v1^2 + w1^2);

%Calculate acceleration (body: Vb = [udot; vdot; wdot] components
%Vb = -w_b*V_b+f_b/m
Vb(1) = (r1*v1-q1*w1)+g*(-sin(theta1))+1/m*(-v*V_a*u1);
Vb(2) = (p1*w1-r1*u1)+g*(cos(theta1)*sin(phi1))+1/m*(-v*V_a*v1);
Vb(3) = (q1*u1-p1*v1)+g*(cos(theta1)*cos(phi1))+1/m*(-v*V_a*w1)+1/m*(Zc);

%Calculate L, M, N
m_a = -mu*sqrt(p1^2+q1^2+r1^2)*[p1; q1; r1];

%Calculate m_ctl
m_ctl(1) = (r/sqrt(2))*(-f1-f2+f3+f4);
m_ctl(2) = (r/sqrt(2))*(f1-f2-f3+f4);
m_ctl(3) = (r/sqrt(2))*(f1-f2+f3-f4);

%Calculate omega_dot(pdot,qdot,rdot)
omega_dot(1) = (Iy-Iz)/(Ix)*q1*r1 + 1/Ix*m_a(1) + 1/Ix*m_ctl(1);
omega_dot(2) = (Iz-Ix)/(Iy)*p1*r1 + 1/Iy*m_a(2) + 1/Iy*m_ctl(2);
omega_dot(3) = (Ix-Iy)/(Iz)*p1*q1 + 1/Iz*m_a(3) + 1/Iz*m_ctl(3);

%Put back into ode45
xstate(1) = Pdot(1); %xdot
xstate(2) = Pdot(2); %ydot
xstate(3) = Pdot(3); %zdot
```

```
xstate(4) = Odot(1); %phidot
xstate(5) = Odot(2); %thetadot
xstate(6) = Odot(3); %psidot
xstate(7) = Vb(1); %udot
xstate(8) = Vb(2); %vdot
xstate(9) = Vb(3); %wdot
xstate(10) = omega_dot(1); %pdot
xstate(11) = omega_dot(2); %qdot
xstate(12) = omega_dot(3); %rdot
xstate = xstate';
end
```

6. Rotation Matrix

```
function [REB] = R_eb(phi,theta,psi,units)
%switch if input is either rad or deg
switch units
    case 'deg'
        REB(1,1) = cosd(theta)*cosd(psi);
        REB(1,2) = cosd(theta)*sind(psi);
        REB(1,3) = -sind(theta);
        REB(2,1) = sind(phi)*sind(theta)*cosd(psi)-cosd(phi)*sind(psi);
        REB(2,2) = sind(phi)*sind(theta)*sind(psi)+cosd(phi)*cosd(psi);
        REB(2,3) = sind(phi)*cosd(theta);
        REB(3,1) = cosd(phi)*sind(theta)*cosd(psi)+sind(phi)*sind(psi);
        REB(3,2) = cosd(phi)*sind(theta)*sind(psi)-sind(phi)*cosd(psi);
        REB(3,3) = cosd(phi)*cosd(theta);
    case 'rad'
        phi = phi * (180/pi);
        theta = theta * (180/pi);
        psi = psi * (180/pi);
        REB(1,1) = cosd(theta)*cosd(psi);
        REB(1,2) = cosd(theta)*sind(psi);
        REB(1,3) = -sind(theta);
        REB(2,1) = sind(phi)*sind(theta)*cosd(psi)-cosd(phi)*sind(psi);
        REB(2,2) = sind(phi)*sind(theta)*sind(psi)+cosd(phi)*cosd(psi);
        REB(2,3) = sind(phi)*cosd(theta);
        REB(3,1) = cosd(phi)*sind(theta)*cosd(psi)+sind(phi)*sind(psi);
        REB(3,2) = cosd(phi)*sind(theta)*sind(psi)-sind(phi)*cosd(psi);
        REB(3,3) = cosd(phi)*cosd(theta);
end
REB = inv(REB);

end
```

7. Angular Rotation Matrix

```
function [Tmat] = T(phi,theta,psi,units)
%T Summary of this function goes here
%   Detailed explanation goes here
switch units
    case 'deg'
        Tmat(1,1) = 1;
        Tmat(1,2) = sind(phi)*tand(theta);
        Tmat(1,3) = cosd(phi)*tand(theta);
        Tmat(2,1) = 0;
        Tmat(2,2) = cosd(phi);
```

```
        Tmat(2,3) = -sind(phi);
        Tmat(3,1) = 0;
        Tmat(3,2) = sind(phi)*secd(theta);
        Tmat(3,3) = cosd(phi)*secd(theta);

    case 'rad'
        phi = phi * (180/pi);
        theta = theta * (180/pi);
        psi = psi * (180/pi);
        Tmat(1,1) = 1;
        Tmat(1,2) = sind(phi)*tand(theta);
        Tmat(1,3) = cosd(phi)*tand(theta);
        Tmat(2,1) = 0;
        Tmat(2,2) = cosd(phi);
        Tmat(2,3) = -sind(phi);
        Tmat(3,1) = 0;
        Tmat(3,2) = sind(phi)*secd(theta);
        Tmat(3,3) = cosd(phi)*secd(theta);
end
```