

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

Professor:

Professor ERIC FREW

Author:

AJAY DHINDSA^c

^aSID: 107165781

^bSID: 107054811

^cSID: 108423004



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

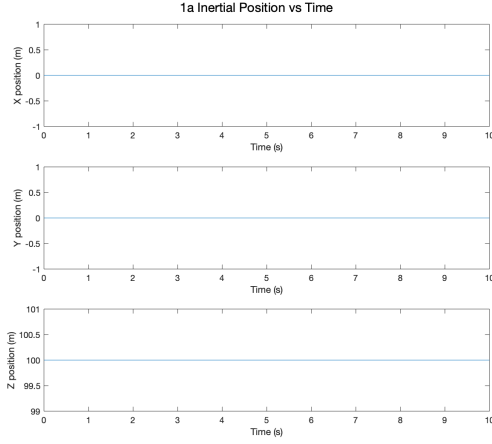
Contents

| | |
|---|----------|
| I Problem 1 | 2 |
| II Problem 2 | 3 |
| II.A | 3 |
| II.B | 4 |
| II.C | 5 |
| III Problem 3 | 6 |
| IV Appendix: MATLAB Code | 7 |
| IV..1 Main | 7 |
| IV..2 Plotting | 9 |
| IV..3 State Vector | 10 |
| IV..4 State Vector No Drag | 12 |
| IV..5 State Vector Disturbed | 13 |
| IV..6 Rotation Matrix | 15 |
| IV..7 Angular Rotation Matrix | 15 |

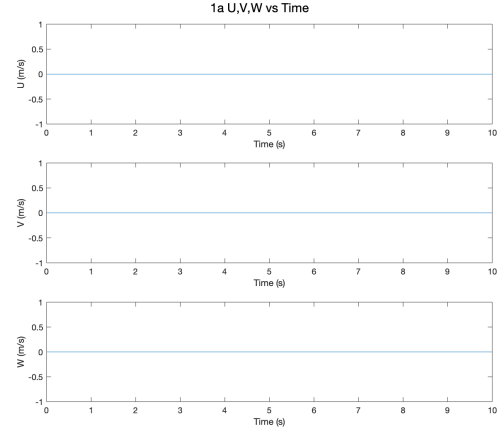
I. Problem 1

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.

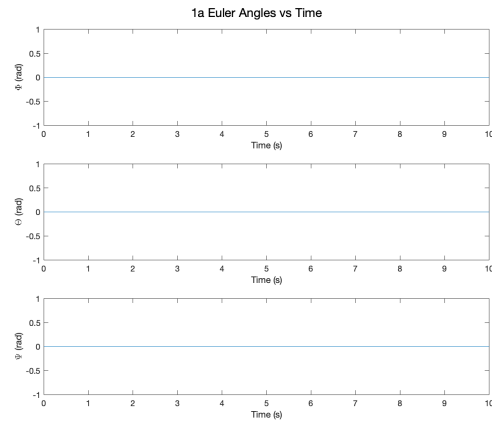
In order to determine the trim thrusts values for steady hovering flight, the quadrotor is placed at an initial height of 100 m off the ground. The motors on the quadrotor are then set to each produce a force equal to one fourth the weight of the quadrotor, such that $F_1 = F_2 = F_3 = F_4 = \frac{m_{quadrotor}g}{4}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



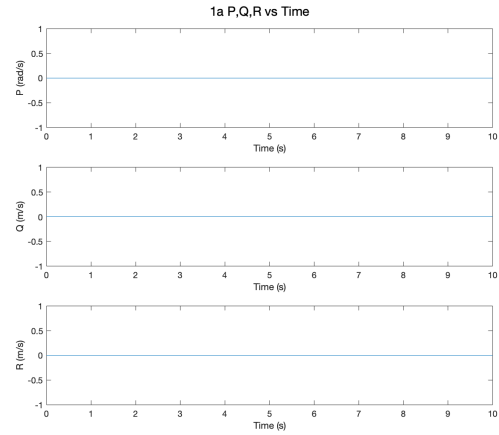
(a) Inertial Position VS Time



(b) Inertial Velocity VS Time



(c) Euler Angles VS Time



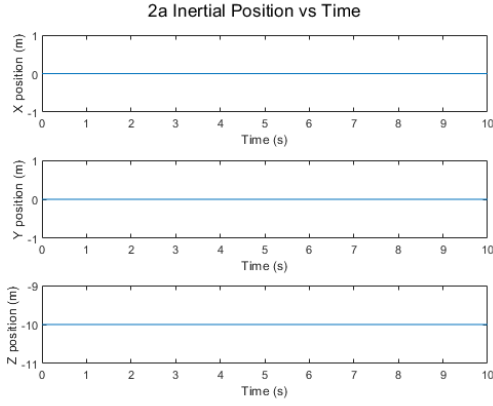
(d) Angular Velocity VS Time

Fig. 1 Quadrotor State Vector Over Time

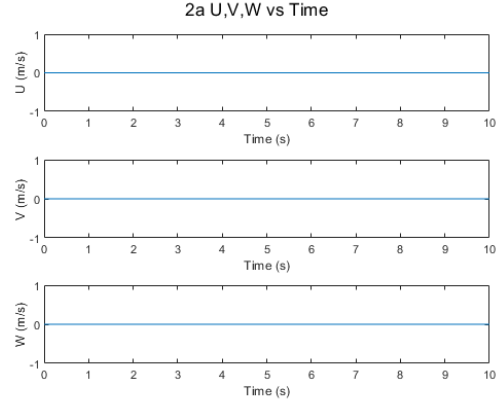
II. Problem 2

A.

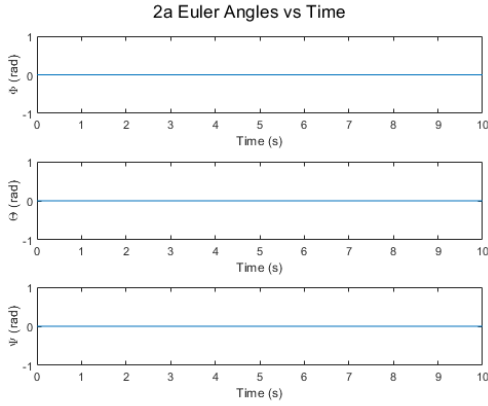
This simulation is calculated after adding in aerodynamics forces and moments into the calculation. As seen below, adding these extra forces does not alter the trim state for steady hover.



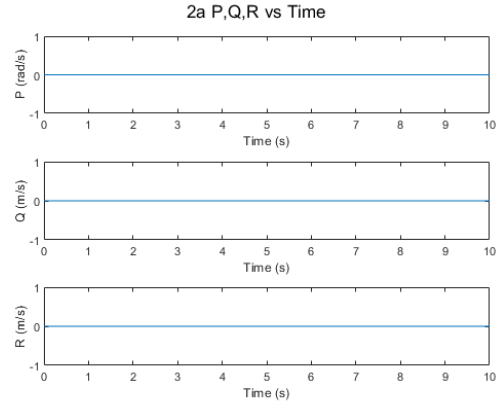
(a) Inertial Position VS Time



(b) Inertial Velocity VS Time



(c) Euler Angles VS Time

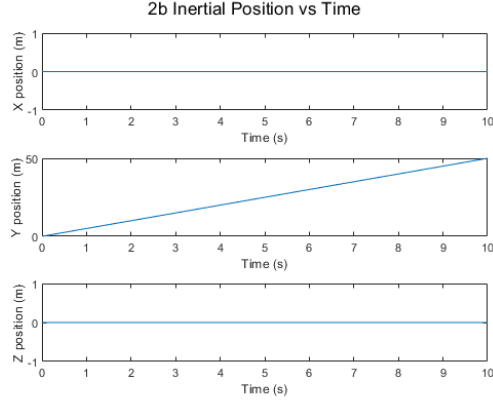


(d) Angular Velocity VS Time

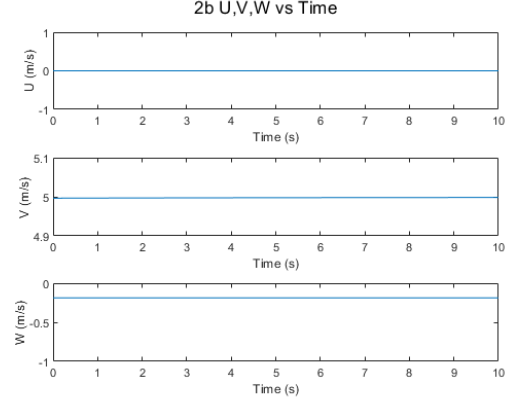
Fig. 2 Quadrotor State Vector Over Time With Aerodynamic Forces

B.

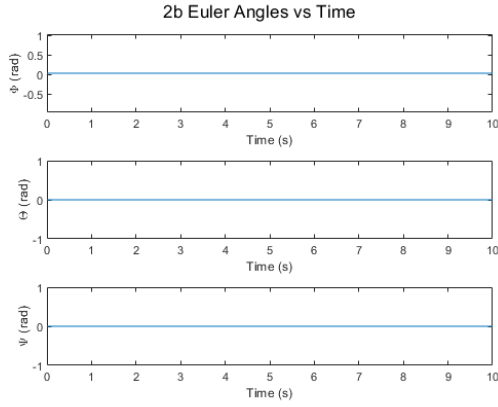
The quadrotor is now translating East with a constant velocity of 5 m/s, while maintaining a yaw angle of 0° . We calculated the trim state to be $\phi = 0.0375$ radians and $F_1 = F_2 = F_3 = F_4 = -0.1669$. Seen below are the resulting trim states given these new initial conditions.



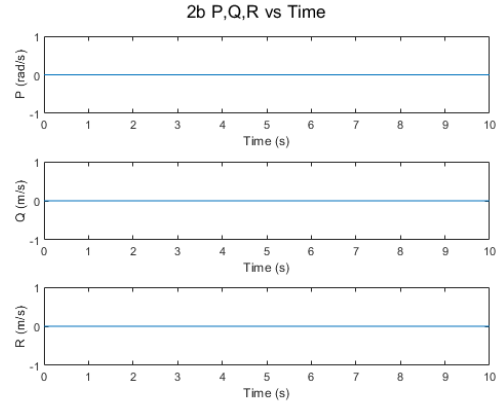
(a) Inertial Position VS Time



(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.

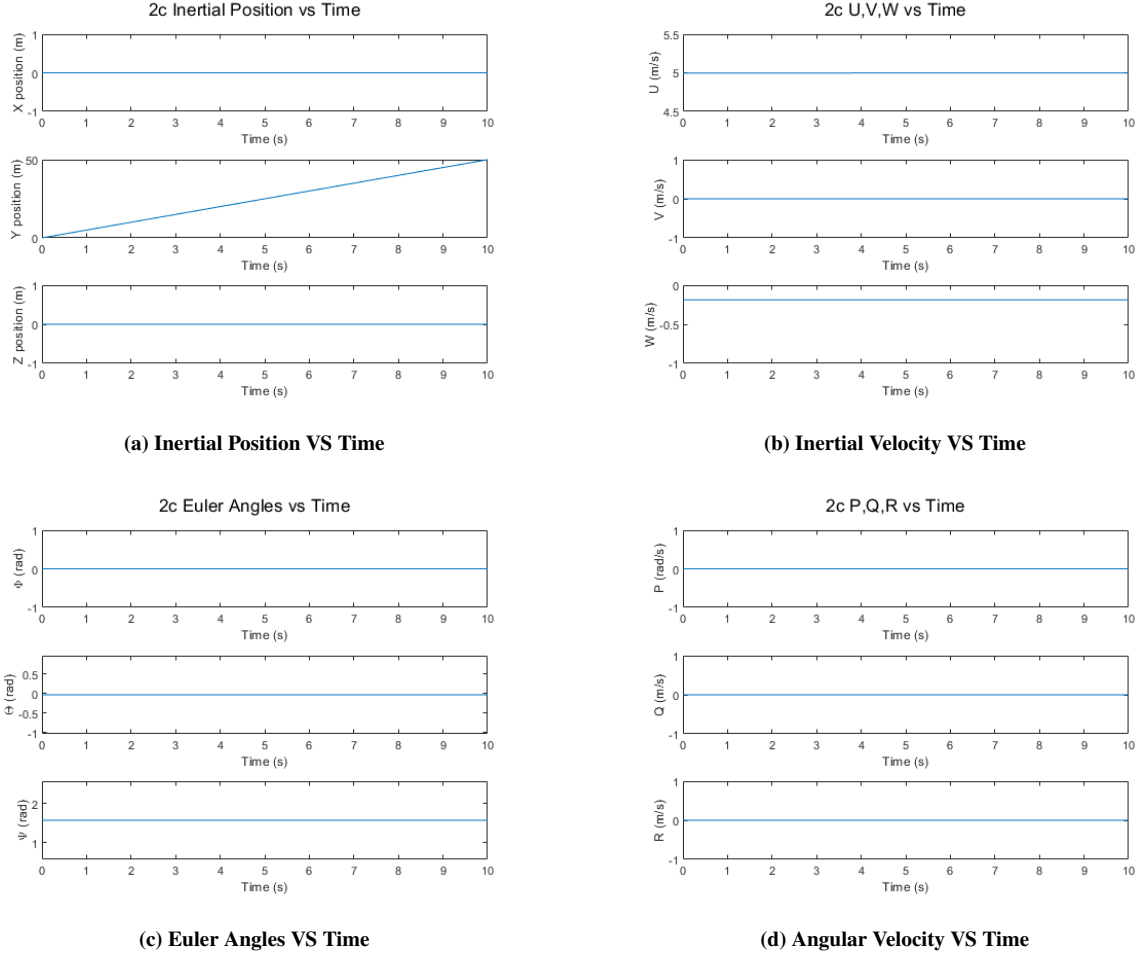


Fig. 4 Quadrotor State Vector Over Time Translating East With Yaw of 90°

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 $f_4 = 0$ 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.

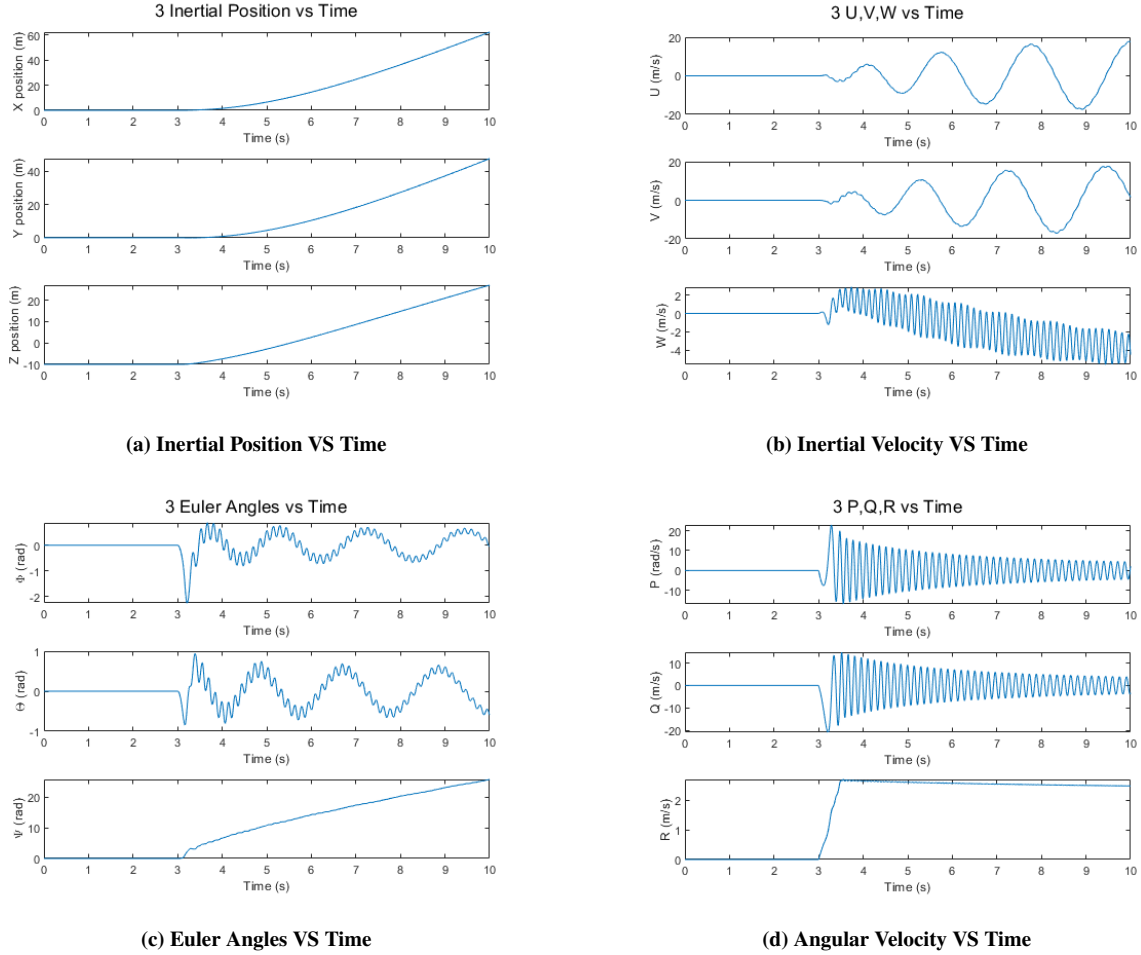


Fig. 5 Quadrotor State Vector Over Time Stability Response

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

```
1 % Luke Engelken
2 % ASEN 3128
3 % probl.m
4 % Created: 9/11/20
5
6 %To run, in command window type "prob(<insert quetsion number>)" and you
7 %will see plots
8 %ex: prob('1a')
9
10 function [] = drone(prob)
11     %% Declare Constants
12     m = 0.068; %mass of drone (kg)
13     r = 0.06; %radial distance from cg to motor (m)
14     km = 0.0024; %control moment coefficient (N*m/N)
15     Ix = 6.8e-5; %x-axis moment of inertia (kg*m^2)
16     Iy = 9.2e-5; %y-axis moment of inertia (kg*m^2)
17     Iz = 1.35e-3; %z-axis moment of inertia (kg*m^2)
18     v = 1e-3; %aerodynamic force coefficient (N/(m/s^2))
19     mu = 2e-6; %N*m/(rad/s^2)
20     g = 9.81; %m/s^2
21     %xstate vector: [x; y; z; phi; theta; psi; xdot; ydot; zdot; phidot;
22     %thetadot; psidot] (integral of eom state vector)
23     tspan = [0 10];
24     switch prob
25         case '1a'
26             %Prob 1a - steady state hover, no drag
27             f1 = g*m/4; %Assume -force up
28             f2 = f1;
29             f3 = f1;
30             f4 = f1;
31             pinit = [0; 0; 100; 0; 0; 0; 0; 0; 0; 0; 0; 0];
32             [tx,x1] = ode45(@(t,x)
33                 objectEOMnodrag(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4),tspan,pinit);
34         case '2a'
35             %Prob 2a - steady state hover, with drag
36             f1 = g*m/4; %Assume -force up
37             f2 = f1;
38             f3 = f1;
39             f4 = f1;
40             pinit = [0; 0; -10; 0; 0; 0; 0; 0; 0; 0; 0; 0];
41             [tx,x1] = ode45(@(t,x) objectEOM(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4),tspan,pinit);
42         case '2b'
43             %Prob 2b - constant y velocity = 5m/s, psi = 0
44             syms phi
45             %eqn = 5*v*5*cos(phi)+m*g*sin(phi)^3+m*g*sin(phi)*cos(phi)^2 == 0;
46             eqn = atan(phi) == v*25/(m*g);
47             phi = solve(eqn);
48             phi = eval(phi(1,1)); %3.1041
49             %phi = 2.12;
50             pinit = [0; 0; 0; phi; 0; 0; 0; 5*cos(phi); -5*sin(phi); 0; 0; 0];
51             Zc = m*g*(cos(pinit(4))+sin(pinit(4))^2/cos(pinit(4))); %-0.667080298221064
52             %Zc = 25*v/sin(phi); %-.6681
53             %Zc = .6665;
54             f1 = Zc/4;
```



```

54     f2 = f1;
55     f3 = f2;
56     f4 = f3;
57     [tx,x1] = ode45(@(t,x) objectEOM(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4),tspan,pinit);
58     case '2c'
59         %Prob 2c - constant y velocity = -5m/s, psi = 90deg
60         syms phi
61         %eqn = 5*v*5*cos(phi)+m*g*sin(phi)^3+m*g*sin(phi)*cos(phi)^2 == 0;
62         eqn = atan(phi) == v*25/(m*g);
63         phi = solve(eqn);
64         phi = eval(phi(1,1)); %3.1041
65         %phi = 2.12;
66         pinit = [0; 0; 0; 0; -phi; pi/2; 5*cos(phi); 0; -5*sin(phi); 0; 0; 0];
67         Zc = m*g*(cos(pinit(5))+sin(pinit(5))^2/cos(pinit(5))); %-0.667080298221064
68         %Zc = 25*v/sin(phi); %-.6681
69         %Zc = .6665;
70         f1 = Zc/4;
71         f2 = f1;
72         f3 = f2;
73         f4 = f3;
74         [tx,x1] = ode45(@(t,x) objectEOM(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4),tspan,pinit);
75     case '3'
76         %Prob 2a - steady state hover, with drag
77         f1 = g*m/4; %Assume -force up
78         f2 = f1;
79         f3 = f1;
80         f4 = f1;
81         pinit = [0; 0; -10; 0; 0; 0; 0; 0; 0; 0; 0; 0];
82         [tx,x1] = ode45(@(t,x)
            objectEOM_disturb(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4),tspan,pinit);
83     end
84
85     xval = x1(:,1);
86     yval = x1(:,2);
87     zval = x1(:,3);
88     phival = x1(:,4);
89     thetaval = x1(:,5);
90     psival = x1(:,6);
91     uval = x1(:,7);
92     vval = x1(:,8);
93     wval = x1(:,9);
94     pval = x1(:,10);
95     qval = x1(:,11);
96     rval = x1(:,12);
97
98     figure(1)
99     drone_plot(x1,tx,'pqr',prob);
100
101     figure(2)
102     drone_plot(x1,tx,'uvw',prob);
103
104     figure(3)
105     drone_plot(x1,tx,'euler',prob);
106
107     figure(4)
108     drone_plot(x1,tx,'xyz',prob);
109 end

```

2. Plotting

```
1 function [] = drone_plot(x1,tx,type,prob)
2 %DRONE_PLOT: Plots the 4 subplots of inertial position, Euler angles,
3 %body acceleration, and Euler rates
4 xval = x1(:,1);
5 yval = x1(:,2);
6 zval = x1(:,3);
7 phival = x1(:,4);
8 thetaval = x1(:,5);
9 psival = x1(:,6);
10 uval = x1(:,7);
11 vval = x1(:,8);
12 wval = x1(:,9);
13 pval = x1(:,10);
14 qval = x1(:,11);
15 rval = x1(:,12);
16
17 switch type
18     case 'pqr'
19         subplot(3,1,1)
20         plot(tx,pval);
21         xlabel('Time (s)');
22         ylabel('P (rad/s)');
23         subplot(3,1,2)
24         plot(tx,qval);
25         xlabel('Time (s)');
26         ylabel('Q (m/s)');
27         subplot(3,1,3)
28         plot(tx,rval);
29         xlabel('Time (s)');
30         ylabel('R (m/s)');
31         sgttitle(sprintf('%s P,Q,R vs Time',prob));
32     case 'uvw'
33         subplot(3,1,1)
34         plot(tx,uval);
35         xlabel('Time (s)');
36         ylabel('U (m/s)');
37         if prob == '2c'
38             axis([0 10 4.5 5.5]);
39         end
40         subplot(3,1,2)
41         plot(tx,vval);
42         xlabel('Time (s)');
43         ylabel('V (m/s)');
44         if prob == '2b'
45             axis([0 10 4.9 5.1]);
46         end
47         subplot(3,1,3)
48         plot(tx,wval);
49         xlabel('Time (s)');
50         ylabel('W (m/s)');
51         if prob == '2b' | prob == '2c'
52             axis([0 10 -1 0]);
53         end
54         sgttitle(sprintf('%s U,V,W vs Time',prob));
55     case 'euler'
56         subplot(3,1,1)
57         plot(tx,phival);
```

```

58     xlabel('Time (s)');
59     ylabel('\Phi (rad)');
60     subplot(3,1,2)
61     plot(tx,thetaval);
62     xlabel('Time (s)');
63     ylabel('\Theta (rad)');
64     subplot(3,1,3)
65     plot(tx,psival);
66     xlabel('Time (s)');
67     ylabel('\Psi (rad)');
68     sgtitle(sprintf('%s Euler Angles vs Time',prob));
69 case 'xyz'
70     subplot(3,1,1)
71     plot(tx,xval);
72     xlabel('Time (s)');
73     ylabel('X position (m)');
74     subplot(3,1,2)
75     plot(tx,yval);
76     xlabel('Time (s)');
77     ylabel('Y position (m)');
78     subplot(3,1,3)
79     plot(tx,zval);
80     if prob == '2b' | prob == '2c'
81         axis([0 10 -1 1]);
82     end
83     xlabel('Time (s)');
84     ylabel('Z position (m)');
85     sgtitle(sprintf('%s Inertial Position vs Time',prob));
86
87 end
88

```

3. State Vector

```

1 function [xstate] = objectEOM(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4)
2 %
3 %Inputs:
4 % t = time
5 % x = 12-dimension state vector includes the inertial velocity in
6 % inertial coordinates and the inertial position in inertial coordinates
7 % [x; y; z; phi; theta; psi; u; v; w; p; q; r]
8 % m = mass of drone (kg)
9 % r = radius of frame from motor to cg
10 % km = control moment coefficient
11 % Ix,Iy,Iz = moments about axis
12 % v = drag coefficient
13 % mu = moment coefficient
14 % f1,f2,f3,f4 = forces from motors
15 %
16 %Outputs:
17 % xdot = 12-dimension state vector includes inertial velocity in inertial
18 % coordinates and the inertial acceleration in inertial coordinates
19 % [xdot; ydot; zdot; phidot; thetadot; psidot; udot; vdot; wdot;
20 % pdot; qdot; rdot]
21 %
22 %Methodology: Use Newton's second law F=ma to calculate the acceleration
23 % and velocity at each point in time for ode45 to integrate to find
24 % position. Drag, gravity and motor thrust are only forces acting on drone.

```

```

25 %
26
27 g = 9.81;
28
29 %Get IV's
30 x1 = x(1);
31 y1 = x(2);
32 z1 = x(3);
33 phi1 = x(4);
34 theta1 = x(5);
35 psi1 = x(6);
36 u1 = x(7);
37 v1 = x(8);
38 w1 = x(9);
39 p1 = x(10);
40 q1 = x(11);
41 r1 = x(12);
42 Zc = -f1 - f2 - f3 - f4; %sum of 4 motor thrusts in -z direction
43
44 %Find Pdot(xdot ydot zdot) (earth fixed)
45 Pdot = R_eb(phi1,theta1,psi1,'rad')*[u1;v1;w1];
46
47 %Find Odot(thetadot phidot psidot)
48 Odot = T(phi1,theta1,psi1,'rad')*[p1;q1;r1];
49
50 %Get magnitude of velocity (B)
51 V_a = sqrt(u1^2 + v1^2 + w1^2);
52
53 %Calculate acceleration (body: Vb = [udot; vdot; wdot] components
54 %Vb = -w_b*V_b+f_b/m
55 Vb(1) = (r1*v1-q1*w1)+g*(-sin(theta1))+1/m*(-v*V_a*u1);
56 Vb(2) = (p1*w1-r1*u1)+g*(cos(theta1)*sin(phi1))+1/m*(-v*V_a*v1);
57 Vb(3) = (q1*u1-p1*v1)+g*(cos(theta1)*cos(phi1))+1/m*(-v*V_a*w1)+1/m*(Zc);
58
59 %Calculate L, M, N
60 m_a = -mu*sqrt(p1^2+q1^2+r1^2)*[p1; q1; r1];
61
62 %Calculate m_ctl
63 m_ctl(1) = (r/sqrt(2))*(-f1-f2+f3+f4);
64 m_ctl(2) = (r/sqrt(2))*(f1-f2-f3+f4);
65 m_ctl(3) = (r/sqrt(2))*(f1-f2+f3-f4);
66
67 %Calculate omega_dot(pdot,qdot,rdot)
68 omega_dot(1) = (Iy-Iz)/(Ix)*q1*r1 + 1/Ix*m_a(1) + 1/Ix*m_ctl(1);
69 omega_dot(2) = (Iz-Ix)/(Iy)*p1*r1 + 1/Iy*m_a(2) + 1/Iy*m_ctl(2);
70 omega_dot(3) = (Ix-Iy)/(Iz)*p1*q1 + 1/Iz*m_a(3) + 1/Iz*m_ctl(3);
71
72 %Put back into ode45
73 xstate(1) = Pdot(1); %xdot
74 xstate(2) = Pdot(2); %ydot
75 xstate(3) = Pdot(3); %zdot
76 xstate(4) = Odot(1); %phidot
77 xstate(5) = Odot(2); %thetadot
78 xstate(6) = Odot(3); %psidot
79 xstate(7) = Vb(1); %udot
80 xstate(8) = Vb(2); %vdot
81 xstate(9) = Vb(3); %wdot
82 xstate(10) = omega_dot(1); %pdot
83 xstate(11) = omega_dot(2); %qdot

```

```

84 | xstate(12) = omega_dot(3); %rdot
85 | xstate = xstate';
86 | end

```

4. State Vector No Drag

```

1 | function [xstate] = objectEOM(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4)
2 | %
3 | %Inputs:
4 | % t = time
5 | % x = 12-dimension state vector includes the inertial velocity in
6 | % inertial coordinates and the inertial position in inertial coordinates
7 | % [x; y; z; phi; theta; psi; u; v; w; p; q; r]
8 | % m = mass of drone (kg)
9 | % r = radius of frame from motor to cg
10 | % km = control moment coefficient
11 | % Ix,Iy,Iz = moments about axis
12 | % v = drag coefficient
13 | % mu = moment coefficient
14 | % f1,f2,f3,f4 = forces from motors
15 | %
16 | %Outputs:
17 | % xdot = 12-dimension state vector includes inertial velocity in inertial
18 | % coordinates and the inertial acceleration in inertial coordinates
19 | % [xdot; ydot; zdot; phidot; thetadot; psidot; udot; vdot; wdot;
20 | % pdot; qdot; rdot]
21 | %
22 | %Methodology: Use Newton's second law F=ma to calculate the acceleration
23 | % and velocity at each point in time for ode45 to integrate to find
24 | % position. Drag, gravity and motor thrust are only forces acting on drone.
25 | %
26 |
27 | g = 9.81;
28 |
29 | %Get IV's
30 | x1 = x(1);
31 | y1 = x(2);
32 | z1 = x(3);
33 | phi1 = x(4);
34 | theta1 = x(5);
35 | psi1 = x(6);
36 | u1 = x(7);
37 | v1 = x(8);
38 | w1 = x(9);
39 | p1 = x(10);
40 | q1 = x(11);
41 | r1 = x(12);
42 | Zc = -f1 - f2 - f3 - f4; %sum of 4 motor thrusts in -z direction
43 |
44 | %Find Pdot(xdot ydot zdot) (earth fixed)
45 | Pdot = R_eb(phi1,theta1,psi1,'rad')*[u1;v1;w1];
46 |
47 | %Find Odot(thetadot phidot psidot)
48 | Odot = T(phi1,theta1,psi1,'rad')*[p1;q1;r1];
49 |
50 | %Get magnitude of velocity (B)
51 | V_a = sqrt(u1^2 + v1^2 + w1^2);
52 |

```

```

53 %Calculate acceleration (body: Vb = [udot; vdot; wdot] components
54 %Vb = -w_b*V_b+f_b/m
55 Vb(1) = (r1*v1-q1*w1)+g*(-sin(theta1));
56 Vb(2) = (p1*w1-r1*u1)+g*(cos(theta1)*sin(phi1));
57 Vb(3) = (q1*u1-p1*v1)+g*(cos(theta1)*cos(phi1))+1/m*(Zc);
58
59 %Calculate L, M, N
60 m_a = -mu*sqrt(p1^2+q1^2+r1^2)*[p1; q1; r1];
61
62 %Calculate m_ctl
63 m_ctl(1) = (r/sqrt(2))*(-f1-f2+f3+f4);
64 m_ctl(2) = (r/sqrt(2))*(f1-f2-f3+f4);
65 m_ctl(3) = (r/sqrt(2))*(f1-f2+f3-f4);
66
67 %Calculate omega_dot(pdot,qdot,rdot)
68 omega_dot(1) = (Iy-Iz)/(Ix)*q1*r1 + 1/Ix*m_a(1) + 1/Ix*m_ctl(1);
69 omega_dot(2) = (Iz-Ix)/(Iy)*p1*r1 + 1/Iy*m_a(2) + 1/Iy*m_ctl(2);
70 omega_dot(3) = (Ix-Iy)/(Iz)*p1*q1 + 1/Iz*m_a(3) + 1/Iz*m_ctl(3);
71
72 %Put back into ode45
73 xstate(1) = Pdot(1); %xdot
74 xstate(2) = Pdot(2); %ydot
75 xstate(3) = Pdot(3); %zdot
76 xstate(4) = Odot(1); %phidot
77 xstate(5) = Odot(2); %thetadot
78 xstate(6) = Odot(3); %psidot
79 xstate(7) = Vb(1); %udot
80 xstate(8) = Vb(2); %vdot
81 xstate(9) = Vb(3); %wdot
82 xstate(10) = omega_dot(1); %pdot
83 xstate(11) = omega_dot(2); %qdot
84 xstate(12) = omega_dot(3); %rdot
85 xstate = xstate';
86 end

```

5. State Vector Disturbed

```

1 function [xstate] = objectEOM(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4)
2 %
3 %Inputs:
4 % t = time
5 % x = 12-dimension state vector includes the inertial velocity in
6 %   inertial coordinates and the inertial position in inertial coordinates
7 %   [x; y; z; phi; theta; psi; u; v; w; p; q; r]
8 % m = mass of drone (kg)
9 % r = radius of frame from motor to cg
10 % km = control moment coefficient
11 % Ix,Iy,Iz = moments about axis
12 % v = drag coefficient
13 % mu = moment coefficient
14 % f1,f2,f3,f4 = forces from motors
15 %
16 %Outputs:
17 % xdot = 12-dimension state vector includes inertial velocity in inertial
18 %   coordinates and the inertial acceleration in inertial coordinates
19 %   [xdot; ydot; zdot; phidot; thetadot; psidot; udot; vdot; wdot;
20 %   pdot; qdot; rdot]
21 %

```

```

22 %Methodology: Use Newton's second law F=ma to calculate the acceleration
23 % and velocity at each point in time for ode45 to integrate to find
24 % position. Drag, gravity and motor thrust are only forces acting on drone.
25 %
26
27 g = 9.81;
28
29 %Get IV's
30 x1 = x(1);
31 y1 = x(2);
32 z1 = x(3);
33 phi1 = x(4);
34 theta1 = x(5);
35 psi1 = x(6);
36 u1 = x(7);
37 v1 = x(8);
38 w1 = x(9);
39 p1 = x(10);
40 q1 = x(11);
41 r1 = x(12);
42
43 if (t > 3 && t < 3.5)
44     f4 = 0;
45 end
46
47 Zc = -f1 - f2 - f3 - f4; %sum of 4 motor thrusts in -z direction
48
49 %Find Pdot(xdot ydot zdot) (earth fixed)
50 Pdot = R_eb(phi1,theta1,psi1,'rad')*[u1;v1;w1];
51
52 %Find Odot(thetadot phidot psidot)
53 Odot = T(phi1,theta1,psi1,'rad')*[p1;q1;r1];
54
55 %Get magnitude of velocity (B)
56 V_a = sqrt(u1^2 + v1^2 + w1^2);
57
58 %Calculate acceleration (body: Vb = [udot; vdot; wdot] components
59 %Vb = -w_b*V_b+f_b/m
60 Vb(1) = (r1*v1-q1*w1)+g*(-sin(theta1))+1/m*(-v*V_a*u1);
61 Vb(2) = (p1*w1-r1*u1)+g*(cos(theta1)*sin(phi1))+1/m*(-v*V_a*v1);
62 Vb(3) = (q1*u1-p1*v1)+g*(cos(theta1)*cos(phi1))+1/m*(-v*V_a*w1)+1/m*(Zc);
63
64 %Calculate L, M, N
65 m_a = -mu*sqrt(p1^2+q1^2+r1^2)*[p1; q1; r1];
66
67 %Calculate m_ctl
68 m_ctl(1) = (r/sqrt(2))*(-f1-f2+f3+f4);
69 m_ctl(2) = (r/sqrt(2))*(f1-f2-f3+f4);
70 m_ctl(3) = (r/sqrt(2))*(f1-f2+f3-f4);
71
72 %Calculate omega_dot(pdot,qdot,rdot)
73 omega_dot(1) = (Iy-Iz)/(Ix)*q1*r1 + 1/Ix*m_a(1) + 1/Ix*m_ctl(1);
74 omega_dot(2) = (Iz-Ix)/(Iy)*p1*r1 + 1/Iy*m_a(2) + 1/Iy*m_ctl(2);
75 omega_dot(3) = (Ix-Iy)/(Iz)*p1*q1 + 1/Iz*m_a(3) + 1/Iz*m_ctl(3);
76
77 %Put back into ode45
78 xstate(1) = Pdot(1); %xdot
79 xstate(2) = Pdot(2); %ydot
80 xstate(3) = Pdot(3); %zdot

```

```

81 xstate(4) = Odot(1); %phidot
82 xstate(5) = Odot(2); %thetadot
83 xstate(6) = Odot(3); %psidot
84 xstate(7) = Vb(1); %udot
85 xstate(8) = Vb(2); %vdot
86 xstate(9) = Vb(3); %wdot
87 xstate(10) = omega_dot(1); %pdot
88 xstate(11) = omega_dot(2); %qdot
89 xstate(12) = omega_dot(3); %rdot
90 xstate = xstate';
91 end

```

6. Rotation Matrix

```

1 function [REB] = R_eb(phi,theta,psi,units)
2 %switch if input is either rad or deg
3 switch units
4     case 'deg'
5         REB(1,1) = cosd(theta)*cosd(psi);
6         REB(1,2) = cosd(theta)*sind(psi);
7         REB(1,3) = -sind(theta);
8         REB(2,1) = sind(phi)*sind(theta)*cosd(psi)-cosd(phi)*sind(psi);
9         REB(2,2) = sind(phi)*sind(theta)*sind(psi)+cosd(phi)*cosd(psi);
10        REB(2,3) = sind(phi)*cosd(theta);
11        REB(3,1) = cosd(phi)*sind(theta)*cosd(psi)+sind(phi)*sind(psi);
12        REB(3,2) = cosd(phi)*sind(theta)*sind(psi)-sind(phi)*cosd(psi);
13        REB(3,3) = cosd(phi)*cosd(theta);
14    case 'rad'
15        phi = phi * (180/pi);
16        theta = theta * (180/pi);
17        psi = psi * (180/pi);
18        REB(1,1) = cosd(theta)*cosd(psi);
19        REB(1,2) = cosd(theta)*sind(psi);
20        REB(1,3) = -sind(theta);
21        REB(2,1) = sind(phi)*sind(theta)*cosd(psi)-cosd(phi)*sind(psi);
22        REB(2,2) = sind(phi)*sind(theta)*sind(psi)+cosd(phi)*cosd(psi);
23        REB(2,3) = sind(phi)*cosd(theta);
24        REB(3,1) = cosd(phi)*sind(theta)*cosd(psi)+sind(phi)*sind(psi);
25        REB(3,2) = cosd(phi)*sind(theta)*sind(psi)-sind(phi)*cosd(psi);
26        REB(3,3) = cosd(phi)*cosd(theta);
27    end
28    REB = inv(REB);
29
30 end

```

7. Angular Rotation Matrix

```

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

```



```

11     Tmat(2,3) = -sind(phi);
12     Tmat(3,1) = 0;
13     Tmat(3,2) = sind(phi)*secd(theta);
14     Tmat(3,3) = cosd(phi)*secd(theta);
15
16     case 'rad'
17         phi = phi * (180/pi);
18         theta = theta * (180/pi);
19         psi = psi * (180/pi);
20         Tmat(1,1) = 1;
21         Tmat(1,2) = sind(phi)*tand(theta);
22         Tmat(1,3) = cosd(phi)*tand(theta);
23         Tmat(2,1) = 0;
24         Tmat(2,2) = cosd(phi);
25         Tmat(2,3) = -sind(phi);
26         Tmat(3,1) = 0;
27         Tmat(3,2) = sind(phi)*secd(theta);
28         Tmat(3,3) = cosd(phi)*secd(theta);
29     end

```