

```

1  #                               TITLE BLOCK
2  #*****
3  #Author:    Brandon White
4  #Date:      08/28/2019
5  #Desc:      This set of functions to solve the problems
6  #           given on HW1 of MAE 5010.
7  #*****
8  from rotations import *
9
10 def Euler3212EP(angles, radians = False, rounding = True):
11     #Expects degrees but can accept radians with flag set
12
13     import math
14
15     #convert to radians for math
16     if not radians:
17         for i in range(3):
18             angles[i] = angles[i] * (math.pi/180)
19
20     [psi, theta, phi] = angles
21
22     e = [math.cos(psi/2)*math.cos(theta/2)*math.cos(phi/2) +
23     •   math.sin(psi/2)*math.sin(theta/2)*math.sin(phi/2),
24         math.cos(psi/2)*math.cos(theta/2)*math.sin(phi/2) -
25     •   math.sin(psi/2)*math.sin(theta/2)*math.cos(phi/2),
26         math.cos(psi/2)*math.sin(theta/2)*math.cos(phi/2) +
27     •   math.sin(psi/2)*math.cos(theta/2)*math.sin(phi/2),
28         math.sin(psi/2)*math.cos(theta/2)*math.cos(phi/2) -
29     •   math.cos(psi/2)*math.sin(theta/2)*math.sin(phi/2)]
30
31     if rounding:
32         for i in range(4):
33             e[i] = round(e[i],5)
34
35     return e
36
37 def EP2Euler321(e, rounding = True):
38     import math
39
40     angles = [math.atan2(2 * (e[0]*e[1] + e[2]*e[3]), e[0]**2 +
41     •   e[3]**2 - e[1]**2 - e[2]**2),
42         math.asin(2 * (e[0]*e[2] - e[1]*e[3])),
43         math.atan2(2 * (e[0]*e[3] + e[2]*e[1]), e[0]**2 +
44     •   e[1]**2 - e[2]**2 - e[3]**2)]
45
46

```

```

33
40     #Convert to degrees
41     for i in range(3):
42         angles[i] = angles[i] * (180/math.pi)
43         if rounding:
44             angles[i] = round(angles[i],2)
45
46     return [angles[2], angles[1], angles[0]] #returns degrees
47
48 def make_gamma(I):
49     [Ix, Iz, Iy, Iz] = I
50     Gamma = [Ix*Iz - Ixz**2, 0, 0, 0, 0, 0, 0, 0, 0]
51     Gamma[1] = Ixz*(Ix-Iy-Iz)/Gamma[0]
52     Gamma[2] = (Iz*(Iz-Iy)+Ixz**2)/Gamma[0]
53     Gamma[3] = Iz/Gamma[0]
54     Gamma[4] = Ixz/Gamma[0]
55     Gamma[5] = (Iz - Ix)/Iy
56     Gamma[6] = Ixz/Iy
57     Gamma[7] = (Ix*(Ix-Iy)+Ixz**2)/Gamma[0]
58     Gamma[8] = Ix/Gamma[0]
59     return Gamma
60
61 def pos_kin(psi, theta, phi, u, v, w):
62     #d/dt([p_n, p_e, p_d])
63     from math import cos, sin
64     from numpy import matmul
65
66     A1 = [[cos(theta)*cos(psi), sin(phi)*sin(theta)*sin(psi) -
        • cos(phi)*cos(psi), cos(phi)*sin(theta)*cos(psi) +
        • sin(phi)*sin(psi)],
67           [cos(theta)*sin(psi), sin(phi)*sin(theta)*sin(psi) +
        • cos(phi)*cos(psi), cos(phi)*sin(theta)*sin(psi) -
        • sin(phi)*cos(psi)],
68           [-sin(theta), sin(phi)*cos(theta),
        • cos(phi)*cos(theta)]]
69     b1 = [u, v, w]
70     return matmul(A1, b1)
71
72 def pos_dyn(p, q, r, u, v, w, Fx, Fy, Fz, m):
73     #d/dt([u, v, w])
74     #>>> Need to move to the body frame?
75     x_dot = [r*v-q*w + Fx/m,
76              p*w-r*u + Fy/m,
77              q*u-p*v + Fz/m]
78     return x_dot

```

```

79
80 def rot_kin(e0, e1, e2, e3, r, p, q):
81     #d/dt( e )
82     from numpy import matmul
83     A3 = [[0, -p/2, -q/2, -r/2],
84            [p/2, 0, r/2, -q/2],
85            [q/2, -r/2, 0, p/2],
86            [r/2, q/2, -p/2, 0]]
87     b3 = [e0, e1, e2, e3]
88     return matmul(A3, b3)
89
90 def rot_dyn(Gamma, p, q, r, L, M, N, Iy):
91     #d/dt([p, q, r])
92     #>>> Need to move to the body frame?
93     x_dot = [Gamma[1]*p*q - Gamma[2]*q*r + Gamma[3]*L + Gamma[4]*N,
94              Gamma[5]*p*r - Gamma[6]*(p**2-r**2) + 1/Iy*M,
95              Gamma[7]*p*q - Gamma[1]*q*r + Gamma[4]*L + Gamma[8]*N]
96     return x_dot
97
98 def derivatives(state, t, MAV):
99     #state: [p_n, p_e, p_d, u, v, w, e0, e1, e2, e3, p, q, r]
100    #FM:     [Fx, Fy, Fz, ELL, M, N]
101    #MAV:    MAV.inert, MAV.m, MAV.gravity_needed
102
103    from math import sin, cos
104
105    #Unpack state, FM, MAV
106    [p_n, p_e, p_d, u, v, w, e0, e1, e2, e3, p, q, r] = state
107    storage = MAV.update_FM(t)
108    [Fx, Fy, Fz, L, M, N] = MAV.FM
109    [Ixz, Ix, Iy, Iz] = MAV.inert
110
111    #Get angle measures
112    angles = EP2Euler321([e0, e1, e2, e3])
113    [psi, theta, phi] = angles
114
115    #Get Xdot Terms
116    d_dt = [[], [], [], []]
117    d_dt[0] = pos_kin(psi, theta, phi, u, v, w)
118    d_dt[1] = pos_dyn(p, q, r, u, v, w, Fx, Fy, Fz, MAV.mass)
119    d_dt[2] = rot_kin(e0, e1, e2, e3, p, q, r)
120    d_dt[3] = rot_dyn(make_gamma(MAV.inert), p, q, r, L, M, N,
121    • MAV.inert[2])
122
123    #Build One Vector of Xdot

```

```

121         #added one vector of xdot
122
123     xdot = []
124     for eqn_set in d_dt:
125         for dot in eqn_set:
126             xdot.append(dot)
127
128     return xdot
129
130 def integrator(MAV, tf = 1, delta_t = 0.1, graphing = False):
131     from numpy import linspace
132     from scipy.integrate import odeint
133
134     #Make the time values
135     discrete_pts = (tf/delta_t) // 1 # force integer
136     t = linspace(0, tf, discrete_pts + 1)
137
138     #Integration Step
139     outputs = odeint(derivatives, MAV.state0, t, args = (MAV,))
140
141     #Optional 3D Path Graphing
142     if graphing:
143         from mpl_toolkits import mplot3d
144         import matplotlib.pyplot as plt
145         fig = plt.figure()
146         ax = plt.axes(projection="3d")
147         ax.plot3D(outputs[:,0], outputs[:,1], outputs[:,2],
148             • linestyle='-', marker='.')
149         ax.set_xlabel('P_n')
150         ax.set_ylabel('P_e')
151         ax.set_zlabel('P_z')
152         ax.invert_zaxis()
153         plt.show()
154
155     return [t, outputs]

```