```python
#                    TITLE BLOCK
#***************************************************
#Author:     Brandon White
#Date:       08/28/2019
#Desc:       This set of functions to solve the problems
#            given on HW1 of MAE 5010.
#***************************************************
from rotations import *

def Euler3212EP(angles, radians = False, rounding = True):
    #Expects degrees but can accept radians with flag set

    import math

    #convert to radians for math
    if not radians:
        for i in range(3):
            angles[i] = angles[i] * (math.pi/180)

    [psi, theta, phi] = angles

    e = [math.cos(psi/2)*math.cos(theta/2)*math.cos(phi/2) + math.sin(psi/2)*math.sin(theta/2)*math.sin(phi/2),
            math.cos(psi/2)*math.cos(theta/2)*math.sin(phi/2) - math.sin(psi/2)*math.sin(theta/2)*math.cos(phi/2),
            math.cos(psi/2)*math.sin(theta/2)*math.cos(phi/2) + math.sin(psi/2)*math.cos(theta/2)*math.sin(phi/2),
            math.sin(psi/2)*math.cos(theta/2)*math.cos(phi/2) - math.cos(psi/2)*math.sin(theta/2)*math.sin(phi/2)]

    if rounding:
        for i in range(4):
            e[i] = round(e[i],5)

    return e

def EP2Euler321(e, rounding = True):
    import math

    #outputs psi, theta, phi
    angles = [math.atan2(2 * (e[0]*e[3] + e[2]*e[1]), e[0]**2 + e[1]**2 - e[2]**2 - e[3]**2),
                math.asin(2 * (e[0]*e[2] - e[1]*e[3])),
                math.atan2(2 * (e[0]*e[1] + e[2]*e[3]), e[0]**2 + e[3]**2 - e[1]**2 - e[2]**2)]
```

```
                   e[3]**2 - e[1]**2 - e[2]**2)]

    #Convert to degrees
    for i in range(3):
        angles[i] = angles[i] * (180/math.pi)
        if rounding:
            angles[i] = round(angles[i],2)

    return [angles[2], angles[1], angles[0]] #returns degrees

def make_gamma(I):
    [Ixz, Ix, Iy, Iz] = I
    Gamma = [Ix*Iz - Ixz**2, 0, 0, 0, 0, 0, 0, 0, 0]
    Gamma[1] = Ixz*(Ix-Iy-Iz)/Gamma[0]
    Gamma[2] = (Iz*(Iz-Iy)+Ixz**2)/Gamma[0]
    Gamma[3] = Iz/Gamma[0]
    Gamma[4] = Ixz/Gamma[0]
    Gamma[5] = (Iz - Ix)/Iy
    Gamma[6] = Ixz/Iy
    Gamma[7] = (Ix*(Ix-Iy)+Ixz**2)/Gamma[0]
    Gamma[8] = Ix/Gamma[0]
    return Gamma

def pos_kin(psi, theta, phi, u, v, w):
    #d/dt([p_n, p_e, p_d])
    from math import cos, sin
    from numpy import matmul

    A1 = [[cos(theta)*cos(psi), sin(phi)*sin(theta)*cos(psi) -
    cos(phi)*sin(psi), cos(phi)*sin(theta)*cos(psi) +
    sin(phi)*sin(psi)],
                [cos(theta)*sin(psi), sin(phi)*sin(theta)*sin(psi) +
                cos(phi)*cos(psi), cos(phi)*sin(theta)*sin(psi) -
                sin(phi)*cos(psi)],
                [-sin(theta), sin(phi)*cos(theta),
                cos(phi)*cos(theta)]]
    b1 = [u, v, w]
    return matmul(A1, b1)

def pos_dyn(p, q, r, u, v, w, Fx, Fy, Fz, m):
    #d/dt([u, v, w])
    #>>> Need to move to the body frame?
    x_dot = [r*v-q*w + Fx/m,
            p*w-r*u + Fy/m,
            q*u-p*v + Fz/m]
```

```
    return x_dot

def rot_kin(e0, e1, e2, e3, r, p, q):
    #d/dt( e )
    from numpy import matmul
    A3 = [[0, -p/2, -q/2, -r/2],
            [p/2, 0, r/2, -q/2],
            [q/2, -r/2, 0, p/2],
            [r/2, q/2, -p/2, 0]]
    b3 = [e0, e1, e2, e3]
    return matmul(A3, b3)

def rot_dyn(Gamma, p, q, r, L, M, N, Iy):
    #d/dt([p, q, r])
    #>>> Need to move to the body frame?
    x_dot = [Gamma[1]*p*q - Gamma[2]*q*r + Gamma[3]*L + Gamma[4]*N,
            Gamma[5]*p*r - Gamma[6]*(p**2-r**2) + 1/Iy*M,
            Gamma[7]*p*q - Gamma[1]*q*r + Gamma[4]*L + Gamma[8]*N]
    return x_dot

def normalize_quaterions(e):
    import numpy
    from numpy.linalg import norm
    [e0, e1, e2, e3] = numpy.array(e)/norm(numpy.array(e))
    return [round(e0,3), round(e1,3), round(e2,3), round(e3,3)]


def derivatives(state, t, MAV):
    #state: [p_n, p_e, p_d, u, v, w, e0, e1, e2, e3, p, q, r]
    #FM:    [Fx, Fy, Fz, Ell, M, N]
    #MAV:   MAV.inert, MAV.m, MAV.gravity_needed

    state[6:10] = normalize_quaterions(state[6:10])

    MAV.state0 = state

    from math import sin, cos
    print()
    print('Time: ' + str(t))
    print('State: ' + str(state))
    #Unpack state, FM, MAV
    [p_n, p_e, p_d, u, v, w, e0, e1, e2, e3, p, q, r] = state
    storage = MAV.update_FM(t)
    [Fx, Fy, Fz, L, M, N] = MAV.FM
    [Ixz, Ix, Iy, Iz] = MAV.inert
```

```
    [Ixz, Ix, Iy, Iz] = MAV.inert

    #Get angle measures
    angles = EP2Euler321([e0, e1, e2, e3])
    [psi, theta, phi] = angles

    #Get Xdot Terms
    d_dt = [[], [], [], []]
    d_dt[0] = pos_kin(psi, theta, phi, u, v, w)
    d_dt[1] = pos_dyn(p, q, r, u, v, w, Fx, Fy, Fz, MAV.mass)
    d_dt[2] = rot_kin(e0, e1, e2, e3, p, q, r)
    d_dt[3] = rot_dyn(make_gamma(MAV.inert), p, q, r, L, M, N, MAV.inert[2])

    #Build One Vector of Xdot
    xdot = []
    for eqn_set in d_dt:
        for dot in eqn_set:
            xdot.append(dot)

    return xdot

def integrator(MAV, tf = 1, delta_t = 0.05, graphing = False):
    from numpy import linspace
    from scipy.integrate import odeint

    #Make the time values
    descrete_pts = (tf/delta_t) // 1  # force an integer
    t = linspace(0, tf, descrete_pts + 1)

    MAV.delta_t = delta_t
    #Integration Step
    outputs = odeint(derivatives, MAV.state0, t, args = (MAV,))

    #Optional 3D Path Graphing
    if graphing:
        from mpl_toolkits import mplot3d
        import matplotlib.pyplot as plt
        fig = plt.figure()
        ax = plt.axes(projection="3d")
        ax.plot3D(outputs[:,0], outputs[:,1], outputs[:,2], linestyle='-', marker='.')
        ax.set_xlabel('P_n')
        ax.set_ylabel('P_e')
        ax.set_zlabel('P_z')
```

```
        ax.invert_zaxis()
        plt.show()

    return [t, outputs]

```