```
#                   TITLE BLOCK
#************************************************
#Author:    Brandon White
#Date:      08/26/2019
#Desc:      Creates a MAV object with mass, moment
#          of inertia, and gravity properties
#************************************************

from rotations import *

#Calling the class with an aircraft name below creates an MAV object
class MAV:
    def __init__(self, aircraft = "None"):
       #All units listed in English units as denoted
       #NOTE: alpha and beta in radians
       self.name = aircraft
       self.last_update = 0
       self.delta_t = 10
       self.mass = 10  # Mass (slug)
       self.dynamic_density = False #True uses lapse rate for
       rho=f(h)

       #Inert = [Ixz, Ix, Iy, Iz]
       self.inert = [20, 10, 10, 10]  # Moment of Inertia (lbf*ft^2)

       #State = [p_n, p_e, p_d, u, v, w, e0, e1, e2, e3, p, q, r]
       self.state0 = [0, 0, -500, 50, 0, 0, 1, 0, 0, 0, 0, 0, 0]
            #Level flight at 500 ft at 50 ft/s

       #FM = [Fx, Fy, Fz, Ell, M, N]
       self.FM = [0, 0, 0, 0, 0, 0]
            #Equations for time-variant forces and moments
       self.FMeq = [0, 0, 0, 0, 0, 0]

       self.thrust_max = 10  #lbf

       #Geometric Properties and Coefficients
        #Order: b, c, x_cg, y_cg, z_cg, stall, coefficients
            #stall: True/False, M, alpha_0
            #coeff: CL0, CLa/b, CLq, CL_del_control, CD0, CDa/b,
            CDq, CD_del_control, [spare]
       self.wing = [0, 0, 0, 0, 0, [True, 50, 0.471],
                    [0, 0, 0, 0, 0, 0, 0, 0, [0,0,0]]]
       self.hstab = [0, 0, 0, 0, 0, [False, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, [0,0,0]]]
```

```
                    [0, 0, 0, 0, 0, 0, 0, 0, [0,0,0]]]
        self.vstab = [0, 0, 0, 0, 0, [False, 0, 0],
                      [0, 0, 0, 0, 0, 0, 0, 0, [0,0,0]]]

        #Controls Deflections: del_e, del_t(<1), del_a, del_r
        self.controls = [ 0, 1, 0, 0]

        #self.coeffeq = [0, 0, 0, 0, 0, 0, 0, 0,
         #                 0, 0, 0, 0, 0, 0, 0, 0,
         #                 0, 0, 0, 0, 0, 0, 0, 0]

        if aircraft != "None":
            try:
                method_to_call = getattr(self, aircraft.lower())
                method_to_call()
            except:
                print("No preconfig by given name: " +
                aircraft.lower())

    def density(self):
        if self.dynamic_density:
            #Define STD SL Terms
            ##P_0 = 101325 #Pa
            L = 0.0065 #K/m
            M = 0.0289644 #kg/mol
            R = 8.31447 #J/(mol K)
            g = 9.80665 #m/s^2
            T_0 = 288.15 #K
            p = P_0*(1 + (L*0.3048*self.state0[2])/T_0)**(g*M/(R*L))
            return p*M/(R*T) * (0.00194) #Covert to slug/ft^3
        else:
            return 0.002377 #SL slug/ft^3

    def update_state0(self, new_state):
        if len(new_state) != 13:
            print("Error - Not 13 items! \n You might need to
            convert angular\
                    values to quaternions...")
        else:
            self.state0 = new_state

    def CL_stall(self, alpha, model, coeff):
        from math import exp, sin, cos
        from numpy import sign
        [discard, M, alpha_star] = model
```

```
86          del_alpha = alpha - alpha_star
87          add_alpha = alpha + alpha_star
88          sig = (1+exp(-M*del_alpha)+exp(M*add_alpha))/((1+exp(-
            M*del_alpha))*(1+exp(M*add_alpha)))
89          CL = (1-sig)*(coeff[0]+coeff[1]*alpha) +
            sig*2*sign(alpha)*(sin(alpha)**2)*cos(alpha)
90          return CL
91
92      def CD(self,alpha, cd0, coeff, AR):
93          #Assume Oswald = 0.8
94          from math import pi
95          return cd0 + (coeff[0] + coeff[1]*alpha)**2/(pi*0.8*AR)
96
97      def aero_terms(self):
98          from math import atan, sin, cos
99          from numpy import matmul, transpose
100
101         [p_n, p_e, p_d, u, v, w, e0, e1, e2, e3, p, q, r] =
            self.state0
102         V_t = (u**2 + v**2 + w**2)**(1/2)  #NOTE: No wind included
103         Q = 0.5 * V_t**2 *self.density()
104
105         #print("Q: " + str(Q))
106
107         alpha = atan(w/u)  #NOTE: Negative sign since Pd is inverted
108         beta = atan(v/u)
109         angles = [alpha, beta]
110
111         w_coeff = self.wing[6]
112         h_coeff = self.hstab[6]
113         v_coeff = self.vstab[6]
114
115         #Rotated coefficients
116         Cx = -self.CD(alpha, w_coeff[4], w_coeff[0:2], self.wing[0]/
            self.wing[1])*cos(alpha) + self.CL_stall(alpha,
            self.wing[5], w_coeff[0:2])*sin(alpha)
117         Cxq = -w_coeff[6]*cos(alpha) + w_coeff[2]*sin(alpha)
118         Cxdele = -h_coeff[7]*cos(alpha) + h_coeff[3]*sin(alpha)
119         Cz = -self.CD(alpha, w_coeff[4], w_coeff[0:2], self.wing[0]/
            self.wing[1])*sin(alpha) - self.CL_stall(alpha,
            self.wing[5], w_coeff[0:2])*cos(alpha)
120         Czq = -w_coeff[6]*sin(alpha) - w_coeff[2]*cos(alpha)
121         Czdele = -h_coeff[7]*sin(alpha) - h_coeff[3]*cos(alpha)
122
123         #NEEDS REVISION
```

```
        #NEEDS REVISION
        #FORCES
        X = Q*self.wing[0]*self.wing[1]*(Cx + Cxq*self.wing[1]/
        (2*V_t)*q + Cxdele*self.controls[0])
        Y = Q*self.wing[0]*self.wing[1]*(v_coeff[3]*self.controls[3])
        Z = Q*self.wing[0]*self.wing[1]*(Cz + Czq*self.wing[1]/
        (2*V_t)*q + Czdele*self.controls[0])

        #MOMENTS
        L =
        Q*self.wing[0]**2*self.wing[1]*(w_coeff[8][0]*self.controls[2
        ] + v_coeff[8][0]*self.controls[3])
        M = Q*self.wing[0]*self.wing[1]**2*(w_coeff[8][3] +
        w_coeff[8][4]*alpha + w_coeff[8][5]*self.wing[1]/(2*V_t)*q +
        h_coeff[8][1]*self.controls[0])
        N =
        Q*self.wing[0]**2*self.wing[1]*(w_coeff[8][2]*self.controls[2
        ])

        #return [X, Y, Z, L, M, N]
        return [X, Y, Z, 0, 0, 0]

    def update_FM(self, t):
        from math import sin, cos
        from integrator import EP2Euler321

        #Angularize Gravity
        [psi, theta, phi] = EP2Euler321(self.state0[6:10])
        print('Angles:' + str([psi, theta, phi]))

        #All Forcing Functions
        for i in range(6):
            try:
                self.FM[i] = self.FMeq[i](t)
            except:
                self.FM[i] = self.FMeq[i]
        #print("FM w/Forcing:" + str(self.FM))

        #Add in Aero Terms
        aero_b = self.aero_terms()
        self.FM[0] += -32.2*self.mass*sin(theta) + aero_b[0] +
        self.thrust_max*self.controls[1]
        self.FM[1] += 32.2*self.mass*cos(theta)*sin(phi) + aero_b[1]
        self.FM[2] += 32.2*self.mass*cos(phi)*cos(phi) + aero_b[2]
        self.FM[3] += aero_b[3]
```

```
159         self.FM[4] += aero_b[4]
160         self.FM[5] += aero_b[5]
161         print("TOTAL FM" + str(self.FM))
162         return self.FM
163 
164     #================================================================
165     #Add templated aircraft below this line to pre-generate aircraft
166     #================================================================
167     def hw1_1(self):
168         import warnings
169         warnings.warn("This aircraft is depreciated",
    •         DeprecationWarning)
170         self.state0 = [100, 200, -500, 50, 0, 0,
171                        0.70643, 0.03084, 0.21263, 0.67438, 0, 0, 0]
172         self.FMeq = [0, 0, 0, 0, 0, 0]
173 
174     def hw1_2(self):
175         import warnings
176         warnings.warn("This aircraft is depreciated",
    •         DeprecationWarning)
177         from math import sin, cos
178         self.state0 = [100, 200, -500, 50, 0, 0,
179                        0.70643, 0.03084, 0.21263, 0.67438, 0, 0, 0]
180         self.FMeq = [(lambda t: sin(t)), 0, 0,
181                        0, 1e-4, 0]
182 
183     def hw2(self):
184         #All units listed in English units as denoted
185         #NOTE: alpha and beta in radians
186         self.mass = 0.925  # Mass (slug)
187         self.dynamic_density = False #True uses lapse rate for
    •         rho=f(h)
188 
189         #Inert = [Ixz, Ix, Iy, Iz]
190         self.inert = [2.857, 19.55, 26.934, 14.74]  # Moment of
    •         Inertia (lbf*ft^2)
191 
192         #State = [p_n, p_e, p_d, u, v, w, e0, e1, e2, e3, p, q, r]
193         self.state0 = [0, 0, 0, 20, 0, 0, 1, 0, 0, 0, 0, 0, 0]
194              #Level flight at 0ft at 20 ft/s
195 
196         #FM = [Fx, Fy, Fz, Ell, M, N]
197         self.FM = [0, 0, 0, 0, 0, 0]
198              #Equations for time-variant forces and moments
199         self.FMeq = [0, 0, 0, 0, 0, 0]
```

```
200
201         self.thrust_max = 5.62  #lbf
202
203         #Geometric Properties and Coefficients
204          #Order: b, c, x_cg, y_cg, z_cg, stall, coefficients
205              #stall: True/False, M, alpha_0
206              #coeff: CL0, CLa/b, CLq, CL_del_control, CD0, CDa/b,
  •              CDq, CD_del_control, [spare]
207         self.wing = [9.413, 0.6791, 0.9843, 0, 0, [True, 50, 0.471],
208                      [0.28, 3.5, 0, 0, 0.015, 0, 0, 0, [0.08, 0,
  •                      0.06, -0.02, -0.38, -3.6]]]
209         self.hstab = [2.297, 0.381, 0.8202, 0, 0, [False, 0, 0],
210                      [0.1, 5.79, 0, -0.36, 0.01, 0, 0, 0, [0, -0.5,
  •                      0]]]
211         self.vstab = [.9843, 0.381, 0.8202, 0, 0, [False, 0, 0],
212                      [0, 5.79, 0, -0.17, 0.01, 0, 0, 0, [0.105, 0,
  •                      0]]]
213
214         #Controls Deflections: del_e, del_t(<1), del_a, del_r
215         self.controls = [ 0, 1, 0, 0]
216
```