Brandon White

MAE 5010 – Autopilot Design and Test

Homework #1 (Due 08/29/2019)

===

2. Write a function that converts between quaternions and 3-2-1 Euler angles and one that converts back. These should look like EP = Euler3212EP([heading,pitch,roll]) EA = EP2Euler321([q0,q1,q2,q3]) Include an example of running your code to convert $[\psi, \theta, \varphi] = [150^\circ, 15^\circ, -30^\circ]$ to quaternions and ~q = [0.82205,0.26538,0.05601,0.50066] to Euler angles.

See Appendix for angle conversion code.

```
>>> import white_brandon_HW1 as HW1
>>> EP = HW1.Euler3212EP([150, 15, -30])
>>> EP
[0.21523, -0.1882, -0.21523, 0.93377]
>>> angles = HW1.EP2Euler321([0.82205, 0.26538, 0.05601, 0.50066])
>>> angles
[60.0, -10.0, 30.0]
```

Figure 1. Command Line I/O for Problem 2

3. Implement the kinematics and dynamics equations (using the quaternion formulation) in an integrator that takes in forces and moments. You should have a function that takes in state and returns xdot = derivatives(self, state, FM, MAV) where state = [pn,pe,pd, u,v,w, e0,e1,e2,e3, p,q,r] FM = [Fx,Fy,Fz, Ell,M,N] The function will contain computations of each state derivative, e.g.,

% position kinematics
  pn_dot = pe_dot = pd_dot =
% position dynamics
  u_dot = v_dot = w_dot =
% rotational kinematics
  e0_dot = e1_dot = 1 e2_dot = e3_dot =
% rotational dynamics
  p_dot = q_dot = r_dot =
% collect all the derivaties of the states
  xdot = [pn_dot; pe_dot; pd_dot; u_dot; v_dot; w_dot;... e0_dot; e1_dot; e2_dot; e3_dot; p_dot; q_dot; r_dot];

Include a gravitational force vector mg ˆfz. To make this into a simulation, you need to add initial conditions, vehicle parameters, forces and moments, and then integrate it. You'll add the first two in the step below.

See Appendix for derivative scripts.

```python
def update_FM(self, t):
    from math import sin, cos
    from white_brandon_HW1 import EP2Euler321

    #Angularize Gravity
    angles = EP2Euler321(self.state0[6:10])
    Fg = f2b(angles, [0, 0, 32.2*self.mass])

    #All Other Forcing Functions
    for i in range(6):
        try:
            self.FM[i] = self.FMeq[i](t)
        except:
            self.FM[i] = self.FMeq[i]

    #Add in Gravity
    self.FM[0] += Fg[0]
    self.FM[1] += Fg[1]
    self.FM[2] += Fg[2]

    return self.FM
```

Figure 2. Gravity Force Code

4. Select a candidate air vehicle you may use in your research, and approximate the mass and inertia of this vehicle. Define these in a separate file as terms like MAV.mass, MAV.Ix, MAV.Iy, MAV.Iz MAV.Ixz, and also define initial conditions for position, orientation, and rates. Move your definition of gravitational constant g here. Use consistent units.

```
class MAV:
    def __init__(self, aircraft = "None"):
        #All units listed in English units as denoted
        self.name = aircraft
        self.mass = 10  # Mass (lbf)
        #Inert = [Ixz, Ix, Iy, Iz]
        self.inert = [20, 10, 10, 10]  # Moment of Inertia (lbf*ft^2)
        self.gravity_needed = False
        #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]
            #Gravity ONLY in base model
        self.FMeq = [0, 0, (lambda t: 32.2*self.mass), 0, 0, 0]
```

Figure 3. MAV Class Object Code

The rate of descent was verified for the gravity only case through solving the ODE via WolframAlpha.com.

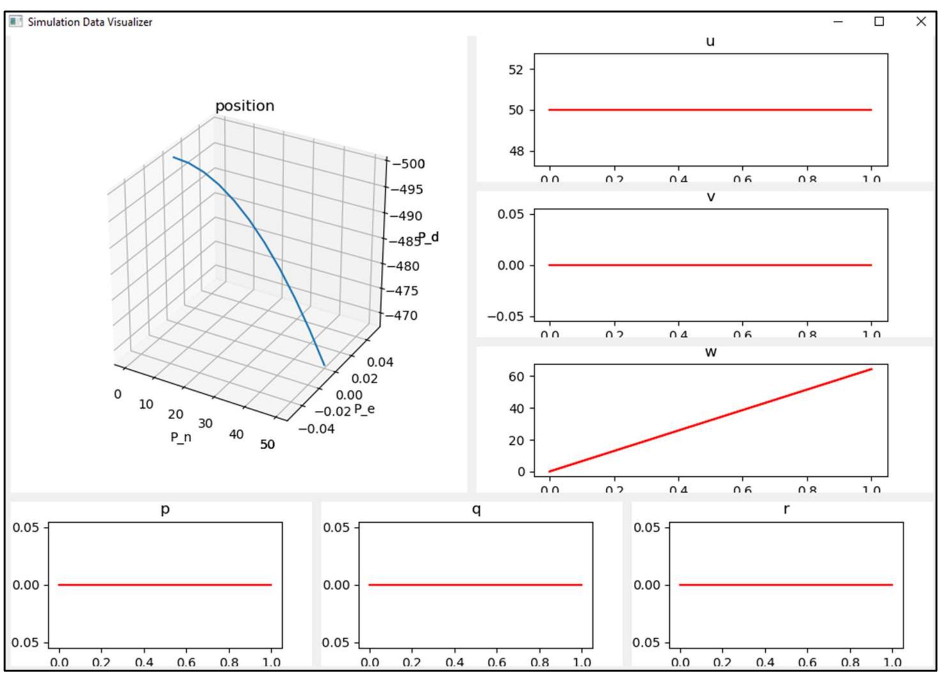

Figure 4. Gravity Only Simulation Results

5. Choose an integrator (you could try a Runge-Kutta integrator, or ode23), and connect it to the above dynamics function, having it read in the parameters to form a simple dynamics simulator. Simulate this vehicle from an initial condition of x, y, z = [100, 200, −500], ψ, θ, φ = [90, 15, 20]◦ (1) with F and M set to zero. Plot the positions, Euler angles, and rates. Does this make sense? Do you need to choose a different integrator? Simulate the system from the same initial conditions but use F = [sin(t),0,0] and M = [0,1e-4,0] Include plots of the output from these two cases and label your axes.

Selected integrator: scipy.integation.odeint()

The overall flight patterns make sense. As M is made non-zero, we see a resultant change in angular speed and quaternions not present in the purely translational motion of the first test. All axis are in terms of ft/s or rad/s as applicable.

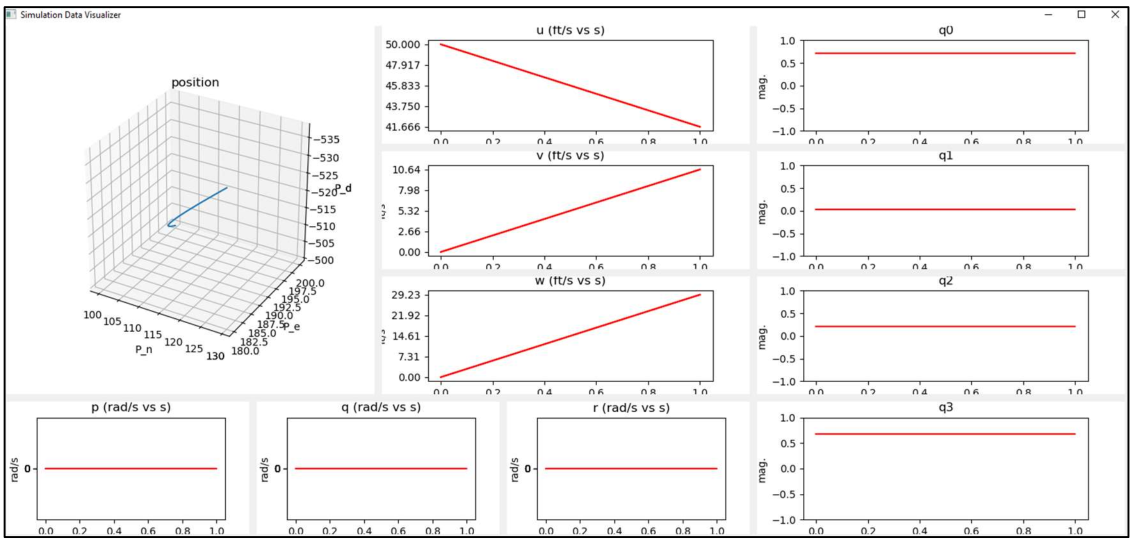

Figure 5. Outputs for Initial Condition with M = <0, 0, 0>

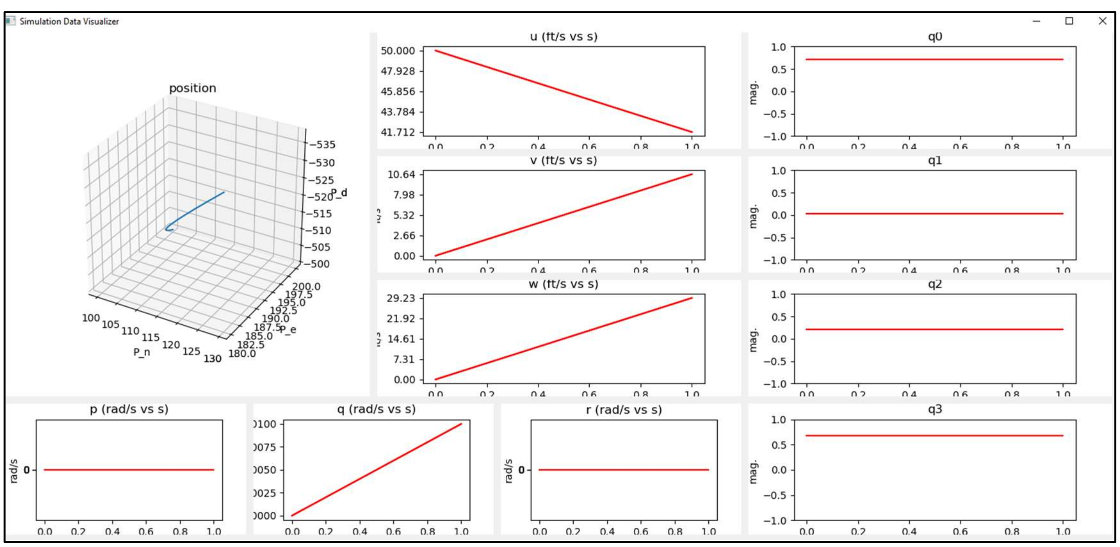

Figure 6. Outputs for Initial Condition with M = <0, 1e-4, 0>

6. Create a visualization scheme that shows the simulator states and rates, including a 3D visulation. You may use any interface or method you are happy with (displaying Euler angles is usually more intuitive that other parameterizations). Include a screen capture of its interface.

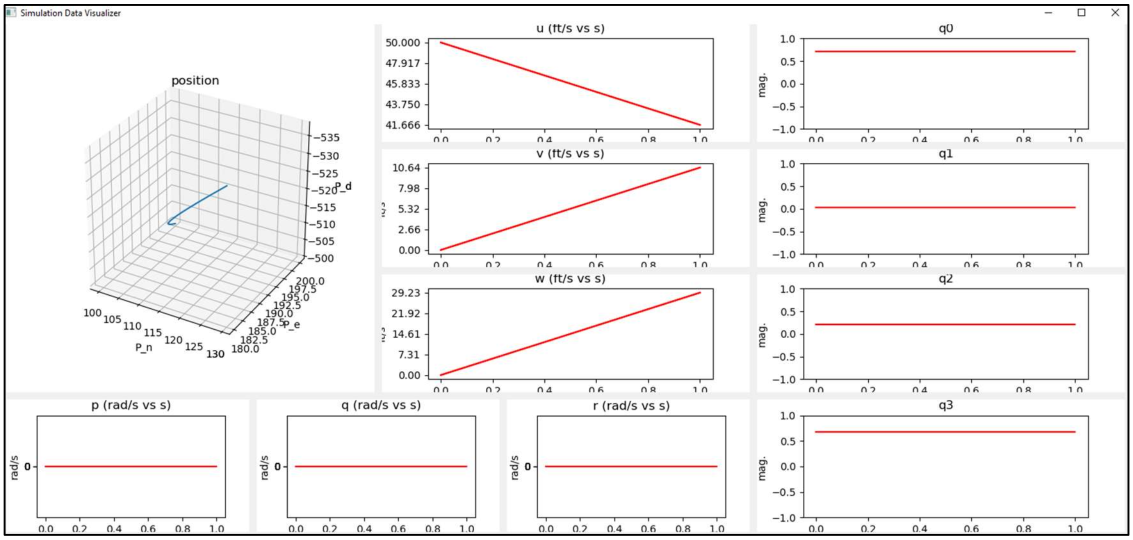

Figure 7. 3D and Parameter Visualization

Appendix A

Contents