```python
import numpy as np
import param as P


class planarVTOLDynamics:

    def __init__(self):

        # Initial state conditions
        self.state = np.matrix([[P.z0],         # z initial position
                                [P.h0],         # h initial position
                                [P.theta0],     # Theta initial orientation
                                [P.zdot0],      # zdot initial velocity
                                [P.hdot0],      # hdot initial velocity
                                [P.thetadot0]]) # Thetadot initial velocity

    def propagateDynamics(self,u):
        # P.Ts is the time step between function calls.
        # u contains the force and/or torque input(s).

        # RK4 integration
        k1 = self.Derivatives(self.state, u)
        k2 = self.Derivatives(self.state + P.Ts/2*k1, u)
        k3 = self.Derivatives(self.state + P.Ts/2*k2, u)
        k4 = self.Derivatives(self.state + P.Ts*k3, u)
        self.state += P.Ts/6 * (k1 + 2*k2 + 2*k3 + k4)


    # Return the derivatives of the continuous states
    def Derivatives(self,state,u):

        # States and forces
        z = state.item(0)
        h = state.item(1)
        theta = state.item(2)
        zdot = state.item(3)
        hdot = state.item(4)
        thetadot = state.item(5)
        fl = u[0]
        fr = u[1]

        zddot = -(fr+fl)*np.sin(theta)/(P.mc+2*P.mr)
        hddot = (-(P.mc+2*P.mr)*P.g + (fr+fl)*np.cos(theta))/(P.mc+2*P.mr)
        thetaddot = P.d*(fr-fl)/(P.Jc+2*P.mr*P.d**2)

        xdot = np.matrix([[zdot],[hdot],[thetadot],[zddot],[hddot],[thetaddot]])

        return xdot


    # Returns the observable states
    def Outputs(self):
        # Return them in a list and not a matrix
        return self.state[0:3].T.tolist()[0]

    # Returns all current states
    def States(self):
        # Return them in a list and not a matrix
        return self.state.T.tolist()[0]
```