E_blockbeam\python\ctrlPD.py

```python
import numpy as np
import blockBeamParam as P

class ctrlPD:
    def __init__(self):
        ####################################################
        #       PD Control: Time Design Strategy
        ####################################################
        # tuning parameters
        tr_z = 3  # rise time for outer loop - first part of problem
        zeta_z = 0.707  # damping ratio for outer loop
        zeta_th  = 0.707  # damping ratio for inner loop

        #---------------------------------------------------
        #                    Inner Loop
        #---------------------------------------------------
        ze = P.length/2.0  # equilibrium position - center of beam
        b0 = P.length/(P.m2*P.length**2/3.0+P.m1*ze**2)
        M = 10  # time scale separation between inner and outer loop
        tr_theta = tr_z/M  # rise time for inner loop
        wn_th = 2.2/tr_theta  # natural frequency for inner loop
        self.kp_th = wn_th**2/b0  # kp - inner
        self.kd_th = 2.0*zeta_th*wn_th/b0  # kd - inner

        # DC gain for inner loop
        DC_gain = 1.0

        #---------------------------------------------------
        #                    Outer Loop
        #---------------------------------------------------
        wn_z = 2.2/tr_z  # natural frequency - outer loop
        self.kp_z = -wn_z**2/P.g  # kp - outer
        self.kd_z = -2.0*zeta_z*wn_z/P.g  # kd - outer

        # print control gains to terminal
        print('DC_gain', DC_gain)
        print('kp_th: ', self.kp_th)
        print('kd_th: ', self.kd_th)
        print('kp_z: ', self.kp_z)
        print('kd_z: ', self.kd_z)

    def update(self, z_r, state):
        z = state[0][0]
        theta = state[1][0]
        zdot = state[2][0]
        thetadot = state[3][0]

        # the reference angle for theta comes from the outer loop PD control
        theta_r = self.kp_z * (z_r - z) - self.kd_z * zdot

        # the force applied to the cart comes from the inner loop PD control
        F_tilde = self.kp_th * (theta_r - theta) - self.kd_th * thetadot

        # feedback linearizing force
        F_fl = P.m1 * P.g * (z / P.length) + P.m2 * P.g / 2.0

        # total force
        F_unsat = F_tilde + F_fl

        # using the saturation block/function
        F = saturate(F_unsat, P.F_max)
        return F


def saturate(u, limit):
    if abs(u) > limit:
        u = limit * np.sign(u)
    return u
```