```
'''goals8step8c.py

  Goals 8 Step 8

  '''

# ------------------- SETUP THE KEYBOARD INTERFACE -------------------
# Import the necessary packages
import atexit, select, sys, termios

# Set up a handler, so the terminal returns to normal on exit.
stdattr = termios.tcgetattr(sys.stdin.fileno())
def reset_attr():
    termios.tcsetattr(sys.stdin.fileno(), termios.TCSAFLUSH, stdattr)
atexit.register(reset_attr)

# Switch terminal to canonical mode: do not wait for <return> press.
newattr    = termios.tcgetattr(sys.stdin.fileno())
newattr[3] = newattr[3] & ~termios.ICANON
termios.tcsetattr(sys.stdin.fileno(), termios.TCSAFLUSH, newattr)

# Define the kbhit() and getch() functions.
def kbhit():
    return sys.stdin in select.select([sys.stdin], [], [], 0)[0]
def getch():
    return sys.stdin.read(1)
#---------------------------------------------------------------------

# Import useful packages
import hebi
import numpy as np              # For future use
import matplotlib.pyplot as plt

from math import pi, sin, cos, asin, acos, atan2, sqrt
from time import sleep, time
import enum
def controller(shared):
    class Mode(enum.Enum):
        Hold=0
        Spline=1
        Scanning =2
        Reset = 3
        Setscan = 4

    class Object:
        def __init__(self, pan = 0.5, tilt = 0.5):
            self.pan = pan
            self.tilt = tilt
            self.conf = 0.2

    mode = Mode.Scanning

    #
    #  HEBI Initialization
    #
    #  Create the motor group, and pre-allocate the command and feedback
    #  data structures.  Remember to set the names list to match your
    #  motor.
    #
    names = ['9.8', '2.1']
    group = hebi.Lookup().get_group_from_names(['robotlab'], names)
    if group is None:
        print("Unable to find both motors " + str(names))
        raise Exception("Unable to connect to motors")

    command  = hebi.GroupCommand(group.size)
    feedback = hebi.GroupFeedback(group.size)

    #setup time
    t0 = 0.0
    Tmove = 8                  # 5 seconds
    tf = t0 + Tmove
    T=15

    #
    # Calibration
    #
    calib = np.array([[0], [-pi/6]])

    #Find initial position and velocities
    feedback = group.get_next_feedback(reuse_fbk=feedback)
    pinit = np.array([feedback.position[0], feedback.position[1]])
    p0 = np.array([[pinit[0]],[pinit[1]]])
    pf = p0
    phold = p0
    v0 = np.array([[0.0], [0.0]])
    vf = np.array([[0.0], [0.0]])

    #
    #  Pre-allocate the storage
    N = int(100 * T)            # 100 samples/second.
    Time = np.array([0.0] * N)
    PAct = np.array([[0.0] * N] * 2)
    PCmd = np.array([[0.0] * N] * 2)
    VAct = np.array([[0.0] * N] * 2)
    VCmd = np.array([[0.0] * N] * 2)
    objectpan = np.array([0.5] * N)
    objecttilt = np.array([0.5] * N)
    obj = []
    monitor = []
    (opan, otilt) = (0.5, 0.5)

    # Initialize the index and time.
    index = 0
    t      = 0.0

    #
    # Maximum Velocities/Accelerations
    #
    t_min = np.array([[2], [5]])
    v_max = 2/3*pi*pi/t_min
    a_max = v_max/0.25

    #
    # starred index
    #
    starred = 0
    #
    # calculate spline parameters
    #
    def splineparameters(Tmove, p0, v0, pf, vf):
        a = p0
        b = v0
        c = 3*(pf-p0)/(Tmove**2)-vf/Tmove-2*v0/Tmove
        d = -2*(pf-p0)/(Tmove**3)+vf/(Tmove**2)+v0/(Tmove**2)
        return (a, b, c, d)

    def splinetime(p0, pf, v0, vf):
        return max(np.amax(1.5*(np.absolute(p0-pf)/v_max+np.absolute(v0)/a_max+np.absolute(vf)/a_max)), 1)
    #
    #  Plot.
    #
    # Create a plot of position and velocity, actual and command!
    def plotPos():
        pan = [m.pan for m in monitor]
        tilt = [m.tilt for m in monitor]
        plt.scatter(pan, tilt, marker = "X")
        plt.xlim(-1.5,1.5)
        plt.ylim(-1.0, 1.0)
        plt.title('Goals 9 Step 6 - Monitor Multiple Detection')
        plt.ylabel('Tilt')
        plt.xlabel('Pan')
        for m in monitor:
            print("tilt ", m.tilt, "pan ", m.pan, "confidence ", m.conf)
        plt.show()

    def plot():
        fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
        ax1.plot(Time[0:index], PAct[0][0:index], color='blue', linestyle='-',  label='Pan PAct')
        ax1.plot(Time[0:index], PCmd[0][0:index], color='blue', linestyle='--', label='Pan PCmd')
        ax1.plot(Time[0:index], objectpan[0:index], color='red', linestyle='-.', label='Pan object')
        ax1.plot(Time[0:index], PAct[1][0:index], color='green', linestyle='-',  label='Tilt PAct')
        ax1.plot(Time[0:index], PCmd[1][0:index], color='green', linestyle='--', label='Tilt PCmd')
        ax1.plot(Time[0:index], objecttilt[0:index], color='black', linestyle='-.', label='Tilt object')

        ax2.plot(Time[0:index], VAct[0][0:index], color='blue', linestyle='-',  label='Pan VAct')
        ax2.plot(Time[0:index], VCmd[0][0:index], color='blue', linestyle='--', label='Pan VCmd')
        ax2.plot(Time[0:index], VAct[1][0:index], color='green', linestyle='-',  label='Tilt VAct')
        ax2.plot(Time[0:index], VCmd[1][0:index], color='green', linestyle='--', label='Til VCmd')

        ax1.set_title('Goals 9 Step 3 - Cammera Scale and Latency')
        ax1.set_ylabel('Position (rad)')
        ax2.set_ylabel('Velocity (rad/s)')
        ax2.set_xlabel('Time (s)')

        ax1.grid()
        ax2.grid()
        ax1.legend(loc = 'lower right')
        ax2.legend(loc = 'lower right')
        #plt.scatter(objectpan, objecttilt)
        plt.show()
    ##
    ## Execute Movement
    ##
    while True:
        # Read the actual data. This blocks (internally waits) 10ms for the data.
        feedback = group.get_next_feedback(reuse_fbk=feedback)
        pact = np.array([[feedback.position[0]], [feedback.position[1]]])
        vact = np.array([[feedback.velocity[0]], [feedback.velocity[1]]])
        # Compute the commands for this time step
```

```
            A = np.array([[1.0], [0.5]])
            Tperiod =np.array([[4], [8]])
            pcmd = A*np.sin(2*pi*(t-t0)/Tperiod)
            vcmd = A*(2*pi/Tperiod)*np.cos(2*pi*(t-t0)/Tperiod)
            if t >= tf and len(monitor) > 0:
                plotPos()
                starred = 0
                mode = Mode.Spline
                p0 = pcmd
                v0 = vcmd
                pf = np.array([[monitor[0].pan],[monitor[0].pan]])
                phold = pf
                vf = np.array([[0.0],[0.0]])
                Tmove = splinetime(p0, pf, v0, vf)
                (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
                t0 = t
                tf = t0 + Tmove
        elif mode is Mode.Setscan:
            pcmd = (a + b*(t-t0) + c*(t-t0)**2+d*(t-t0)**3)
            vcmd = (b + 2*c*(t-t0) + 3*d*(t-t0)**2)
            if t>=tf:
                phold= pcmd
                mode = Mode.Scanning
                t0 = t
                tf = t0 + 8
        elif mode is Mode.Hold:
            pcmd = phold
            vcmd = np.array([[0.0], [0.0]])
            if t >= tf:
                if starred >= len(monitor):
                    mode = Mode.Setscan
                    p0 = pcmd
                    v0 = vcmd
                    pf = np.array([[0],[0]])
                    phold = pf
                    vf = np.array([[0.0],[0.0]])
                    Tmove = splinetime(p0, pf, v0, vf)
                    (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
                    t0 = t
                    tf = t0 + Tmove
                else:
                    mode = Mode.Spline
                    p0 = pcmd
                    v0 = vcmd
                    pf = np.array([[monitor[starred].pan],[monitor[starred].pan]])
                    phold = pf
                    vf = np.array([[0.0],[0.0]])
                    Tmove = splinetime(p0, pf, v0, vf)
                    (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
                    t0 = t
                    tf = t0 + Tmove
        elif mode is Mode.Setscan:
                p0 = pcmd
                v0 = vcmd
                pf = np.array([[0.0],[0.0]]) + calib
                print("test: ", pf)
                vf = np.array([[1.0],[0.25]])
                Tmove = splinetime(p0, pf, v0, vf)
                (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
                t0 = t
                tf = t0 + Tmove
                mode = Mode.Reset
        elif mode is Mode.Reset:
            pcmd = (a + b*(t-t0) + c*(t-t0)**2+d*(t-t0)**3)
            #print("pcmd", pcmd)
            vcmd = (b + 2*c*(t-t0) + 3*d*(t-t0)**2)
            #print("reset", pf, pcmd)
            if t>=tf:
                mode = Mode.Hold
                t0 = t
        elif mode is Mode.Spline:
            pcmd = (a + b*(t-t0) + c*(t-t0)**2+d*(t-t0)**3)
            vcmd = (b + 2*c*(t-t0) + 3*d*(t-t0)**2)
            if t>=tf:
                starred = starred + 1
                phold= pcmd
                mode = Mode.Hold
                t0 = t
                tf = t0 + 2
        else:
            print("this should never happen")
        # Send the commands.  This returns immediately.
        command.position = pcmd
        command.velocity = vcmd
        group.send_command(command)

        # Store the data for this time step (at the current index).
        if(index< N):
            Time[index] = t
            PAct[0][index], PAct[1][index] = pact[0], pact[1]
            VAct[0][index], VAct[1][index] = vact[0], vact[1]
            VCmd[0][index], VCmd[1][index] = vcmd[0], vcmd[1]
            objectpan[index] = opan
            objecttilt[index] = otilt
        elif index == N:
            print("plot")
            # plot()
        # check if keyboard hit
        if kbhit():
            # Take action, based on the character.
            ch = getch()
            mode = Mode.Spline
            if   (ch == 'a'):
                # make robot move without wait
                p0 = pcmd
                v0 = vcmd
                pf = np.array([[-0.15337068238377316], [-0.037060368086649906]]) + calib
                vf = np.array([[0.0], [0.0]])
                Tmove = splinetime(p0, pf, v0, vf)
                (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
                print('A: pan: 60, tilt: 0')
            elif (ch == 'b'):
                p0 = pcmd
                v0 = vcmd
                pf = np.array([[-pi/8],[pi/6]]) + calib
                vf = np.array([[0.0], [0.0]])
                Tmove = splinetime(p0, pf, v0, vf)
                (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
                print('B: pan: -45, tilt: -30')
            elif (ch == 'c'):
                p0 = pcmd
                v0 = vcmd
                pf = np.array([[0],[pi/8]]) + calib
                vf = np.array([[0.0], [0.0]])
                Tmove = splinetime(p0, pf, v0, vf)
                (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
                print('C: pan: 60, tilt: 45')
            elif (ch == 'd'):
                p0 = pcmd
                v0 = vcmd
                pf = np.array([[0],[pi/6]]) + calib
                vf = np.array([[0.0], [0.0]])
                Tmove = splinetime(p0, pf, v0, vf)
                (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
                print('D: pan: 0, tilt: -30')
            elif (ch == 's'):
                p0 = pcmd
                v0 = vcmd
                pf = np.array([[0],[0]])
                vf = np.array([[0.0], [0.0]])
                Tmove = splinetime(p0, pf, v0, vf)
                (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
                mode = Mode.Setscan
                obj = []
            elif (ch == 'q'):
                # If we want to quit, break out of the loop.
                print('Quitting')
                break
            else:
                # Report the bad press.
                print("'Unknown character '%c'" % ch)
            t0 = np.array(t)
            tf = np.array(t0 + Tmove)
        if shared.lock.acquire(timeout=0.001):
            if shared.newdata and mode is Mode.Scanning:
                # Compute the object angles.
                #print("p", shared.deltapan, "t", shared.deltatilt)
                # Lpan = 0.41709845
                Lpan = 0.2
                Ltilt = 0.8
                R = 0.5
                minpan = pact[0][0] - vact[0][0] * Lpan - shared.scalepan * 320
                maxpan = pact[0][0] - vact[0][0] * Lpan + shared.scalepan * 320
                mintilt = pact[1][0] + vact[1][0] * Ltilt - shared.scaletilt * 240
                maxtilt = pact[1][0] + vact[1][0] * Ltilt + shared.scaletilt * 240
                #Ltilt = 0.8100304
                #Lpan = 0.
                #Ltilt = 0.1
                #opan = pact[0][0] - shared.deltapan
                print("shared", shared.balls)
                print("monitored", monitor)
                obj = []
                for ball in shared.balls:
                    #print(ball.deltapan, ball.deltatilt)
                    opan = pact[0][0] - vact[0][0] * Lpan - ball.deltapan #+ vact[0][0] * Lpan
                    otilt = pact[1][0] + vact[1][0] * Ltilt + ball.deltatilt

                    obj.append(Object(opan, otilt))
                def closest(mp, mt):
                    minR = 100
                    flag = False
                    for o in reversed(obj):
                        if(sqrt((o.pan-mp)**2+(o.tilt-mt)**2)<minR):
                            flag = True
                            minR = sqrt((o.pan-mp)**2+(o.tilt-mt)**2)
                            minO = o
                    if flag:
                        obj.remove(minO)
                    return minO, minR, flag
                for m in reversed(monitor):
                    cObj, cDist, exist = closest(m.pan, m.tilt)
                    if exist:
                        if cDist < R:
                            m = cObj
                            m.conf = min(m.conf+0.2, 1)
                        elif m.pan < minpan or m.pan > minpan or m.tilt < mintilt or m.tilt > maxtilt:
                            m.conf = m.conf-0.005
                        if m.conf < -0.5:
                            monitor.remove(m)
                    else:
                        m.conf = m.conf-0.1
                print(len(obj))
                for o in reversed(obj):
                    monitor.append(o)
                """Lpan = 2.5//1.2
       Ltilt = 1.71//1.47
       opan = pact[0][0] – shared.deltapan
       otilt = pact[1][0] + shared.deltatilt
       opan = pact[0][0] + vact[0][0] * Lpan – shared.deltapan
       otilt = pact[1][0] + vact[1][0] * Ltilt + shared.deltatilt"""
                """mode = Mode.Spline
       p0 = pact
       pf = np.array([[opan], [otilt]])
       v0 = vact
       vf = np.array([[0], [0]])
       Tmove = splinetime(p0, pf, v0, vf)
       (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
       t0 = t
       tf = t0 + Tmove
       print("a=", a, "b=", b, "c =",c, "d=",d)
       print(’going to object’)"""
                if(index<N):
                    objectpan[index] = opan
                    objecttilt[index] = otilt
                    # Clear the data.
                shared.newdata = False
            shared.lock.release()
        # Check whether we have a new object location.
        # Advance the index/time.
        index += 1
        t     += 0.01
```