```python
'''goals3democode.py

  Demo code for Goals 3
'''

# 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, inf
from time import sleep, time
from keycheck import kbhit, getch

def controller(shared):
    #
    #  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 = ['5.5', '1.2']
    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)

    dt = 0.01                       # HEBI feedback comes in at 100Hz!

    # Also read the initial position.
    feedback = group.get_next_feedback(reuse_fbk=feedback)
    pinit = feedback.position[0]


    #
    #  Define the parameters
    #
    T = 15.0                         # 5 seconds total run time


    #
    #  Pre-allocate the storage.
    #
    N = int(T / dt)                 # 100 samples/second.

    Time = [0.0] * N
    PAct = np.zeros((2, N))
    PCmd = np.zeros((2, N))
    PErr = np.zeros((2, N))
    VAct = np.zeros((2, N))
    VCmd = np.zeros((2, N))
    VErr = np.zeros((2, N))
    ObjAngles = np.zeros((2,N))

    # Helper functions
    # Helper Functions
    def movetime(po, pf, v0, vf):
        tms = abs(3*(pf - p0)/(2*v0))
        tms = max(tms)
        tms += np.max(np.abs(v0/amax))
        tms += np.max(np.abs(vf/amax))
        return tms


    def calcparams(t0, tf, p0, pf, v0, vf):
        T_move = tf - t0
        a = p0
        b = v0
        c = 3*(pf - p0)/(T_move)**2 - vf/T_move - 2*v0/T_move
        d = (-2)*(pf - p0)/(T_move)**3 + vf/(T_move)**2 + v0/(T_move)**2
        return (a, b, c, d)


    def splinecmds(t, t_spline, spline_params):
        (a, b, c, d) = spline_params
        pcmd = a + b*(t - t_spline) + c*(t - t_spline)**2 + d*(t - t_spline)**3
        vcmd = b + 2*c*(t - t_spline) + 3*d*(t - t_spline)**2
        return (pcmd, vcmd)

    def scancmds(t, t_0, Apan, Atilt, Tscan):
        pcmd = np.array([Apan*sin((2*pi)*(t - t_0)/Tscan), Atilt*sin((8*pi)*(t - t_0)/Tscan)]) + offset
        vcmd = np.array([2*pi*Apan/Tscan*cos((2*pi)*(t - t_0)/ Tscan), 8*pi*Atilt/Tscan*cos((8*pi)*(t - t_0)/ Tscan)])
        return (pcmd, vcmd)


    #
    #  Execute the movement.
    #
    # Initialize the index and time.
    index = 0
    t = 0.0
    offset = np.array([-pi/24, -pi/8])
    feedback = group.get_next_feedback(reuse_fbk=feedback)
    phold = np.array([feedback.position[0], feedback.position[1]])
    vmax = np.array([2.5, 2.5])
    amax = np.array([vmax/0.4, vmax/0.4])
```

```
    from enum import Enum

    class Traj(Enum):
        HOLD = 0 # Keeps a constant pos, zero velocity forever
        SPLINE = 1 # Computes a cubic spline, ends at tf
        SCAN = 2 # Computes sinusoidal pos/vel, never ends

    class Mode(Enum):
        GOHOME = 0 # Go to the home position (0,0)
        TRACKING = 1 # Track the primary object of interest
        SCANNING = 2 # Scan the entire field of view (w/o tracking)

    traj = Traj.HOLD
    mode = Mode.GOHOME

    p0 = phold
    pf = p0
    v0 = np.array([0.0, 0.0])
    vf = np.array([0.0, 0.0])
    Apan = 3*pi/8
    Atilt = pi/6
    Tscan = 4*pi**2/(3*vmax[1])

    t0 = 0.0
    tm = inf #inf before
    tf = t0 + tm

    (a,b,c,d) = calcparams(t0, tf, p0, pf, v0, vf)
    segment_num = 1

    key_positions = {'s': np.array([0.0, 0.0]),
    'a': np.array([pi/12, 0.0]), \
    'b': np.array([-pi/3, 0.0]), \
    'c': np.array([pi/3, pi/4]), \
    'd': np.array([0.0, -pi/12]), \
    'e': np.array([-pi/4, pi/6]), \
    'z': np.array([0.0, 0.0]), \
    't': phold}

    historyofobjects = []
    knownobjects = []
    Rmatch = 0.3  # in radians
    objofinterest = 0

    while True:
        if traj is Traj.SPLINE:
            (pcmd, vcmd) = splinecmds(t, t0, (a, b, c, d))
        elif traj is Traj.SCAN:
            (pcmd, vcmd) = scancmds(t, t0, Apan, Atilt, Tscan)
        elif traj is Traj.HOLD:
            (pcmd, vcmd) = (phold, np.array([0.0, 0.0]))
        else:
            raise ValueError(f'Bad Trajectory Type {traj}')
        # Compute the commands for this time step.
        # Check for key presses.
        if kbhit():
            # Grab and report the key
            key_pressed = getch()
            print("Saw key '%c'" % key_pressed)
            # Do something

            if key_pressed in key_positions:
                t0 = t
                v0 = vcmd
                p0 = pcmd
                pf = key_positions[key_pressed] + offset
                vf = np.array([0.0, 0.0])
                if key_pressed == 's':
                    traj = Traj.SPLINE
                    mode = Mode.SCANNING
                    historyofobjects = []
                    vf = np.array([2*pi*Apan/Tscan, 8*pi*Atilt/Tscan])
                elif key_pressed == 'z':
                    traj = Traj.SPLINE
                    mode = Mode.GOHOME
                    phold = pf
                elif key_pressed == 't':
                    pf = pcmd
                    traj = Traj.SPLINE
                    mode = Mode.TRACKING
                    phold = pf

                tm = movetime(p0, pf, vmax, vf)

                tm = max(tm, 1)
                tf = t0 + tm
                (a, b, c, d) = calcparams(t0, tf, p0, pf, v0, vf)

            elif key_pressed == 'q':
                break

        if traj is Traj.SPLINE and t + dt > tf:
            t0 = t
            p0 = pcmd
            v0 = vcmd
            tm = inf
            if mode is Mode.SCANNING:
                # In SCANNING mode: Transition to the SCAN trajectory.
                traj = Traj.SCAN
                # tm = Tscan
            elif mode is Mode.GOHOME:
```

```python
                # IN GOHOME mode: Transition to the HOLD trajectory.
                traj = Traj.HOLD
                pf = phold
            elif mode is Mode.TRACKING:
                traj = Traj.HOLD
                phold = pcmd
            else:
                raise ValueError('Unexpected end of motion')
            tf = t0 + tm

        # if traj is Traj.SCAN and t+dt > tf:
            #break



        obj_newdat = False
        if shared.lock.acquire():
            obj_newdat = shared.newdata
            num_objs_detected = len(shared.detectedobjs)
            historyofobjects = historyofobjects + shared.detectedobjs

            for obj in shared.detectedobjs:
                if len(knownobjects) == 0:
                    knownobjects.append(obj)
                for i in range(len(knownobjects)):
                    dist = np.sqrt((obj[0] - knownobjects[i][0])**2 + (obj[1] - knownobjects[i][1])**2)
                    if (dist > Rmatch):
                        knownobjects.append(obj)


            if mode is Mode.TRACKING and obj_newdat and num_objs_detected > 0:
                # cant do this in next if statement because don't have access to shared.params
                pf = np.array([shared.detectedobjs[0][0], shared.detectedobjs[0][1]]) - offset
                shared.newdata = False
            shared.lock.release()

        if mode is Mode.TRACKING and obj_newdat and num_objs_detected > 0:

            traj = Traj.SPLINE
            t0 = t
            p0 = pcmd
            pf = np.array([knownobjects[objofinterest][0], objofinterest[0][1]]) - offset
            vf = np.array([0.0, 0.0])
            tm = movetime(p0, pf, vmax, vf)
            tm = max(tm, 1)
            tf = t0 + tm
            (a, b, c, d) = calcparams(t0, tf, p0, pf, v0, vf)

        # Send the commands.  This returns immediately.
        command.position = list(pcmd)
        command.velocity = list(vcmd)
        group.send_command(command)

        # Read the actual data. This blocks (internally waits) 10ms for
        # the data and therefor replaces the "sleep(0.01)".
        feedback = group.get_next_feedback(reuse_fbk=feedback)
        pact = np.array([feedback.position[0], feedback.position[1]]) + offset
        vact = np.array([feedback.velocity[0], feedback.velocity[1]])

        if shared.lock.acquire():
            shared.motorpan = pact[0]
            shared.motortilt = pact[1]
            shared.lock.release()


        # Advance the index/time.
        if index < N:
            # Store the data for this time step (at the current index).
            Time[index] = t
            PAct[:,index] = pact
            PCmd[:,index] = pcmd
            PErr[:,index] = pact - pcmd
            VAct[:,index] = vact
            VCmd[:,index] = vcmd
            VErr[:,index] = vact - vcmd
            if shared.lock.acquire():
                val1 = None
                val2 = None
                if len(shared.detectedobjs) > 0:
                    val1 = shared.detectedobjs[0][0]
                    val2 = shared.detectedobjs[0][1]
                ObjAngles[0,index] = val1
                ObjAngles[1,index] = val2
                shared.lock.release()

            index += 1

            # Do not end loop but stop storing values
        t  += dt


    #
    #  Plot.
    #
    # Create a plot of position and velocity, actual and command!
    fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, sharex=True)

    fig, (ax5) = plt.subplots(1, 1, sharex=True)
    ax5.scatter(ObjAngles[0,0:index], ObjAngles[1,0:index], color='black')
    ax5.set_xlim(-1.7, 1.7)
    ax5.set_ylim(-1, 1)
```

```python
    ax5.set_xlabel('theta pan')
    ax5.set_ylabel('theta tilt')
    ax5.set_title('Objects locations from scan')
    plt.show()

    ax1.plot(Time[0:index], PAct[0, 0:index], color='blue', linestyle='-',  label='Pan P Act')
    ax1.plot(Time[0:index], PCmd[0, 0:index], color='blue', linestyle='--', label='Pan P Cmd')
    ax2.plot(Time[0:index], VAct[0, 0:index], color='blue', linestyle='-',  label='Pan V Act')
    ax2.plot(Time[0:index], VCmd[0, 0:index], color='blue', linestyle='--', label='Pan V Cmd')
    ax3.plot(Time[0:index], PAct[1, 0:index], color='green', linestyle='-',  label='Tilt P Act')
    ax3.plot(Time[0:index], PCmd[1, 0:index], color='green', linestyle='--', label='Tilt P Cmd')
    ax4.plot(Time[0:index], VAct[1, 0:index], color='green', linestyle='-',  label='Tilt V Act')
    ax4.plot(Time[0:index], VCmd[1, 0:index], color='green', linestyle='--', label='Tilt V Cmd')
    # ax2.plot(Time[0:index], VErr[0:index], color='red', linestyle='-.', label='Err')
    ax1.plot(Time[0:index], ObjAngles[0, 0:index], color='red', linestyle='--', label='Object Pan Angles')
    ax3.plot(Time[0:index], ObjAngles[1, 0:index], color='red', linestyle='--', label='Object Tilt Angles')

    ax1.set_title(f'Robot Data - Step 4')
    ax1.set_ylabel('Pan Position (rad)')
    ax2.set_ylabel('Pan Velocity (rad/s)')
    ax3.set_ylabel('Tilt Position (rad/s)')
    ax4.set_ylabel('Tilt Velocity (rad/s)')
    ax4.set_xlabel('Time (s)')

    ax1.grid()
    ax2.grid()
    ax1.legend()
    ax2.legend()
    ax3.grid()
    ax4.grid()
    ax3.legend()
    ax4.legend()
    ax5.grid()

    plt.show()

if __name__ == '__main__':
    controller(None)
```