```
'''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


#
#   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))

# Helper functions
# Helper Functions
def movetime(po, pf, vmax):
    tms = abs(3*(pf - p0)/(2*vmax))
    return max(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)
```

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

p0 = pinit
pf = p0
v0 = np.array([0.0, 0.0])
vf = np.array([0.0, 0.0])
t0 = 0.0
tm = inf
tf = t0 + tm
# tf = t0 + movetime(p0, pf, vmax)
offset = np.array([-2.25*pi/8, -pi/8])

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

key_positions = {'a': np.array([pi/3, 0.0]), \
'b': np.array([-pi/3, 0.0]), \
'c': np.array([pi/3, pi/4]), \
'd': np.array([0.0, -pi/6]), \
'e': np.array([-pi/4, pi/6]), \
'z': np.array([0.0, 0.0])}

while True:
    (pcmd, vcmd) = splinecmds(t, t0, (a, b, c, d))
    # 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:
            # Update to new spline
            t0 = t
            p0 = pcmd
            v0 = vcmd
            pf = key_positions[key_pressed] + offset
            tm = movetime(p0, pf, vmax)

            vf = np.array([0.0, 0.0])
            tm += np.max(np.abs(v0/amax))
            tm = max(tm, 0.1)
            tf = t0 + tm
            (a, b, c, d) = calcparams(t0, tf, p0, pf, v0, vf)
        elif key_pressed == 'q':
            break

    if t + dt > tf:
        # HOLD
        t0 = t
        p0 = pcmd
        # v0 = vcmd
        v0 = np.array([0.0, 0.0])
        pf = p0
        tm = inf
        vf = np.array([0.0, 0.0])
        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]])
    vact = np.array([feedback.velocity[0], feedback.velocity[1]])

```

```python
    # 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
        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)

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.set_title(f'Robot Data - Step 6')
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()

plt.show()
```