```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
from time import sleep, time


#
#  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']
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 = 7.0                         # 5 seconds total run time


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

Time = [0.0] * N
PAct = [0.0] * N
PCmd = [0.0] * N
PErr = [0.0] * N
VAct = [0.0] * N
VCmd = [0.0] * N
VErr = [0.0] * N


#
#  Execute the movement.
#
# Initialize the index and time.
index = 0
t = 0.0
feedback = group.get_next_feedback(reuse_fbk=feedback)
p_init = feedback.position[0]
print(p_init)
p_boundaries = [p_init, pi/2, -pi/2, p_init]
v_max = 2.9
num_splines = 3
T_moves = [0.0] * num_splines
for i in range(num_splines):
    T_moves[i] = abs(3*(p_boundaries[i+1] - p_boundaries[i])/(2*v_max))
    print(f'T_moves: {T_moves}')

v_init = [0.0] * num_splines
v_final = [0.0] * num_splines

first_move = 1.0

t_boundaries = [first_move]*(num_splines + 1)

for i in range(num_splines):
    t_boundaries[i+1] = t_boundaries[i] + T_moves[i]
print(f'T boundaries: {t_boundaries}')

a = [0.0] * num_splines
b = [0.0] * num_splines
c = [0.0] * num_splines
d = [0.0] * num_splines
for i in range(num_splines):
    a[i] = p_boundaries[i]
    b[i] = v_init[i]
    c[i] = 3*(p_boundaries[i+1] - p_boundaries[i])/(T_moves[i])**2 - v_final[i]/T_moves[i] - 2*v_init[i]/T_moves[i]
    d[i] = (-2)*(p_boundaries[i+1] - p_boundaries[i])/(T_moves[i])**3 + v_final[i]/(T_moves[i])**2 + v_init[i]/(T_moves[i])**2

while True:
    # Compute the commands for this time step.
    if t < t_boundaries[0]:
        pcmd = p_boundaries[0]
        vcmd = v_init[0]
    elif t >= t_boundaries[-1]:
        if t > first_move + t_boundaries[-1]:
```

```
                break
            else:
                pcmd = p_boundaries[num_splines]
                vcmd = v_final[num_splines - 1]
        else:
            i = 0
            if t < t_boundaries[1]:
                i = 0
            elif t < t_boundaries[2]:
                i = 1
            elif t < t_boundaries[3]:
                i = 2
            vcmd = b[i] + 2*c[i]*(t - t_boundaries[i]) + 3*d[i]*(t - t_boundaries[i])**2
            pcmd = a[i] + b[i]*(t - t_boundaries[i]) + c[i]*(t - t_boundaries[i])**2 + d[i]*(t - t_boundaries[i])**3



        # Send the commands.  This returns immediately.
        command.position = [pcmd]
        command.velocity = [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 = feedback.position[0]
        vact = feedback.velocity[0]

        # 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

        # Advance the index/time.
        index += 1
        t     += dt

        # Break (end) the loop, if we run out of storage or time.
        if index >= N or t >= T:
            break


#
#  Plot.
#
# Create a plot of position and velocity, actual and command!
fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True)

ax1.plot(Time[0:index], PAct[0:index], color='blue', linestyle='-',  label='Act')
ax1.plot(Time[0:index], PCmd[0:index], color='blue', linestyle='--', label='Cmd')
ax1.plot(Time[0:index], PErr[0:index], color='red', linestyle='-.', label='Err')
ax2.plot(Time[0:index], VAct[0:index], color='blue', linestyle='-',  label='Act')
ax2.plot(Time[0:index], VCmd[0:index], color='blue', linestyle='--', label='Cmd')
ax2.plot(Time[0:index], VErr[0:index], color='red', linestyle='-.', label='Err')

ax1.set_title(f'Robot Data - Step 8')
ax1.set_ylabel('Position (rad)')
ax2.set_ylabel('Velocity (rad/s)')
ax2.set_xlabel('Time (s)')

ax1.grid()
ax2.grid()
ax1.legend()
ax2.legend()

plt.show()
```