```
'''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.Hold

    #
    #  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 = 0.0                     # 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
```

```
92      N = int(100 * T)            # 100 samples/second.
93      Time = np.array([0.0] * N)
94      PAct = np.array([[0.0] * N] * 2)
95      PCmd = np.array([[0.0] * N] * 2)
96      VAct = np.array([[0.0] * N] * 2)
97      VCmd = np.array([[0.0] * N] * 2)
98      objectpan = np.array([0.5] * N)
99      objecttilt = np.array([0.5] * N)
100     obj = []
101     monitor = []
102     (opan, otilt) = (0.5, 0.5)
103
104     # Initialize the index and time.
105     index = 0
106     t     = 0.0
107
108     #
109     # Maximum Velocities/Accelerations
110     #
111     t_min = np.array([[2], [5]])
112     v_max = 2/3*pi*pi/t_min
113     a_max = v_max/0.25
114
115
116     #
117     # calculate spline parameters
118     #
119     def splineparameters(Tmove, p0, v0, pf, vf):
120         a = p0
121         b = v0
122         c = 3*(pf-p0)/(Tmove**2)-vf/Tmove-2*v0/Tmove
123         d = -2*(pf-p0)/(Tmove**3)+vf/(Tmove**2)+v0/(Tmove**2)
124         return (a, b, c, d)
125
126     def splinetime(p0, pf, v0, vf):
127         return max(np.amax(1.5*(np.absolute(p0-pf)/v_max+np.absolute(v0)/a_max+np.absolute(vf)/a_max)), 1)
128     #
129     #  Plot.
130     #
131     # Create a plot of position and velocity, actual and command!
132     def plotPos():
133         pan = [m.pan for m in monitor]
134         tilt = [m.tilt for m in monitor]
135         plt.scatter(pan, tilt, marker = "X")
136         plt.xlim(-1.5,1.5)
137         plt.ylim(-1.0, 1.0)
138         plt.title('Goals 9 Step 6 – Monitor Multiple Detection')
139         plt.ylabel('Tilt')
140         plt.xlabel('Pan')
141         for m in monitor:
142             print("tilt ", m.tilt, "pan ", m.pan, "confidence ", m.conf)
143         plt.show()
144
145     def plot():
146         fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
147         ax1.plot(Time[0:index], PAct[0][0:index], color='blue', linestyle='-',  label='Pan PAct')
148         ax1.plot(Time[0:index], PCmd[0][0:index], color='blue', linestyle='--', label='Pan PCmd')
149         ax1.plot(Time[0:index], objectpan[0:index], color='red', linestyle='-.', label='Pan object')
150         ax1.plot(Time[0:index], PAct[1][0:index], color='green', linestyle='-',  label='Tilt PAct')
151         ax1.plot(Time[0:index], PCmd[1][0:index], color='green', linestyle='--', label='Tilt PCmd')
152         ax1.plot(Time[0:index], objecttilt[0:index], color='black', linestyle='-.', label='Tilt object')
153
154         ax2.plot(Time[0:index], VAct[0][0:index], color='blue', linestyle='-',  label='Pan VAct')
155         ax2.plot(Time[0:index], VCmd[0][0:index], color='blue', linestyle='--', label='Pan VCmd')
156         ax2.plot(Time[0:index], VAct[1][0:index], color='green', linestyle='-',  label='Tilt VAct')
157         ax2.plot(Time[0:index], VCmd[1][0:index], color='green', linestyle='--', label='Til VCmd')
158
159         ax1.set_title('Goals 9 Step 3 – Cammera Scale and Latency')
160         ax1.set_ylabel('Position (rad)')
161         ax2.set_ylabel('Velocity (rad/s)')
162         ax2.set_xlabel('Time (s)')
163
164         ax1.grid()
165         ax2.grid()
166         ax1.legend(loc = 'lower right')
167         ax2.legend(loc = 'lower right')
168         #plt.scatter(objectpan, objecttilt)
169         plt.show()
170     ##
171     ## Execute Movement
172     ##
173     while True:
174         # Read the actual data. This blocks (internally waits) 10ms for the data.
175         feedback = group.get_next_feedback(reuse_fbk=feedback)
176         pact = np.array([[feedback.position[0]], [feedback.position[1]]])
177         vact = np.array([[feedback.velocity[0]], [feedback.velocity[1]]])
178         # Compute the commands for this time step
179         if mode is Mode.Scanning:
180             A = np.array([[1.0], [0.5]])
181             Tperiod =np.array([[4], [8]])
```

```
183                 vcmd = A*(2*pi/Tperiod)*np.cos(2*pi*(t-t0)/Tperiod)
184                 if t >= tf:
185                     mode = Mode.Hold
186                     plotPos()
187                     """p0 = pcmd
188             v0 = vcmd
189             pf = np.array([[0.0],[0.0]])
190             phold = pf
191             vf = np.array([[1.0],[0.25]])
192             Tmove = splinetime(p0, pf, v0, vf)
193             (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
194             print("param", "p0", p0, "v0", v0, "Tmove", Tmove)
195             print("coefficients", "a", a, "b", b, "c", c, "d", d)
196             t0 = t
197             tf = t0 + Tmove
198             mode = Mode.Reset"""
199               elif mode is Mode.Setscan:
200                     pcmd = (a + b*(t-t0) + c*(t-t0)**2+d*(t-t0)**3)
201                     vcmd = (b + 2*c*(t-t0) + 3*d*(t-t0)**2)
202                     if t>=tf:
203                         phold= pcmd
204                         mode = Mode.Scanning
205                         t0 = t
206                         tf = t0 + 8
207               elif mode is Mode.Hold:
208                     pcmd = phold
209                     vcmd = np.array([[0.0], [0.0]])
210                     """if t >= tf:
211             p0 = pcmd
212             v0 = vcmd
213             pf = np.array([[0.0],[0.0]]) + calib
214             print("test: ", pf)
215             vf = np.array([[1.0],[0.25]])
216             Tmove = splinetime(p0, pf, v0, vf)
217             (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
218             t0 = t
219             tf = t0 + Tmove
220             mode = Mode.Reset"""
221               elif mode is Mode.Reset:
222                     pcmd = (a + b*(t-t0) + c*(t-t0)**2+d*(t-t0)**3)
223                     #print("pcmd", pcmd)
224                     vcmd = (b + 2*c*(t-t0) + 3*d*(t-t0)**2)
225                     #print("reset", pf, pcmd)
226                     if t>=tf:
227                         mode = Mode.Hold
228                         t0 = t
229               elif mode is Mode.Spline:
230                     pcmd = (a + b*(t-t0) + c*(t-t0)**2+d*(t-t0)**3)
231                     vcmd = (b + 2*c*(t-t0) + 3*d*(t-t0)**2)
232                     if t>=tf:
233                         phold= pcmd
234                         mode = Mode.Hold
235                         t0 = t
236                         tf = t0 + 1
237               else:
238                     print("this should never happen")
239               # Send the commands.  This returns immediately.
240               command.position = pcmd
241               command.velocity = vcmd
242               group.send_command(command)
243
244               # Store the data for this time step (at the current index).
245               if(index< N):
246                     Time[index] = t
247                     PAct[0][index], PAct[1][index] = pact[0], pact[1]
248                     PCmd[0][index], PCmd[1][index] = pcmd[0], pcmd[1]
249                     VAct[0][index], VAct[1][index] = vact[0], vact[1]
250                     VCmd[0][index], VCmd[1][index] = vcmd[0], vcmd[1]
251                     objectpan[index] = opan
252                     objecttilt[index] = otilt
253               elif index == N:
254                     print("plot")
255                     plot()
256               # check if keyboard hit
257               if kbhit():
258                     # Take action, based on the character.
259                     ch = getch()
260                     mode = Mode.Spline
261                     if   (ch == 'a'):
262                         # make robot move without wait
263                         p0 = pcmd
264                         v0 = vcmd
265                         pf = np.array([[0], [0]]) + calib
266                         vf = np.array([[0.0], [0.0]])
267                         Tmove = splinetime(p0, pf, v0, vf)
268                         (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
269                         print('A: pan: 60, tilt: 0')
270                     elif (ch == 'b'):
271                         p0 = pcmd
272                         v0 = vcmd
```

```
                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:
                # Compute the object angles.
                #print("p", shared.deltapan, "t", shared.deltatilt)
                # Lpan = 0.41709845
                Lpan = 0.24
                Ltilt = 0.1
                R = 0.0
                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
                    otilt = pact[1][0] + vact[1][0] * Ltilt + ball.deltatilt
                    obj.append(Object(opan, otilt))
                def closest(mp, mt):
                    minR = 100
                    minO = Object(0, 0)
                    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 < -1.0:
                            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
```

```
365             opan = pact[0][0] – shared.deltapan
366             otilt = pact[1][0] + shared.deltatilt
367             opan = pact[0][0] + vact[0][0] * Lpan – shared.deltapan
368             otilt = pact[1][0] + vact[1][0] * Ltilt + shared.deltatilt"""
369                     """mode = Mode.Spline
370             p0 = pact
371             pf = np.array([[opan], [otilt]])
372             v0 = vact
373             vf = np.array([[0], [0]])
374             Tmove = splinetime(p0, pf, v0, vf)
375             (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
376             t0 = t
377             tf = t0 + Tmove
378             print("a=", a, "b=", b, "c =",c, "d=",d)
379             print('going to object')"""
380                     if(index<N):
381                         objectpan[index] = opan
382                         objecttilt[index] = otilt
383                         # Clear the data.
384                     shared.newdata = False
385                 shared.lock.release()
386               # Check whether we have a new object location.
387               # Advance the index/time.
388               index += 1
389               t     += 0.01
```