```
goals9step7c.py
Nov 30, 23 16:20
                                                                                                            Page 1/5
   ''' goals8step8c.py
    Goals 8 Step 8
5
                  ----- SETUP THE KEYBOARD INTERFACE -----
   # Import the necessary packages
8
   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)
16
   # Switch terminal to canonical mode: do not wait for <return> press.
17
  newattr = termios.tcgetattr(sys.stdin.fileno())
newattr[3] = newattr[3] & ~termios.ICANON
18
19
  termios.tcsetattr(sys.stdin.fileno(), termios.TCSAFLUSH, newattr)
20
22
   # Define the kbhit() and getch() functions.
   def kbhit():
23
       return sys.stdin in select.select([sys.stdin], [], [], 0)[0]
25
   def getch():
       return sys.stdin.read(1)
27
28
   # Import useful packages
  import hebi
30
   import numpy as np
                                       # For future use
31
   import matplotlib.pyplot as plt
33
   from math import pi, sin, cos, asin, acos, atan2, sqrt
34
   from time import sleep, time
35
36
   import enum
37
   def controller(shared):
       class Mode(enum.Enum):
38
            Hold=0
            Spline=1
            Scanning =2
            Reset = 3
42
            Setscan = 4
43
44
       class Object:
45
            def __init__(self, pan = 0.5, tilt = 0.5):
46
                self.pan = pan
self.tilt = tilt
47
48
                self.conf = 0.2
49
51
       mode = Mode.Scanning
52
        # HEBI Initialization
55
        # Create the motor group, and pre-allocate the command and feedback
56
          data structures. Remember to set the names list to match your
57
58
         motor.
59
       names = ['9.8', '2.1']
60
        group = hebi.Lookup().get_group_from_names(['robotlab'], names)
61
       if group is None:
    print("Unable to find both motors" + str(names))
62
63
64
            raise Exception ("Unable to connect to motors")
        command = hebi.GroupCommand(group.size)
        feedback = hebi.GroupFeedback(group.size)
67
68
        #setup time
69
70
        t0 = 0.0
        Tmove = 8
                                     # 5 seconds
71
        tf = t0 + Tmove
72
        T = 15
73
74
75
76
        # Calibration
77
        calib = np.array([[0], [-pi/6]])
        #Find initial position and velocities
        feedback = group.get_next_feedback(reuse_fbk=feedback)
81
       pinit = np.array([feedback.position[0], feedback.position[1]])
82
       p0 = np.array([[pinit[0]],[pinit[1]]])
pf = p0
83
84
        phold = p0
85
        v0 = np.array([[0.0], [0.0]])
86
        vf = np.array([[0.0], [0.0]])
87
88
```

89

Pro-allocate the storage

```
goals9step7c.py
Nov 30, 23 16:20
                                                                                                                                                              Page 2/5
           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)
94
95
           PCHIC = HP.AIIAY([[0.0] * N] * 2)
VAct = np.array([[0.0] * N] * 2)
VCmd = np.array([[0.0] * N] * 2)
objectpan = np.array([0.5] * N)
96
97
           objectpan = np.array([0.5] * N)
objecttilt = np.array([0.5] * N)
98
99
100
           obj = []
           monitor = []
101
            (opan, otilt) = (0.5, 0.5)
102
103
            # Initialize the index and time.
           index = 0
105
                    = 0.0
106
107
108
            # Maximum Velocities/Accelerations
109
110
           t_min = np.array([[2], [5]])
v_max = 2/3*pi*pi/t_min
111
112
           a_max = v_max/0.25
113
114
115
116
            # starred index
117
            starred = 0
118
119
120
            # calculate spline parameters
121
           def splineparameters(Tmove, p0, v0, pf, vf):
122
                  a = p0
                  b = v0
124
                  c = 3*(pf-p0)/(Tmove**2)-vf/Tmove-2*v0/Tmove
125
                  d = -2*(pf-p0) / (Tmove**3) + vf / (Tmove**2) + v0 / (Tmove**2)
126
127
                  return (a, b, c, d)
128
           def splinetime(p0, pf, v0, vf):
129
130
                  return max(np.amax(1.5*(np.absolute(p0-pf)/v_max+np.absolute(v0)/a_max+np.absolute(vf)/a_max)), 1)
131
132
                Plot.
133
            # Create a plot of position and velocity, actual and command!
134
           def plotPos():
135
                  pan = [m.pan for m in monitor]
136
                  tilt = [m.tilt for m in monitor]
137
                  plt.scatter(pan, tilt, marker = "X")
138
                  plt.xlim(-1.5,1.5)
plt.ylim(-1.0, 1.0)
139
140
                  plt.title('Goals 9 Step 6 - Monitor Multiple Detection')
plt.ylabel('Tilt')
141
142
                  plt.xlabel('Pan')
143
144
                  for m in monitor:
                        print("tilt", m.tilt, "pan", m.pan, "confidence", m.conf)
146
                  plt.show()
147
148
           def plot():
                  fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
149
                 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')
150
151
152
153
154
155
156
                 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='Tilt VCmd')
157
158
159
160
161
                  ax1.set_title('Goals 9 Step 3 - Cammera Scale and Latency')
162
                  ax1.set_ylabel('Position(rad)')
ax2.set_ylabel('Velocity(rad/s)')
163
164
                  ax2.set_xlabel('Time(s)')
165
166
167
                  ax1.grid()
168
                  ax2.grid()
                  ax1.legend(loc = 'lower right')
169
                  ax2.legend(loc = 'lower right')
170
171
                  #plt.scatter(objectpan, objecttilt)
172
                  plt.show()
173
            ## Execute Movement
174
175
176
            while True:
                  # Read the actual data. This blocks (internally waits) 10ms for the data.
177
178
                  feedback = group.get_next_feedback(reuse_fbk=feedback)
179
                  pact = np.array([[feedback.position[0]], [feedback.position[1]]])
                  vact = np.array([[feedback.velocity[0]], [feedback.velocity[1]]])
180
```

101

```
goals9step7c.py
Nov 30, 23 16:20
                                                                                                           Page 3/5
                 A = np.array([[1.0], [0.5]])
                 Tperiod =np.array([[4], [8]])
                 pcmd = A*np.sin(2*pi*(t-t0)/Tperiod)
185
                 vcmd = A*(2*pi/Tperiod)*np.cos(2*pi*(t-t0)/Tperiod)
186
                 if t >= tf and len(monitor) > 0:
187
                     plotPos()
188
                     starred = 0
189
                     mode = Mode.Spline
190
191
                     p0 = pcmd
                     v0 = vcmd
192
                     pf = np.array([[monitor[0].pan],[monitor[0].pan]])
193
194
                     phold = pf
                     vf = np.array([[0.0], [0.0]])
195
196
                     Tmove = splinetime(p0, pf, v0, vf)
197
                     (a, b, c, d) = splineparameters (Tmove, p0, v0, pf, vf)
198
                     t0 = t
                     tf = t0 + Tmove
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>=t.f:
203
                     phold= pcmd
204
                     mode = Mode.Scanning
205
                     t0 = t
tf = t0 + 8
206
207
            elif mode is Mode.Hold:
208
                 pcmd = phold
209
210
                 vcmd = np.array([[0.0], [0.0]])
211
                 if t >= tf:
                     if starred >= len(monitor):
212
                         mode = Mode.Setscan
213
                         p0 = pcmd
214
                         v0 = vcmd
215
                         pf = np.array([[0],[0]])
216
                         phold = pf
217
                         vf = np.array([[0.0], [0.0]])
218
                         Tmove = splinetime(p0, pf, v0, vf)
219
220
                          (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
221
                         t0 = t
222
                         tf = t0 + Tmove
223
                         mode = Mode.Spline
224
                         p0 = pcmd
225
                         v0 = vcmd
226
                         pf = np.array([[monitor[starred].pan],[monitor[starred].pan]])
227
                         phold = pf
228
                         vf = np.array([[0.0], [0.0]])
229
                         Tmove = splinetime(p0, pf, v0, vf)
230
231
                          (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
                         t0 = t
232
233
                         tf = t0 + Tmove
234
            elif mode is Mode.Setscan:
                     p0 = pcmd

v0 = vcmd
235
236
                     pf = np.array([[0.0], [0.0]]) + calib
237
                     print("test: ", pf)
238
                     vf = np.array([[1.0], [0.25]])
239
                     Tmove = splinetime(p0, pf, v0, vf)
240
                     (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
t0 = t
241
242
                     tf = t0 + Tmove
243
                     mode = Mode.Reset
244
245
            elif mode is Mode.Reset:
                246
247
248
                 #print("reset", pf, pcmd)
249
250
                 if t>=tf:
                     mode = Mode.Hold
251
252
                     t0 = t
            elif mode is Mode.Spline:
253
                pcmd = (a + b*(t-t0) + c*(t-t0)**2+d*(t-t0)**3)
254
                 vcmd = (b + 2*c*(t-t0) + 3*d*(t-t0)**2)
255
                 if t>=t.f:
256
                     starred = starred + 1
257
258
                     phold= pcmd
259
                     mode = Mode.Hold
                     t0 = t
260
261
                     tf = t0 + 2
262
263
                print ("this should never happen")
264
            # Send the commands. This returns immediately.
            command.position = pcmd
command.velocity = vcmd
265
266
            group.send_command(command)
267
268
            # Store the data for this time step (at the current index).
269
270
            if(index< N):</pre>
271
                 Time[index] = t
                                  PAct[1][indox] = pact[0] pact[1]
```

DAG+[0][indox]

```
goals9step7c.py
Nov 30, 23 16:20
                                                                                                                                         Page 4/5
                     VAct[0][index], VAct[1][index] = vact[0], vact[1]
                     VCmd[0][index], VCmd[1][index] = vcmd[0], vcmd[1]
275
                     objectpan[index] = opan
276
277
                     objecttilt[index] = otilt
               elif index == N:
278
                     print ("plot")
279
                     # plot()
280
                # check if keyboard hit
281
               if kbhit():
282
283
                     # Take action, based on the character.
                     ch = getch()
284
285
                     mode = Mode.Spline
                           (ch == 'a'):
286
287
                           # make robot move without wait
                          p0 = pcmd
288
                           v0 = vcmd
289
                          pf = np.array([[-0.15337068238377316], [-0.037060368086649906]]) + calib
290
                           vf = np.array([[0.0], [0.0]])
Tmove = splinetime(p0, pf, v0, vf)
291
                     (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)

print('A: pan: 60, tilt: 0')

elif (ch == 'b'):
292
293
294
295
                          p0 = pcmd
296
                           v0 = vcmd
297
                          pf = np.array([[-pi/8],[pi/6]]) + calib
298
                           vf = np.array([[0.0], [0.0]])
299
                           Tmove = splinetime(p0, pf, v0, vf)
300
                     (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
print('B: pan: -45, tilt: -30')
elif (ch == 'c'):
301
302
303
                          p0 = pcmd
304
                           v0 = vcmd
305
                          pf = np.array([[0],[pi/8]]) + calib
vf = np.array([[0.0], [0.0]])
Tmove = splinetime(p0, pf, v0, vf)
306
307
308
                     (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
print('C: pan: 60, tilt: 45')
elif (ch == 'd'):
309
310
311
312
                          p0 = pcmd
313
                           v0 = vcmd
                          pf = np.array([[0],[pi/6]]) + calib
314
                           vf = np.array([[0.0], [0.0]])
Tmove = splinetime(p0, pf, v0, vf)
315
316
                           (a, b, c, d) = splineparameters (Tmove, p0, v0, pf, vf)
317
                     print ('D: pan: 0, tilt: -30')
elif (ch == 's'):
318
319
                          p0 = pcmd
320
                           v0 = vcmd
321
                          pf = np.array([[0],[0]])
322
                           vf = np.array([[0.0], [0.0]])
323
324
                           Tmove = splinetime(p0, pf, v0, vf)
                     (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
mode = Mode.Setscan
obj = []
elif (ch == 'q'):
325
326
328
                           # If we want to quit, break out of the loop.
329
                          print ('Quitting')
330
331
                          break
                     else:
332
                           # Report the bad press.
333
                          print ("'Unknown character '%c'" % ch)
334
                     t0 = np.array(t)
335
                     tf = np.array(t0 + Tmove)
336
337
               if shared.lock.acquire(timeout=0.001):
                     if shared.newdata and mode is Mode.Scanning:
338
                           # Compute the object angles.
339
                           #print("p", shared.deltapan, "t", shared.deltatilt)
340
                           # Lpan = 0.41709845
341
                           Lpan = 0.2
342
                           Ltilt = 0.8
343
                           R = 0.5
344
                          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
345
346
347
348
                           #Ltilt = 0.8100304
#Lpan = 0.
349
350
                           \#Ltilt = 0.1
351
352
                           #opan = pact[0][0] - shared.deltapan
                          print("shared", shared.balls)
353
                          print("monitored", monitor)
354
                           obj = []
355
                           for ball in shared.balls:
356
                                #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
357
358
359
360
                                obj.append(Object(opan, otilt))
361
                           def closest(mp, mt):
362
```

minD - 10

goals9step7c.py Nov 30, 23 16:20 Page 5/5 flag = False for o in reversed(obj): 366 if (sqrt((o.pan-mp) **2+(o.tilt-mt) **2) <minR):</pre> 367 flag = True 368 minR = sqrt((o.pan-mp)**2+(o.tilt-mt)**2)369 minO = o370 if flag: 371 obj.remove(minO) 372 373 return minO, minR, flag 374 for m in reversed(monitor): cObj, cDist, exist = closest(m.pan, m.tilt) 375 376 if exist: 377 if cDist < R:</pre> 378 m = cObjm.conf = min(m.conf+0.2, 1)379 380 elif m.pan < minpan or m.pan > minpan or m.tilt < mintilt or m.tilt > maxtilt: 381 m.conf = m.conf-0.005**if** m.conf < -0.5: 382 monitor.remove(m) 383 else: 384 385 m.conf = m.conf-0.1print(len(obj)) 386 for o in reversed(obj): 387 388 monitor.append(o) """Lpan = 2.5//1.2 389 Ltilt = 1.71//1.47390 391 opan = pact[0][0] - shared.deltapan392 otilt = pact[1][0] + shared.deltatiltopan = pact[0][0] + vact[0][0] * Lpan - shared.deltapan otilt = pact[1][0] + vact[1][0] * Ltilt + shared.deltatilt""" 393 394 """mode = Mode.Spline 395 p0 = pact396 pf = np.array([[opan], [otilt]]) 397 v0 = vact398 vf = np.array([[0], [0]])399 Tmove = splinetime(p0, pf, v0, vf)400 (a, b, c, d) = spline parameters (Tmove, p0, v0, pf, vf)401 402 t0 = ttf = t0 + Tmove403 404 405 406 if(index<N):</pre> objectpan[index] = opan 407 objecttilt[index] = otilt 408 # Clear the data. 409 shared.newdata = False 410 shared.lock.release() 411 # Check whether we have a new object location. 412 # Advance the index/time. 413 414 index += 1+= 0.01 415