

Nov 30, 23 16:20

goals9step7c.py

Page 1/5

```

1  '''goals8step8c.py
2
3  Goals 8 Step 8
4
5  '''
6
7  # ----- SETUP THE KEYBOARD INTERFACE -----
8  # Import the necessary packages
9  import atexit, select, sys, termios
10
11 # Set up a handler, so the terminal returns to normal on exit.
12 stdattr = termios.tcgetattr(sys.stdin.fileno())
13 def reset_attr():
14     termios.tcsetattr(sys.stdin.fileno(), termios.TCSAFLUSH, stdattr)
15 atexit.register(reset_attr)
16
17 # Switch terminal to canonical mode: do not wait for <return> press.
18 newattr = termios.tcgetattr(sys.stdin.fileno())
19 newattr[3] = newattr[3] & ~termios.ICANON
20 termios.tcsetattr(sys.stdin.fileno(), termios.TCSAFLUSH, newattr)
21
22 # Define the kbhit() and getch() functions.
23 def kbhit():
24     return sys.stdin in select.select([sys.stdin], [], [], 0)[0]
25 def getch():
26     return sys.stdin.read(1)
27 #-----
28
29 # Import useful packages
30 import hebi
31 import numpy as np          # For future use
32 import matplotlib.pyplot as plt
33
34 from math import pi, sin, cos, asin, acos, atan2, sqrt
35 from time import sleep, time
36 import enum
37 def controller(shared):
38     class Mode(enum.Enum):
39         Hold=0
40         Spline=1
41         Scanning =2
42         Reset = 3
43         Setscan = 4
44
45     class Object:
46         def __init__(self, pan = 0.5, tilt = 0.5):
47             self.pan = pan
48             self.tilt = tilt
49             self.conf = 0.2
50
51     mode = Mode.Scanning
52
53     #
54     # HEBI Initialization
55     #
56     # Create the motor group, and pre-allocate the command and feedback
57     # data structures. Remember to set the names list to match your
58     # motor.
59     #
60     names = ['9.8', '2.1']
61     group = hebi.Lookup().get_group_from_names(['robotlab'], names)
62     if group is None:
63         print("Unable to find both motors " + str(names))
64         raise Exception("Unable to connect to motors")
65
66     command = hebi.GroupCommand(group.size)
67     feedback = hebi.GroupFeedback(group.size)
68
69     #setup time
70     t0 = 0.0
71     Tmove = 8          # 5 seconds
72     tf = t0 + Tmove
73     T=15
74
75     #
76     # Calibration
77     #
78     calib = np.array([[0], [-pi/6]])
79
80     #Find initial position and velocities
81     feedback = group.get_next_feedback(reuse_fbk=feedback)
82     pinit = np.array([feedback.position[0], feedback.position[1]])
83     p0 = np.array([[pinit[0]], [pinit[1]]])
84     pf = p0
85     phold = p0
86     v0 = np.array([[0.0], [0.0]])
87     vf = np.array([[0.0], [0.0]])
88
89     #
90     # Pre-allocate the storage

```

Nov 30, 23 16:20

goals9step7c.py

Page 2/5

```

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 # starred index
117 #
118 starred = 0
119 #
120 # calculate spline parameters
121 #
122 def splineparameters(Tmove, p0, v0, pf, vf):
123     a = p0
124     b = v0
125     c = 3*(pf-p0)/(Tmove**2)-vf/Tmove-2*v0/Tmove
126     d = -2*(pf-p0)/(Tmove**3)+vf/(Tmove**2)+v0/(Tmove**2)
127     return (a, b, c, d)
128
129 def splinetime(p0, pf, v0, vf):
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 #
134 # Create a plot of position and velocity, actual and command!
135 def plotPos():
136     pan = [m.pan for m in monitor]
137     tilt = [m.tilt for m in monitor]
138     plt.scatter(pan, tilt, marker = "X")
139     plt.xlim(-1.5,1.5)
140     plt.ylim(-1.0, 1.0)
141     plt.title('Goals 9 Step 6 – Monitor Multiple Detection')
142     plt.ylabel('Tilt')
143     plt.xlabel('Pan')
144     for m in monitor:
145         print("tilt ", m.tilt, "pan ", m.pan, "confidence ", m.conf)
146     plt.show()
147
148 def plot():
149     fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
150     ax1.plot(Time[0:index], PAct[0][0:index], color='blue', linestyle='-', label='Pan PAct')
151     ax1.plot(Time[0:index], PCmd[0][0:index], color='blue', linestyle='--', label='Pan PCmd')
152     ax1.plot(Time[0:index], objectpan[0:index], color='red', linestyle='-', label='Pan object')
153     ax1.plot(Time[0:index], PAct[1][0:index], color='green', linestyle='-', label='Tilt PAct')
154     ax1.plot(Time[0:index], PCmd[1][0:index], color='green', linestyle='--', label='Tilt PCmd')
155     ax1.plot(Time[0:index], objecttilt[0:index], color='black', linestyle='-', label='Tilt object')
156
157     ax2.plot(Time[0:index], VAct[0][0:index], color='blue', linestyle='-', label='Pan VAct')
158     ax2.plot(Time[0:index], VCmd[0][0:index], color='blue', linestyle='--', label='Pan VCmd')
159     ax2.plot(Time[0:index], VAct[1][0:index], color='green', linestyle='-', label='Tilt VAct')
160     ax2.plot(Time[0:index], VCmd[1][0:index], color='green', linestyle='--', label='Tilt VCmd')
161
162     ax1.set_title('Goals 9 Step 3 – Cammera Scale and Latency')
163     ax1.set_ylabel('Position (rad)')
164     ax2.set_ylabel('Velocity (rad/s)')
165     ax2.set_xlabel('Time (s)')
166
167     ax1.grid()
168     ax2.grid()
169     ax1.legend(loc = 'lower right')
170     ax2.legend(loc = 'lower right')
171     #plt.scatter(objectpan, objecttilt)
172     plt.show()
173
174 ##
175 ## Execute Movement
176 ##
177 while True:
178     # Read the actual data. This blocks (internally waits) 10ms for the data.
179     feedback = group.get_next_feedback(reuse_fb=feedback)
180     pact = np.array([[feedback.position[0]], [feedback.position[1]]])
181     vact = np.array([[feedback.velocity[0]], [feedback.velocity[1]]])
182     # Compute the commands for this time step

```

```

183     A = np.array([[1.0], [0.5]])
184     Tperiod = np.array([[4], [8]])
185     pcmd = A*np.sin(2*pi*(t-t0)/Tperiod)
186     vcmd = A*(2*pi/Tperiod)*np.cos(2*pi*(t-t0)/Tperiod)
187     if t >= tf and len(monitor) > 0:
188         plotPos()
189         starred = 0
190         mode = Mode.Spline
191         p0 = pcmd
192         v0 = vcmd
193         pf = np.array([monitor[0].pan], [monitor[0].pan])
194         phold = pf
195         vf = np.array([[0.0], [0.0]])
196         Tmove = splintime(p0, pf, v0, vf)
197         (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
198         t0 = t
199         tf = t0 + Tmove
200     elif mode is Mode.Setscan:
201         pcmd = (a + b*(t-t0) + c*(t-t0)**2+d*(t-t0)**3)
202         vcmd = (b + 2*c*(t-t0) + 3*d*(t-t0)**2)
203         if t>=tf:
204             phold= pcmd
205             mode = Mode.Scanning
206             t0 = t
207             tf = t0 + 8
208     elif mode is Mode.Hold:
209         pcmd = phold
210         vcmd = np.array([[0.0], [0.0]])
211         if t >= tf:
212             if starred >= len(monitor):
213                 mode = Mode.Setscan
214                 p0 = pcmd
215                 v0 = vcmd
216                 pf = np.array([0], [0])
217                 phold = pf
218                 vf = np.array([[0.0], [0.0]])
219                 Tmove = splintime(p0, pf, v0, vf)
220                 (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
221                 t0 = t
222                 tf = t0 + Tmove
223             else:
224                 mode = Mode.Spline
225                 p0 = pcmd
226                 v0 = vcmd
227                 pf = np.array([monitor[starred].pan], [monitor[starred].pan])
228                 phold = pf
229                 vf = np.array([[0.0], [0.0]])
230                 Tmove = splintime(p0, pf, v0, vf)
231                 (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
232                 t0 = t
233                 tf = t0 + Tmove
234     elif mode is Mode.Setscan:
235         p0 = pcmd
236         v0 = vcmd
237         pf = np.array([[0.0], [0.0]]) + calib
238         print("test:", pf)
239         vf = np.array([[1.0], [0.25]])
240         Tmove = splintime(p0, pf, v0, vf)
241         (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
242         t0 = t
243         tf = t0 + Tmove
244         mode = Mode.Reset
245     elif mode is Mode.Reset:
246         pcmd = (a + b*(t-t0) + c*(t-t0)**2+d*(t-t0)**3)
247         #print("pcmd", pcmd)
248         vcmd = (b + 2*c*(t-t0) + 3*d*(t-t0)**2)
249         #print("reset", pf, pcmd)
250         if t>=tf:
251             mode = Mode.Hold
252             t0 = t
253     elif mode is Mode.Spline:
254         pcmd = (a + b*(t-t0) + c*(t-t0)**2+d*(t-t0)**3)
255         vcmd = (b + 2*c*(t-t0) + 3*d*(t-t0)**2)
256         if t>=tf:
257             starred = starred + 1
258             phold= pcmd
259             mode = Mode.Hold
260             t0 = t
261             tf = t0 + 2
262     else:
263         print("this should never happen")
264     # Send the commands. This returns immediately.
265     command.position = pcmd
266     command.velocity = vcmd
267     group.send_command(command)
268
269     # Store the data for this time step (at the current index).
270     if(index< N):
271         Time[index] = t
272         pAct[0][index] = pAct[1][index] = pAct[0]
273         pAct[1][index] = pAct[1]

```

Nov 30, 23 16:20

goals9step7c.py

Page 4/5

```

274     VAct[0][index], VAct[1][index] = vact[0], vact[1]
275     VCmd[0][index], VCmd[1][index] = vcmd[0], vcmd[1]
276     objectpan[index] = opan
277     objecttilt[index] = otilt
278     elif index == N:
279         print("plot")
280         # plot()
281     # check if keyboard hit
282     if kbhit():
283         # Take action, based on the character.
284         ch = getch()
285         mode = Mode.Spline
286         if (ch == 'a'):
287             # make robot move without wait
288             p0 = pcmd
289             v0 = vcmd
290             pf = np.array([[ -0.15337068238377316], [ -0.037060368086649906]]) + calib
291             vf = np.array([[0.0], [0.0]])
292             Tmove = splinetime(p0, pf, v0, vf)
293             (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
294             print('A: pan: 60, tilt: 0')
295         elif (ch == 'b'):
296             p0 = pcmd
297             v0 = vcmd
298             pf = np.array([[ -pi/8], [pi/6]]) + calib
299             vf = np.array([[0.0], [0.0]])
300             Tmove = splinetime(p0, pf, v0, vf)
301             (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
302             print('B: pan: -45, tilt: -30')
303         elif (ch == 'c'):
304             p0 = pcmd
305             v0 = vcmd
306             pf = np.array([[0], [pi/8]]) + calib
307             vf = np.array([[0.0], [0.0]])
308             Tmove = splinetime(p0, pf, v0, vf)
309             (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
310             print('C: pan: 60, tilt: 45')
311         elif (ch == 'd'):
312             p0 = pcmd
313             v0 = vcmd
314             pf = np.array([[0], [pi/6]]) + calib
315             vf = np.array([[0.0], [0.0]])
316             Tmove = splinetime(p0, pf, v0, vf)
317             (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
318             print('D: pan: 0, tilt: -30')
319         elif (ch == 's'):
320             p0 = pcmd
321             v0 = vcmd
322             pf = np.array([[0], [0]])
323             vf = np.array([[0.0], [0.0]])
324             Tmove = splinetime(p0, pf, v0, vf)
325             (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
326             mode = Mode.Setscan
327             obj = []
328         elif (ch == 'q'):
329             # If we want to quit, break out of the loop.
330             print('Quitting')
331             break
332         else:
333             # Report the bad press.
334             print("Unknown character '%c'" % ch)
335             t0 = np.array(t)
336             tf = np.array(t0 + Tmove)
337     if shared.lock.acquire(timeout=0.001):
338         if shared.newdata and mode is Mode.Scanning:
339             # Compute the object angles.
340             #print("p", shared.deltapan, "t", shared.deltatilt)
341             # Lpan = 0.41709845
342             Lpan = 0.2
343             Lttilt = 0.8
344             R = 0.5
345             minpan = pact[0][0] - vact[0][0] * Lpan - shared.scalepan * 320
346             maxpan = pact[0][0] - vact[0][0] * Lpan + shared.scalepan * 320
347             mintilt = pact[1][0] + vact[1][0] * Lttilt - shared.scaletilt * 240
348             maxtilt = pact[1][0] + vact[1][0] * Lttilt + shared.scaletilt * 240
349             #Lttilt = 0.8100304
350             #Lpan = 0.
351             #Lttilt = 0.1
352             #opan = pact[0][0] - shared.deltapan
353             print("shared", shared.balls)
354             print("monitored", monitor)
355             obj = []
356             for ball in shared.balls:
357                 #print(ball.deltapan, ball.deltatilt)
358                 opan = pact[0][0] - vact[0][0] * Lpan - ball.deltapan #+ vact[0][0] * Lpan
359                 otilt = pact[1][0] + vact[1][0] * Lttilt + ball.deltatilt
360
361             obj.append(Object(opan, otilt))
362     def closest(mp, mt):
363         minR = 100

```

Nov 30, 23 16:20

goals9step7c.py

Page 5/5

```

365         flag = False
366         for o in reversed(obj):
367             if (sqrt((o.pan-mp)**2+(o.tilt-mt)**2)<minR):
368                 flag = True
369                 minR = sqrt((o.pan-mp)**2+(o.tilt-mt)**2)
370                 minO = o
371         if flag:
372             obj.remove(minO)
373         return minO, minR, flag
374     for m in reversed(monitor):
375         cObj, cDist, exist = closest(m.pan, m.tilt)
376         if exist:
377             if cDist < R:
378                 m = cObj
379                 m.conf = min(m.conf+0.2, 1)
380             elif m.pan < minpan or m.pan > minpan or m.tilt < mintilt or m.tilt > maxtilt:
381                 m.conf = m.conf-0.005
382             if m.conf < -0.5:
383                 monitor.remove(m)
384         else:
385             m.conf = m.conf-0.1
386     print(len(obj))
387     for o in reversed(obj):
388         monitor.append(o)
389     """Lpan = 2.5//1.2
390     Ltlt = 1.71//1.47
391     opan = pact[0][0] - shared.deltapan
392     otlt = pact[1][0] + shared.deltatilt
393     opan = pact[0][0] + vact[0][0] * Lpan - shared.deltapan
394     otlt = pact[1][0] + vact[1][0] * Ltlt + shared.deltatilt"""
395     """mode = Mode.Spline
396     p0 = pact
397     pf = np.array([[opan], [otilt]])
398     v0 = vact
399     vf = np.array([[0], [0]])
400     Tmove = splinetime(p0, pf, v0, vf)
401     (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
402     t0 = t
403     tf = t0 + Tmove
404     print("a=", a, "b=", b, "c =", c, "d=", d)
405     print('going to object')"""
406     if (index<N):
407         objectpan[index] = opan
408         objecttilt[index] = otlt
409         # Clear the data.
410         shared.newdata = False
411         shared.lock.release()
412         # Check whether we have a new object location.
413         # Advance the index/time.
414         index += 1
415         t += 0.01

```