```
"""goals8system.py

  This is the main script that coordinates both the controller and
  the detector.  Feel free to play/edit/...

"""

# Import the system parts
import threading
import traceback

# Import your pieces.  Change the "from" names (being the file names)
# to the file names of your two code parts.  Also, this is a great way
# to quickly switch which detector to run.  E.g. you could import from
# "facedetector" in place or "balldetector".
from goals9step6c   import controller
from goals9step6d import detector
# from goals8facedetector import detector     # Alternate option


#
#  Shared Data
#
class Shared:
    def __init__(self):
        # Thread Lock.  Always acquire() this lock before accessing
        # anything else (either reading or writing) in this object.
        # And don't forget to release() the lock when done!
        self.lock = threading.Lock()

        # Flag - stop the detection.  If this is set to True, the
        # detection should break out of the loop and stop.
        self.stop = False

        # Data - PLEASE UPDATE TO ADD THE DATA YOU NEED!
        self.newdata = False # Flag to indicate fresh, new data
        self.deltapan = 0.0 # Relative pan angle
        self.deltatilt = 0.0 # Relative tilt angle
        self.scalepan = 0.0
        self.sharetilt = 0.0
        self.balls = []

#
#  Main Code
#
def main():
    # Prepare the shared data object.
    shared = Shared()

    # Create a second thread.
    thread = threading.Thread(target=detector, args=(shared,))

    # Start the second thread with the detector.
    print("Starting second thread")
    thread.start()      # Equivalent to detector(shared) in new thread

    # Use the primary thread for the controller, handling exceptions
    # to gracefully to shut down.
    try:
        controller(shared)
    except BaseException as ex:
        # Report the exception
        print("Ending due to exception: %s" % repr(ex))
        traceback.print_exc()

    # Stop/rejoin the second thread.
    print("Stopping second thread...")
    if shared.lock.acquire():
        shared.stop = True
```

```
71      thread.join()        # Wait for thread to end and re-combine.
72
73  if __name__ == "__main__":
74      main()
```

```
'''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
```

```
"""goals8step8d.py

  Run the face (and eye) detectors and show the results.
"""

# Import OpenCV
import numpy as np
import cv2

def detector(shared):
    class Ball:
        def __init__(self, pan = 0.5, tilt = 0.5):
            self.deltapan = pan
            self.deltatilt = tilt
            self.conf = 0.2

    # Set up video capture device (camera).  Note 0 is the camera number.
    # If things don't work, you may need to use 1 or 2?
    #camera = cv2.VideoCapture(0, cv2.CAP_V4L2)
    camera = cv2.VideoCapture(0, cv2.CAP_ANY)

    if not camera.isOpened():
        raise Exception("Could not open video device: Maybe change the cam number?")

    # Change the frame size and rate.  Note only combinations of
    # widthxheight and rate are allowed.  In particular, 1920x1080 only
    # reads at 5 FPS.  To get 30FPS we downsize to 640x480.
    camera.set(cv2.CAP_PROP_FRAME_WIDTH,  640)
    camera.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
    camera.set(cv2.CAP_PROP_FPS,           30)

    # Get the face/eye detector models from XML files.  Instantiate detectors.
    faceXML = "haarcascade_frontalface_default.xml"
    eyeXML1 = "haarcascade_eye.xml"
    eyeXML2 = "haarcascade_eye_tree_eyeglasses.xml"

    faceDetector = cv2.CascadeClassifier(faceXML)
    eyeDetector  = cv2.CascadeClassifier(eyeXML1)

    # Pick some colors.  Note OpenCV uses BGR color codes by default.
    blue  = (255,  0,  0)
    green = (  0,255,  0)
    red   = (  0,  0,255)
    white = (255,255,255)

    hsv = np.array([])
    bgr = np.array([])

    #
    # scale
    #
    scalepan = 0.00161932589506
    scaletilt = 0.00133859370275
    # Keep scanning, until 'q' hit IN IMAGE WINDOW.
    while True:
        # Grab an image from the camera.  Often called a frame (part of sequence).
        ret, frame = camera.read()

        #convert image to hsv
        hsv = cv2.cvtColor(frame, cv2.COLOR_RGB2HSV)

        # Convert the image to gray scale.
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Grab the faces - the cascade detector returns a list faces.
        faces = faceDetector.detectMultiScale(gray,
            scaleFactor = 1.2,
            minNeighbors = 5,
            minSize = (30,30),
            flags = cv2.CASCADE_SCALE_IMAGE)

        # Process the faces: Each face is a bounding box of (x,y,w,h)
        # coordinates.  Draw the bounding box ON THE ORIGINAL IMAGE.
        if len(faces) > 0:
            # Grab the first face.
            face = faces[0]

            # Grab the face coodinates.
            (x, y, w, h) = face

            # Draw the bounding box on the original color frame.
            #cv2.rectangle(frame, (x, y), (x+w-1, y+h-1), green, 3)
```

```
84              # Also look for eyes - only within the region of the face!
85              # This similarly a list of eyes relative to this region.
86              eyes = eyeDetector.detectMultiScale(gray[y:y+h,x:x+w])
87
88              # Process the eyes: As before, eyes is a list of bounding
89              # boxes (x,y,w,h) relative to the processed region.
90              for (xe,ye,we,he) in eyes:
91                  # Can you draw circles around the eyes?  Consider the function:
92                  # cv2.circle(frame, (xc, yc), radius, (b,g,r), linewidth)
93                  pass # replace this.
94
95
96          # Show the image with the given title.  Note this won't actually
97          # appear (draw on screen) until the waitKey(1) below.
98          (H, W, D) = frame.shape
99          #np.append(bgr, frame[W//2, H//2])
100         #np.append(hsv, hsv[W//2, H//2])
101         #print("Frame: ", frame[W//2, H//2])
102         #print("HSV: ", hsv[W//2, H//2])
103
104         #Convert to binary
105         binary = cv2.inRange(hsv, (113, 126, 78), (151, 255, 225))
106         #binary = cv2.inRange(hsv, (107, 142, 119), (119, 255, 255))
107
108         #Erode, then Dialate
109         binary=cv2.erode( binary, None, iterations=3)
110         binary=cv2.dilate(binary, None, iterations=3)
111
112         #Dilate, then Erode
113         binary=cv2.dilate(binary, None, iterations=3)
114         binary=cv2.erode(binary, None, iterations=3)
115
116         #Contours
117         (contours, hierarchy) = cv2.findContours(binary, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
118         contoursTemp = sorted(contours, key=cv2.contourArea, reverse=True)
119         contours=[]
120         (xmax,ymax,rmax) = (0, 0, 0)
121         balls = []
122         for contour in contoursTemp:
123             if cv2.contourArea(contour) > 2000:
124                 contours.append(contour)
125                 ((xr, yr), radius) = cv2.minEnclosingCircle(contour)
126                 cv2.circle(frame, (int(xr), int(yr)), 4, (91, 252, 104), -1)
127                 cv2.circle(frame, (int(xr), int(yr)), int(radius), (91, 252, 104), 3)
128                 x = scalepan * float(xr - 320)
129                 y = scaletilt * float(yr - 240)
130                 balls.append(Ball(x, y))
131                 #print(xr, yr)
132         if shared.lock.acquire():
133             """if rmax > 0:
134     (x0, y0, x, y) = (320, 240, xmax, ymax)
135     shared.deltapan = scalepan * float(x – x0)
136     shared.deltatilt = scaletilt * float(y – y0)
137     shared.newdata = True"""
138             shared.balls = balls
139             shared.lock.release()
140             shared.scalepan = scalepan
141             shared.scaletilt = scaletilt
142             shared.newdata = True
143
144         """ENCLOSING CIRCLE"""
145
146
147         cv2.line(frame, (W//2,0), (W//2,H), (0,0,255), 1)
148         cv2.line(frame, (0,H//2), (W, H//2), (0,0,255), 1)
149         cv2.line(hsv, (W//2,0), (W//2,H), (0,0,255), 1)
150         cv2.line(hsv, (0,H//2), (W, H//2), (0,0,255), 1)
151         cv2.imshow('Processed Image', frame)
152         #cv2.imshow('HSV Funkiness', hsv)
153         cv2. imshow('Binary Image', binary)
154
155         # Check for a key press IN THE IMAGE WINDOW: waitKey(0) blocks
156         # indefinitely, waitkey(1) blocks for at most 1ms.  If 'q' break.
157         # This also flushes the windows and causes it to actually appear.
158         if (cv2.waitKey(1) & 0xFF) == ord('q'):
159             break
160         # Check if the main thread signals this loop to end.
161         if shared.lock.acquire():
162             stop = shared.stop
163             shared.lock.release()
164             if stop:
165                 break
```

```
167         camera.release()
168         cv2.destroyAllWindows()
169
170 if __name__ == "__main__":
171         controller()
```

```
'''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.Scanning

    #
    #  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 = 8                   # 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     # 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='Til 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     ## Execute Movement
175     ##
176     while True:
177         # Read the actual data. This blocks (internally waits) 10ms for the data.
178         feedback = group.get_next_feedback(reuse_fbk=feedback)
179         pact = np.array([[feedback.position[0]], [feedback.position[1]]])
180         vact = np.array([[feedback.velocity[0]], [feedback.velocity[1]]])
181         # 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 = splinetime(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 = splinetime(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 = splinetime(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 = splinetime(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], pact[1]
```

```
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                 Ltilt = 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] * Ltilt - shared.scaletilt * 240
348                 maxtilt = pact[1][0] + vact[1][0] * Ltilt + shared.scaletilt * 240
349                 #Ltilt = 0.8100304
350                 #Lpan = 0.
351                 #Ltilt = 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] * Ltilt + ball.deltatilt
360 
361                     obj.append(Object(opan, otilt))
362                 def closest(mp, mt):
363                     minR = 100
```

```
                        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 < -0.5:
                                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
            Ltilt = 1.71//1.47
            opan = pact[0][0] – shared.deltapan
            otilt = pact[1][0] + shared.deltatilt
            opan = pact[0][0] + vact[0][0] * Lpan – shared.deltapan
            otilt = pact[1][0] + vact[1][0] * Ltilt + shared.deltatilt"""
                    """mode = Mode.Spline
            p0 = pact
            pf = np.array([[opan], [otilt]])
            v0 = vact
            vf = np.array([[0], [0]])
            Tmove = splinetime(p0, pf, v0, vf)
            (a, b, c, d) = splineparameters(Tmove, p0, v0, pf, vf)
            t0 = t
            tf = t0 + Tmove
            print("a=", a, "b=", b, "c =",c, "d=",d)
            print('going to object')"""
                    if(index<N):
                        objectpan[index] = opan
                        objecttilt[index] = otilt
                        # Clear the data.
                    shared.newdata = False
                shared.lock.release()
            # Check whether we have a new object location.
            # Advance the index/time.
            index += 1
            t     += 0.01
```

```
"""goals8step8d.py

  Run the face (and eye) detectors and show the results.
"""

# Import OpenCV
import numpy as np
import cv2

def detector(shared):
    class Ball:
        def __init__(self, pan = 0.5, tilt = 0.5):
            self.deltapan = pan
            self.deltatilt = tilt
            self.conf = 0.2

    # Set up video capture device (camera).  Note 0 is the camera number.
    # If things don't work, you may need to use 1 or 2?
    #camera = cv2.VideoCapture(0, cv2.CAP_V4L2)
    camera = cv2.VideoCapture(0, cv2.CAP_ANY)

    if not camera.isOpened():
        raise Exception("Could not open video device: Maybe change the cam number?")

    # Change the frame size and rate.  Note only combinations of
    # widthxheight and rate are allowed.  In particular, 1920x1080 only
    # reads at 5 FPS.  To get 30FPS we downsize to 640x480.
    camera.set(cv2.CAP_PROP_FRAME_WIDTH,  640)
    camera.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
    camera.set(cv2.CAP_PROP_FPS,           30)

    # Get the face/eye detector models from XML files.  Instantiate detectors.
    faceXML = "haarcascade_frontalface_default.xml"
    eyeXML1 = "haarcascade_eye.xml"
    eyeXML2 = "haarcascade_eye_tree_eyeglasses.xml"

    faceDetector = cv2.CascadeClassifier(faceXML)
    eyeDetector  = cv2.CascadeClassifier(eyeXML1)

    # Pick some colors.  Note OpenCV uses BGR color codes by default.
    blue  = (255,  0,  0)
    green = (  0,255,  0)
    red   = (  0,  0,255)
    white = (255,255,255)

    hsv = np.array([])
    bgr = np.array([])

    #
    # scale
    #
    scalepan = 0.00161932589506
    scaletilt = 0.00133859370275
    # Keep scanning, until 'q' hit IN IMAGE WINDOW.
    while True:
        # Grab an image from the camera.  Often called a frame (part of sequence).
        ret, frame = camera.read()

        #convert image to hsv
        hsv = cv2.cvtColor(frame, cv2.COLOR_RGB2HSV)

        # Convert the image to gray scale.
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Grab the faces - the cascade detector returns a list faces.
        faces = faceDetector.detectMultiScale(gray,
            scaleFactor = 1.2,
            minNeighbors = 5,
            minSize = (30,30),
            flags = cv2.CASCADE_SCALE_IMAGE)

        # Process the faces: Each face is a bounding box of (x,y,w,h)
        # coordinates.  Draw the bounding box ON THE ORIGINAL IMAGE.
        if len(faces) > 0:
            # Grab the first face.
            face = faces[0]

            # Grab the face coodinates.
            (x, y, w, h) = face

            # Draw the bounding box on the original color frame.
            #cv2.rectangle(frame, (x, y), (x+w-1, y+h-1), green, 3)
```

```
84              # Also look for eyes - only within the region of the face!
85              # This similarly a list of eyes relative to this region.
86              eyes = eyeDetector.detectMultiScale(gray[y:y+h,x:x+w])
87
88              # Process the eyes: As before, eyes is a list of bounding
89              # boxes (x,y,w,h) relative to the processed region.
90              for (xe,ye,we,he) in eyes:
91                  # Can you draw circles around the eyes?  Consider the function:
92                  # cv2.circle(frame, (xc, yc), radius, (b,g,r), linewidth)
93                  pass # replace this.
94
95
96          # Show the image with the given title.  Note this won't actually
97          # appear (draw on screen) until the waitKey(1) below.
98          (H, W, D) = frame.shape
99          #np.append(bgr, frame[W//2, H//2])
100         #np.append(hsv, hsv[W//2, H//2])
101         #print("Frame: ", frame[W//2, H//2])
102         #print("HSV: ", hsv[W//2, H//2])
103
104         #Convert to binary
105         binary = cv2.inRange(hsv, (113, 126, 78), (151, 255, 225))
106         #binary = cv2.inRange(hsv, (107, 142, 119), (119, 255, 255))
107
108         #Erode, then Dialate
109         binary=cv2.erode( binary, None, iterations=3)
110         binary=cv2.dilate(binary, None, iterations=3)
111
112         #Dilate, then Erode
113         binary=cv2.dilate(binary, None, iterations=3)
114         binary=cv2.erode(binary, None, iterations=3)
115
116         #Contours
117         (contours, hierarchy) = cv2.findContours(binary, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
118         contoursTemp = sorted(contours, key=cv2.contourArea, reverse=True)
119         contours=[]
120         (xmax,ymax,rmax) = (0, 0, 0)
121         balls = []
122         for contour in contoursTemp:
123             if cv2.contourArea(contour) > 2000:
124                 contours.append(contour)
125                 ((xr, yr), radius) = cv2.minEnclosingCircle(contour)
126                 cv2.circle(frame, (int(xr), int(yr)), 4, (91, 252, 104), -1)
127                 cv2.circle(frame, (int(xr), int(yr)), int(radius), (91, 252, 104), 3)
128                 x = scalepan * float(xr - 320)
129                 y = scaletilt * float(yr - 240)
130                 balls.append(Ball(x, y))
131                 #print(xr, yr)
132         if shared.lock.acquire():
133             """if rmax > 0:
134     (x0, y0, x, y) = (320, 240, xmax, ymax)
135     shared.deltapan = scalepan * float(x – x0)
136     shared.deltatilt = scaletilt * float(y – y0)
137     shared.newdata = True"""
138             shared.balls = balls
139             shared.lock.release()
140             shared.scalepan = scalepan
141             shared.scaletilt = scaletilt
142             shared.newdata = True
143
144         """ENCLOSING CIRCLE"""
145
146
147         cv2.line(frame, (W//2,0), (W//2,H), (0,0,255), 1)
148         cv2.line(frame, (0,H//2), (W, H//2), (0,0,255), 1)
149         cv2.line(hsv, (W//2,0), (W//2,H), (0,0,255), 1)
150         cv2.line(hsv, (0,H//2), (W, H//2), (0,0,255), 1)
151         cv2.imshow('Processed Image', frame)
152         #cv2.imshow('HSV Funkiness', hsv)
153         cv2. imshow('Binary Image', binary)
154
155         # Check for a key press IN THE IMAGE WINDOW: waitKey(0) blocks
156         # indefinitely, waitkey(1) blocks for at most 1ms.  If 'q' break.
157         # This also flushes the windows and causes it to actually appear.
158         if (cv2.waitKey(1) & 0xFF) == ord('q'):
159             break
160         # Check if the main thread signals this loop to end.
161         if shared.lock.acquire():
162             stop = shared.stop
163             shared.lock.release()
164             if stop:
165                 break
```

```
167         camera.release()
168         cv2.destroyAllWindows()
169
170 if __name__ == "__main__":
171         controller()
```