```
"""goals6simple.py

  Read the camera image in preparation for some image manipulation
  and object detection.

"""
# Import OpenCV
import cv2
def detector(shared):

    # 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)
    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)

    # Change the camera settings.
    exposure = 166
    wb = 3377
    focus = 0

    #camera.set(cv2.CAP_PROP_AUTO_EXPOSURE, 3)       # Auto mode
    camera.set(cv2.CAP_PROP_AUTO_EXPOSURE, 1)       # Manual mode
    camera.set(cv2.CAP_PROP_EXPOSURE, exposure)     # 3 - 2047, default 250

    #camera.set(cv2.CAP_PROP_AUTO_WB, 1.0)           # Enable auto white balance
    camera.set(cv2.CAP_PROP_AUTO_WB, 0.0)           # Disable auto white balance
    camera.set(cv2.CAP_PROP_WB_TEMPERATURE, wb)     # 2000 - 6500, default 4000

    #camera.set(cv2.CAP_PROP_AUTOFOCUS, 1)           # Enable autofocus
    camera.set(cv2.CAP_PROP_AUTOFOCUS, 0)           # Disable autofocus
    camera.set(cv2.CAP_PROP_FOCUS, focus)           # 0 - 250, step 5, default 0

    camera.set(cv2.CAP_PROP_BRIGHTNESS, 128)        # 0 - 255, default 128
    camera.set(cv2.CAP_PROP_CONTRAST,   128)        # 0 - 255, default 128
    camera.set(cv2.CAP_PROP_SATURATION, 190)        # 0 - 255, default 128


    # Keep scanning, until 'q' hit IN IMAGE WINDOW.

    count = 0
    while True:
        # Grab an image from the camera.  Often called a frame (part of sequence).
        ret, frame = camera.read()
        if shared.lock.acquire():
            camerapan = shared.motorpan
            cameratilt = shared.motortilt
            shared.lock.release()
        count += 1

        # Grab and report the image shape.
        (H, W, D) = frame.shape

        # Convert the BGR image to RGB or HSV.
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)      # For other objects

        binary = cv2.inRange(hsv, (25, 108, 130), (34, 215, 228))
        binary = cv2.erode(binary, None, iterations=1)
        binary = cv2.dilate(binary, None, iterations=1)
        binary = cv2.dilate(binary, None, iterations=7)
        binary = cv2.erode(binary, None, iterations=7)

        #contours
        (contours, hierarchy) = cv2.findContours(binary, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
        contours=sorted(contours, key=cv2.contourArea, reverse=True)

        #down-select contours
        for contour in contours:
            if cv2.contourArea(contour) > 1300:
                cv2.drawContours(frame, contour, -1, (200, 213, 48), 10)

                ((xr,yr), radius) = cv2.minEnclosingCircle(contour)
                cv2.circle(frame, (int(xr),int(yr)), int(radius), (180, 105, 255), 2)


                print(f"(x, y) for the Enclosing circle is ({int(xr)},{int(yr)})")
```

```
84
85                  pan_object = (-0.001467)*(int(xr) - (W//2)) + camerapan
86                  tilt_object = (-0.001462)*(int(yr) - (H//2)) + cameratilt
87
88                  if shared.lock.acquire():
89                      shared.objectpan = pan_object
90                      shared.objecttilt = tilt_object
91                      shared.new_data = True
92                      shared.lock.release()
93                  break
94
95          # Show the processed image with the given title.  Note this won't
96          # actually appear (draw on screen) until the waitKey(1) below.
97          cv2.imshow('Processed Image', frame)
98          cv2.imshow('Binary Image', binary)
99
100         # Check for a key press IN THE IMAGE WINDOW: waitKey(0) blocks
101         # indefinitely, waitkey(1) blocks for at most 1ms.  If 'q' break.
102         # This also flushes the windows and causes it to actually appear.
103         if (cv2.waitKey(1) & 0xFF) == ord('q'):
104             break
105         if shared.lock.acquire():
106             stop = shared.stop
107             shared.lock.release()
108             if stop:
109                 break
110     # Close everything up.
111     camera.release()
112     cv2.destroyAllWindows()
113
114 if __name__ =="__main__":
115     detector(None)
```

```
"""goals7system.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 motor_mover_5   import controller
from ball_detector_5     import detector
# from goals7facedetector 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

        self.new_data = False

        # Motor data
        self.motorpan  = 0.0
        self.motortilt = 0.0
        self.objectpan = 0.0
        self.objecttilt = 0.0

        # Object Data - PLEASE UPDATE TO ADD THE DATA YOU NEED!


#
#  Main Code
#
def main():
    # Prepare a single instance of 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...")
```

```
71      if shared.lock.acquire():
72          shared.stop = True
73          shared.lock.release()
74      thread.join()        # Wait for thread to end and re-combine.
75
76  if __name__ == "__main__":
77      main()
```

```
'''goals3democode.py

  Demo code for Goals 3
'''
# Import useful packages
import hebi
import numpy as np              # For future use
import matplotlib.pyplot as plt

from math import pi, sin, cos, asin, acos, atan2, sqrt, inf
from time import sleep, time
from keycheck import kbhit, getch

from enum import Enum



def controller(shared):

    #
    #  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 = ['4.6', '6.2']
    group = hebi.Lookup().get_group_from_names(['robotlab'], names)
    if group is None:
      print("Unable to find both motors " + str(names))
      raise Exception("Unable to connect to motors")

    command  = hebi.GroupCommand(group.size)
    feedback = hebi.GroupFeedback(group.size)

    dt = 0.01                       # HEBI feedback comes in at 100Hz!

    #
    #   PARAMATERS
    #
    class Traj(Enum):
        HOLD = 0 # Keeps a constant pos, zero velocity forever
        SPLINE = 1 # Computes a cubic spline, ends at tf
        SCAN = 2 # Computes sinusoidal pos/vel, never ends

    class Mode(Enum):
        GOHOME = 0 # GO TO POSITION (0,0)
        TRACKING = 1 #Track the primary object of intrest
        SCANNING = 2 #Scan the entire field of view w/o tracking

    offset_p = 0.19
    offset_t = 0.39
    v0 = [0.0, 0.0]
    v_max = [2.72, 2.00]
    A = [(75*pi)/180, (30*pi)/180]
    t = 0.0
    t0 = 0.0

    feedback = group.get_next_feedback(reuse_fbk=feedback)
    pinit_pan = feedback.position[0] + offset_p
    pinit_tilt = feedback.position[1] + offset_t
    p0=[pinit_pan, pinit_tilt]

    traj = Traj.HOLD
    mode = Mode.GOHOME
    phold = p0

    #
    #FUNCTIONS
    #
    def movetime(p0, pf, v_max, v0, vf):
        """Computes the time required to move between p0 and pf."""
        distance = abs(pf - p0)
        tm = ((6*distance) / ((4*v_max) + (abs(v0)+ abs(vf))))

        tm = tm + (abs(v0))*0.75
        tm = tm + (abs(vf))*0.75

        if tm < 0.25:
            return 0.25
        return tm

    def calcparams(t0, tf, p0, pf, v0, vf):
        """Computes the cubic spline parameters a, b, c, d."""
        a = p0
        b = v0
        c = (3 * (pf - p0) / (tf-t0)**2) - (2 * v0 + vf) / (tf-t0)
        d = (-2 * (pf - p0) / (tf-t0)**3) + (v0 + vf) / (tf-t0)**2
        return a, b, c, d

    def splinecmds(t,a,b,c,d):
        pcmd = a + b * (t) + c * t**2 + d * (t)**3
        vcmd = b + 2 * c * (t) + 3 * d * (t)**2
        return pcmd, vcmd

    def scancmds(A, t, t0, tscan, motor):
        if motor == True:
            pcmd = A * (sin(2*pi*((t-t0)/tscan)))
            vcmd = ((A*2*pi)/tscan) * (cos(2*pi*((t-t0)/tscan)))
            return pcmd, vcmd
        else:
            pcmd = A * (sin(8*pi*((t-t0)/tscan)))
            vcmd = ((A*8*pi)/tscan) * (cos(8*pi*((t-t0)/tscan)))
```

```
104             return pcmd, vcmd
105
106     # Data for plotting
107     Time, PAct_Pan, PCmd_Pan, VAct_Pan, VCmd_Pan, Verror_Pan, Perror_Pan, OBJ_Pan = [], [], [], [], [], [], [], []
108     PAct_Tilt, PCmd_Tilt, VAct_Tilt, VCmd_Tilt, Verror_Tilt, Perror_Tilt, OBJ_Tilt = [], [], [], [], [], [],[]
109
110     # Main control loop
111     while True:
112         # Compute the current position and velocity commands
113         if traj is Traj.SPLINE:
114             pcmd_pan, vcmd_pan = splinecmds((t - t0), a_p, b_p, c_p, d_p)
115             pcmd_tilt, vcmd_tilt = splinecmds((t - t0), a_t, b_t, c_t, d_t)
116         elif traj is Traj.SCAN:
117             pcmd_pan, vcmd_pan = scancmds(A[0], t, t0, tm, True)
118             pcmd_tilt, vcmd_tilt = scancmds(A[1], t, t0, tm, False)
119         elif traj is Traj.HOLD:
120             pcmd_pan, pcmd_tilt = phold
121             vcmd_pan, vcmd_tilt = [0.0, 0.0]
122
123         else:
124             raise ValueError(f'Bad trajectory type {traj}')
125
126         # Send commands to the motor
127         feedback = group.get_next_feedback(reuse_fbk=feedback)
128         pact_pan = feedback.position[0] + offset_p
129         pact_tilt = feedback.position[1] + offset_t
130         vact_pan = feedback.velocity[0]
131         vact_tilt = feedback.velocity[1]
132
133         if shared.lock.acquire():
134             shared.motorpan = pcmd_pan
135             shared.motortilt = pcmd_tilt
136             objectpan = shared.objectpan
137             objecttilt = shared.objecttilt
138             shared.lock.release()
139
140
141         command.position = [pcmd_pan-offset_p, pcmd_tilt-offset_t]
142         command.velocity = [vcmd_pan, vcmd_tilt]
143         group.send_command(command)
144
145         # Store data for plotting
146         Time.append(t)
147         OBJ_Pan.append(objectpan)
148         OBJ_Tilt.append(objecttilt)
149
150         PAct_Pan.append(pact_pan)
151         PCmd_Pan.append(pcmd_pan)
152         VAct_Pan.append(vact_pan)
153         VCmd_Pan.append(vcmd_pan)
154
155         PAct_Tilt.append(pact_tilt)
156         PCmd_Tilt.append(pcmd_tilt)
157         VAct_Tilt.append(vact_tilt)
158         VCmd_Tilt.append(vcmd_tilt)
159
160         Perror_Pan.append((pact_pan) - (pcmd_pan))
161         Verror_Pan.append(vact_pan - vcmd_pan)
162
163         Perror_Tilt.append((pact_tilt) - (pcmd_tilt))
164         Verror_Tilt.append(vact_tilt - vcmd_tilt)
165
166         if abs(vcmd_pan) > v_max[0] or abs(vcmd_tilt) > v_max[1]:
167             print("Exceeding max vel!")
168
169         # Check for key presses
170         if kbhit():
171             ch = getch()
172             if ch == 's':
173                 if mode is not Mode.SCANNING:
174                     traj = Traj.SPLINE
175                     mode = Mode.SCANNING
176                     t0 = t
177                     p0 = [pcmd_pan, pcmd_tilt]
178                     v0 = [vcmd_pan, vcmd_tilt]
179
180                     pf = [0.0, 0.0]
181                     vf = [((A[0]*2*pi)/((A[1]*8*pi)/v_max[1])), ((A[1]*8*pi)/((A[1]*8*pi)/v_max[1]))]
182                     tm = max(movetime(p0[0], pf[0], v_max[0], v0[0], vf[0]), movetime(p0[1], pf[1], v_max[1], v0[1], vf[
    1]))
183
184                     tf = tm + t0
185                     a_p, b_p, c_p, d_p = calcparams(t0, tf, p0[0], pf[0], v0[0], vf[0])
186                     a_t, b_t, c_t, d_t = calcparams(t0, tf, p0[1], pf[1], v0[1], vf[1])
187
188             elif ch == 'z':
189                 traj = Traj.SPLINE
190                 mode = Mode.GOHOME
191                 t0 = t
192                 p0 = [pcmd_pan, pcmd_tilt]
193                 v0 = [vcmd_pan, vcmd_tilt]
194
195                 pf = [0.0, 0.0]
196
197                 vf = [0.0, 0.0]
198                 tm = max(movetime(p0[0], pf[0], v_max[0], v0[0], vf[0]), movetime(p0[1], pf[1], v_max[1], v0[1], vf[1]))
199
200                 tf = tm + t0
201                 a_p, b_p, c_p, d_p = calcparams(t0, tf, p0[0], pf[0], v0[0], vf[0])
202                 a_t, b_t, c_t, d_t = calcparams(t0, tf, p0[1], pf[1], v0[1], vf[1])
203             elif ch == 't':
204                 if mode is not Mode.TRACKING:
205                     traj = Traj.SPLINE
```

```
                    mode = Mode.TRACKING
                    t0 = t
                    p0 = [pcmd_pan, pcmd_tilt]
                    v0 = [vcmd_pan, vcmd_tilt]

                    pf = p0

                    vf = [0.0, 0.0]
                    tm = max(movetime(p0[0], pf[0], v_max[0], v0[0], vf[0]), movetime(p0[1], pf[1], v_max[1], v0[1], vf[
1]))

                    tf = tm + t0
                    a_p, b_p, c_p, d_p = calcparams(t0, tf, p0[0], pf[0], v0[0], vf[0])
                    a_t, b_t, c_t, d_t = calcparams(t0, tf, p0[1], pf[1], v0[1], vf[1])
            elif ch == 'q':
                # Quit the program
                break

        if shared.lock.acquire():
            new_object_data = shared.new_data
            shared.lock.release()
        if new_object_data and mode is Mode.TRACKING:
            traj = Traj.SPLINE
            t0 = t
            p0 = [pcmd_pan, pcmd_tilt]
            v0 = [vcmd_pan, vcmd_tilt]

            if shared.lock.acquire():
                objectpan = shared.objectpan
                objecttilt = shared.objecttilt
                shared.new_data = False
                shared.lock.release()

            pf = [objectpan, objecttilt]
            vf = [0.0, 0.0]

            tm = max(movetime(p0[0], pf[0], v_max[0], v0[0], vf[0]), movetime(p0[1], pf[1], v_max[1], v0[1], vf[1]))

            tf = tm + t0
            a_p, b_p, c_p, d_p = calcparams(t0, tf, p0[0], pf[0], v0[0], vf[0])
            a_t, b_t, c_t, d_t = calcparams(t0, tf, p0[1], pf[1], v0[1], vf[1])



        if traj is Traj.SPLINE and t+dt > tf:
            if mode is Mode.SCANNING:
                traj = Traj.SCAN
                t0 = t
                p0 = [pcmd_pan, pcmd_tilt]
                v0 = [vcmd_pan, vcmd_tilt]


                tm = max((A[0]*2*pi)/v_max[0], (A[1]*8*pi)/v_max[1])

                tf = tm + t0
            elif mode is Mode.TRACKING:
                traj = Traj.HOLD
                phold = [pcmd_pan, pcmd_tilt]
            elif mode is Mode.GOHOME:
                traj = Traj.HOLD
                phold = [pcmd_pan, pcmd_tilt]
            else:
                raise ValueError('Unexpected end of motion')

        # Advance time
        t += dt


    # Plot the results
    fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
    ax1.plot(Time[0:len(Time)], PAct_Tilt[0:len(PAct_Tilt)], color='green', linestyle='-',  label='Act_Tilt')
    ax1.plot(Time[0:len(Time)], PCmd_Tilt[0:len(PCmd_Tilt)], color='green', linestyle='--', label='Cmd_Tilt')
    ax2.plot(Time[0:len(Time)], VAct_Tilt[0:len(VAct_Tilt)], color='green', linestyle='-',  label='Act_Tilt')
    ax2.plot(Time[0:len(Time)], VCmd_Tilt[0:len(VCmd_Tilt)], color='green', linestyle='--', label='Cmd_Tilt')

    ax1.plot(Time[0:len(Time)], Perror_Tilt[0:len(Perror_Tilt)], color='purple', linestyle='-',  label='Error_Tilt')
    ax2.plot(Time[0:len(Time)], Verror_Tilt[0:len(Verror_Tilt)], color='purple', linestyle='--', label='Error_Tilt')

    ax1.plot(Time[0:len(Time)], OBJ_Pan[0:len(OBJ_Pan)], color='pink', linestyle='-',  label='OBJ_Pan')
    ax1.plot(Time[0:len(Time)], OBJ_Tilt[0:len(OBJ_Tilt)], color='orange', linestyle='--', label='OBJ_Tilt')

    ax1.plot(Time[0:len(Time)], PAct_Pan[0:len(PAct_Pan)], color='blue', linestyle='-',  label='Act_Pan')
    ax1.plot(Time[0:len(Time)], PCmd_Pan[0:len(PCmd_Pan)], color='blue', linestyle='--', label='Cmd_Pan')
    ax2.plot(Time[0:len(Time)], VAct_Pan[0:len(VAct_Pan)], color='blue', linestyle='-',  label='Act_Pan')
    ax2.plot(Time[0:len(Time)], VCmd_Pan[0:len(VCmd_Pan)], color='blue', linestyle='--', label='Cmd_Pan')

    ax1.plot(Time[0:len(Time)], Perror_Pan[0:len(Perror_Pan)], color='red', linestyle='-',  label='Error_Pan')
    ax2.plot(Time[0:len(Time)], Verror_Pan[0:len(Verror_Pan)], color='red', linestyle='--', label='Error_Pan')

    ax1.set_title('Step 5')
    ax1.set_ylabel('Position (rad)')
    ax2.set_ylabel('Velocity (rad/s)')
    ax2.set_xlabel('Time (s)')

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

    plt.show()


if __name__ =="__main__":
```

```
308         controller(None)
```

Step 1

(a) the option.

Option B.

(b) the expression for pan/tilt velocity for that option.

v_tilt= pi/6 * 2pi*4*cos(2pi*4*((t-t_0)/T_scan))

v_pan=(5pi/12*2pi)*cos(2pi*((t-t_0)/T_scan))

(c) the starting position and velocity (t=t_0) for both joints. Please express these in terms of A_pan, A_tilt, T_scan.

Pan:

Starting Position: $\theta\_tilt=A\_tilt \sin(2pi*4*((t-t\_0)/T\_scan))=A\_tilt*\sin(0)=0$

Starting Velocity: $d\ \theta\_tilt/dt=A\_tilt*2pi*4*\cos(2pi*4*((t-t\_0)/T\_scan))=(A\_tilt*2pi*4)/T\_scan$

Tilt:

Starting Position: $\theta\_pan=A\_pan*\sin(2pi*((t-t\_0)/T\_scan))=A\_pan*\sin(0)=0$

Starting Velocity: $d\ \theta\_pan/dt=A\_pan*2pi*\cos(2pi*((t-t\_0)/T\_scan))=(A\_pan*2pi)/T\_scan$