```
"""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)
            # Also look for eyes - only within the region of the face!
            # This similarly a list of eyes relative to this region.
            eyes = eyeDetector.detectMultiScale(gray[y:y+h,x:x+w])

            # Process the eyes: As before, eyes is a list of bounding
            # boxes (x,y,w,h) relative to the processed region.
            for (xe,ye,we,he) in eyes:
                # Can you draw circles around the eyes?  Consider the function:
                # cv2.circle(frame, (xc, yc), radius, (b,g,r), linewidth)
                pass # replace this.


        # Show the image with the given title.  Note this won’t actually
        # appear (draw on screen) until the waitKey(1) below.
        (H, W, D) = frame.shape
        #np.append(bgr, frame[W//2, H//2])
        #np.append(hsv, hsv[W//2, H//2])
        #print("Frame: ", frame[W//2, H//2])
        #print("HSV: ", hsv[W//2, H//2])

        #Convert to binary
        binary = cv2.inRange(hsv, (113, 126, 78), (151, 255, 225))
        #binary = cv2.inRange(hsv, (107, 142, 119), (119, 255, 255))

        #Erode, then Dialate
        binary=cv2.erode( binary, None, iterations=3)
        binary=cv2.dilate(binary, None, iterations=3)

        #Dilate, then Erode
        binary=cv2.dilate(binary, None, iterations=3)
        binary=cv2.erode(binary, None, iterations=3)

        #Contours
        (contours, hierarchy) = cv2.findContours(binary, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
        contoursTemp = sorted(contours, key=cv2.contourArea, reverse=True)
        contours=[]
        (xmax,ymax,rmax) = (0, 0, 0)
        balls = []
        for contour in contoursTemp:
            if cv2.contourArea(contour) > 2000:
                contours.append(contour)
                ((xr, yr), radius) = cv2.minEnclosingCircle(contour)
                cv2.circle(frame, (int(xr), int(yr)), 4, (91, 252, 104), -1)
                cv2.circle(frame, (int(xr), int(yr)), int(radius), (91, 252, 104), 3)
                x = scalepan * float(xr - 320)
                y = scaletilt * float(yr - 240)
                balls.append(Ball(x, y))
                #print(xr, yr)
        if shared.lock.acquire():
            """if rmax > 0:
       (x0, y0, x, y) = (320, 240, xmax, ymax)
       shared.deltapan = scalepan * float(x – x0)
       shared.deltatilt = scaletilt * float(y – y0)
       shared.newdata = True"""
            shared.balls = balls
            shared.lock.release()
            shared.scalepan = scalepan
            shared.scaletilt = scaletilt
            shared.newdata = True

        """ENCLOSING CIRCLE"""


        cv2.line(frame, (W//2,0), (W//2,H), (0,0,255), 1)
        cv2.line(frame, (0,H//2), (W, H//2), (0,0,255), 1)
        cv2.line(hsv, (W//2,0), (W//2,H), (0,0,255), 1)
        cv2.line(hsv, (0,H//2), (W, H//2), (0,0,255), 1)
        cv2.imshow(’Processed Image’, frame)
        #cv2.imshow(’HSV Funkiness’, hsv)
        cv2. imshow(’Binary Image’, binary)

        # Check for a key press IN THE IMAGE WINDOW: waitKey(0) blocks
        # indefinitely, waitkey(1) blocks for at most 1ms.  If ’q’ break.
        # This also flushes the windows and causes it to actually appear.
        if (cv2.waitKey(1) & 0xFF) == ord(’q’):
            break
        # Check if the main thread signals this loop to end.
        if shared.lock.acquire():
            stop = shared.stop
            shared.lock.release()
            if stop:
                break
```

```
    camera.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    controller()
```