```
"""goals6simple.py

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

"""

# Import OpenCV
import cv2
from math import pi

def detector(shared):

    scale_pan = 0.001413
    scale_tilt = 0.001472

    # 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 = 235
    wb = 3273
    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, 154)        # 0 - 255, default 128
    camera.set(cv2.CAP_PROP_CONTRAST,   128)        # 0 - 255, default 128
    camera.set(cv2.CAP_PROP_SATURATION, 210)        # 0 - 255, default 128


    # Keep scanning, until ’q’ hit IN IMAGE WINDOW.
    count = 0
    while True:

        if shared.lock.acquire():
            camerapan = shared.motorpan
            cameratilt = shared.motortilt
            shared.lock.release()

        # Grab an image from the camera.  Often called a frame (part of sequence).
        ret, frame = camera.read()
        count += 1

        # Grab and report the image shape.
        (H, W, D) = frame.shape
        #print(f"Frame #{count:3} is {W}x{H} pixels x{D} color channels.")

        # Convert the BGR image to RGB or HSV.
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)      # For other objects
        # hsv = cv2.cvtColor(frame, cv2.COLOR_RGB2HSV)      # For red objects

        # Print color of center pixel
        # print(’BGR at center pixel: ’, frame[W//2, H//2])
        # print(’HSV at center pixel: ’, hsv[W//2, H//2])

        # Cross hair on center pixel
        (xA1, yA1) = (W // 2, 0)
        (xA2, yA2) = (W // 2, H - 1)
        (xB1, yB1) = (0, H // 2)
        (xB2, yB2) = (W - 1, H // 2)
        cv2.line(frame, (xA1, yA1), (xA2,yA2), (0,0,255), 1)
        cv2.line(frame, (xB1, yB1), (xB2,yB2), (0,0,255), 1)
        # binary = cv2.inRange(hsv, (75, 115, 50), (115, 230, 190))
        binary = cv2.inRange(hsv, (75, 115, 50), (115, 230, 150))
        binary = cv2.erode(binary, None, iterations=3)
        binary = cv2.dilate(binary, None, iterations=1)

        # Add contours
        (contours, hierarchy) = cv2.findContours(binary, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
        contours = sorted(contours, key=cv2.contourArea)
       # cv2.drawContours(frame, contours, -1, (0,0,255), 2)


        CUTOFF = 800
        object_detected = None

        detectedobjects = []
        for contour in contours:
            # fit an ellipse to a single contour
            if (len(contour) >= 5):
                ((xe, ye), (w, h), angle) = cv2.fitEllipse(contour)
            if cv2.contourArea(contour) > CUTOFF and (0.7*h < w and w < 1.3*h):

                cv2.drawContours(frame, [contour], 0, (0,255,0), 2)
                object_detected = [xe, ye]
                # single contour centroid method
                # M = cv2.moments(contour)
                # area = M[’m00’]
                # x_c = int(M[’m10’] / M[’m00’])
                # y_c = int(M[’m01’] / M[’m00’])
                ellipse = cv2.fitEllipse(contour)
                cv2.ellipse(frame, ellipse, (0,255,255), 2)
                #print(f’({xe}, {ye})’)
                cv2.circle(frame, (int(xe), int(ye)), 4, (0, 255, 255), -1)

                # Calculate the pan and tilt for each object
                theta_pan = camerapan - scale_pan*(object_detected[0] - W//2)
                theta_tilt = cameratilt - scale_tilt*(object_detected[1] - H//2)
                if theta_tilt < pi/4:
                    detectedobjects.append((theta_pan, theta_tilt))

            else:
                cv2.drawContours(frame, [contour], 0, (0,0,255), 2)

        # Grab the actual motor angles showing where the camera is pointing.

        if len(detectedobjects) > 0:
            if shared.lock.acquire():
                shared.detectedobjs = detectedobjects.copy()
                shared.newdata = True
                shared.lock.release()
            #print(f’Camera pan/tilt: {camerapan}, {cameratilt}’)

        # Show the processed image with the given title.  Note this won’t
        # actually appear (draw on screen) until the waitKey(1) below.
        cv2.imshow(’Processed Image’, frame)
        #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
        if shared.lock.acquire():
            stop = shared.stop
            shared.lock.release()
            if stop:
                break

    # Close everything up.
    camera.release()
    cv2.destroyAllWindows()

if __name__ == ’__main__’:
    detector(None)
```