Object Detector and Performance Report

Maria Gonzalez and Kaliyah Solomon

Introduction

In this report, we describe the software for an object detecting robot. The robot has “pan” and “tilt” capabilities that are dependent on user keyboard input. With the goal of making an autonomous detector, we developed code enabling the robot to detect a green tennis ball based on its color properties. The system required that we set specific thresholds for (i) pixels with the ball’s HSV color, (ii) erode/dilate filtering restrictions, and (iii) contouring specifications. The implemented system is able to identify tennis balls as they enter the frame by identifying the HSV color range that differentiates the ball from other objects. We used a binary image to adjust for filtering and contour restrictions. We used a centroid approach to mark the center of the circle, which the user can see in the live image along with the contours and an outline of the ball. The final result of the detector could be further improved in cases where two balls touch, but the system successfully differentiates the green tennis balls from all other objects.

General Structure and Organization of the Code

1) **Capture an Image from the Camera:** Retrieve the image frame from the camera.
2) **Report the Image Shape:** Determine and report the captured image's height, width, and color channels.
3) **Convert the BGR Image to HSV:** Use *cv2.cvtColor()* with the *cv2.COLOR_BGR2HSV* parameter to convert the image from BGR color space to HSV color space.
4) **Print the Color of the Center Pixel:** Output the HSV values of the center pixel. This will help narrow down the appropriate HSV range for the target object.
5) **Create a Crosshair:** Utilize *cv2.line()* to draw two lines intersecting at the center pixel, indicating the pixel from which the HSV color is being printed.
6) **Binary Image Transformation Using HSV:** Apply *cv2.inRange()* to create a binary image that highlights only the objects within a specific HSV range (the range corresponding to the object of interest).
7) **Erode and Dilation Process:**
    a) **Erode-Dilate:** This step removes small or isolated pixels. We use a value of one, as higher values were found to eliminate significant portions of the object.
    b) **Dilate-Erode:** This step fills in small holes that may have been missed during the erosion. We used a value of seven, effectively filling in the writing and lines on a tennis ball.
8) **Contours:**
    a) **Find the Contours:** Utilize pre-defined *cv2* functions to detect contours within the binary image and sort them in reverse order.
    b) **For loop:** Iterate through each contour in the list of detected contours.
        i) **If Statement:** Check if the area of the current contour exceeds 1800 pixels.

(1) **Drawing the Contours:** Use the original frame to draw the current contour, selecting the BGR color and thickness.
(2) **Enclosing Circle:** Calculate the center coordinates (x, y) and the radius of the contour enclosing circle. Draw the enclosed circle on the frame by specifying the integer values for the coordinates and radius and selecting the BGR color and thickness.
(3) **Centroid:** Determine the center coordinates for the object, assuming the radius of the ball is 4. Draw this center point on the frame using the specified coordinates, radius, BGR color, and thickness.

9) **Print the Center Coordinates:** Output the calculated center coordinates of the circle to the console.
10) **Display the Image in Real Time:** Show the processed image on the screen in real time.

Detailed Description of the Code Sections.

Camera Settings

We placed the tennis ball in front of the camera and set it to automatic mode to determine the optimal calibration settings for capturing it. Once we identified these values, we switched the camera to manual mode to prevent further changes. The camera selected the following settings:

- Exposure: 330
- White Balance: 1311
- Focus: 0
- Brightness: 128
- Contrast: 128
- Saturation: 190

These settings resulted in a clear and sharp image of the tennis ball, avoiding the washed-out effect that occurred with lower saturation levels. The high saturation helped maintain the ball's vibrancy, while the selected contrast prevented it from appearing dull; hence, the contrast was set to 190 instead of 128. This combination ensured that the image was both clear and visually appealing.

Color Isolation

During the initial round of HSV regular testing, we recorded the following ranges: without using the slider, the values were 15-44 for H (Hue), 65-151 for S (Saturation), and 121-229 for V (Value). When we used the sliders, the ranges changed to 7-50 for H, 61-173 for S, and 174-255 for V. We still detect objects that mix neon yellow and green. If we had made different choices, we might have detected other false objects in bright light greens and yellows, but we wanted to keep our guesses within a reasonable color range that matched the ball. We

could limit the detector to hair more accurately by narrowing the range for H and observing how that adjustment affects the detection code. This might help ensure we only detect objects matching the chosen ball color.

Pixel-Level Filtering.

To further separate the green tennis ball from other objects in frame, we used erode/dilate features. To remove isolated pixels, we used the erode-then-dilate method. When doing this, we tested iterations going from 1-7 and found that using 1 iteration for both erode and dilate was the best for isolating the ball in the binary image while maintaining its shape, as seen in Figure 1.

Figure 1. Erode-then-dilate binary image with iteration of 7 on the left and iteration of 1 on the right.

We then filled in holes using the dilate-then-erode method. Doing this, we found that using iterations of 7 for both erode and dilate was best suited for detecting the ball because lesser iterations identified areas outside of the ball's radius that we did not want in the final image detection.

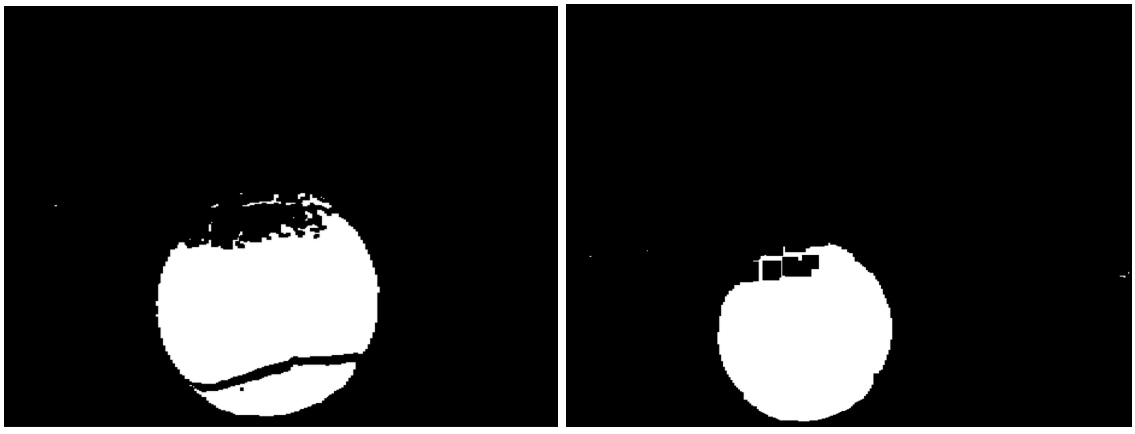

Figure 2. Dilate-then-erode binary image with iteration of 1 on the left and iteration of 7 on the right.

For the best results, we combined the erode-then-dilate and dilate-then-erode methods into our code.

Object-Level Down-Selection and Centroid.

We decided to limit the “valid” contours by checking that the contour area is greater than 1800. We decided on this value by testing out different values and seeing how far away from the camera the ball will still be detected while the contours maintain accuracy. To process the contours into objects, we used a centroid approach.

User Display

In addition to detecting object locations, we presented information to help the user understand what was happening. We printed the coordinates of the center of the enclosing circle and the centroid of the circle we found. This information helps the user understand the position of the ball's center in the image frame.

We also drew a circle around the ball based on the enclosing circle and the centroid we identified. Additionally, we displayed the contour that exceeded our cutoff point, allowing us to verify whether it accurately covered the entire ball. The user may also be interested in the ball's position in radians from the camera's perspective.

In the photo below, the pink circle represents the enclosing circle, the green point indicates the centroid and the light blue line depicts the contour we used to obtain the coordinates for both the enclosing circle and the centroid.

Figure 3. Final user display with enclosing circle, contours, and centroid.

Performance, Limitations, and Possible Improvements.

The system was optimized to differentiate between the green tennis balls and the image background. While our efforts created a working detection algorithm, improvements can be made when it comes to having multiple green tennis balls in frame. When two tennis balls touch each other, the camera detects one larger tennis ball, as seen in Figure 4. Another issue arose when we partially covered a tennis ball. In this case, the detector “split” the tennis ball into two. Both issues can be solved in the future by setting restrictions based on the diameter of one tennis ball.

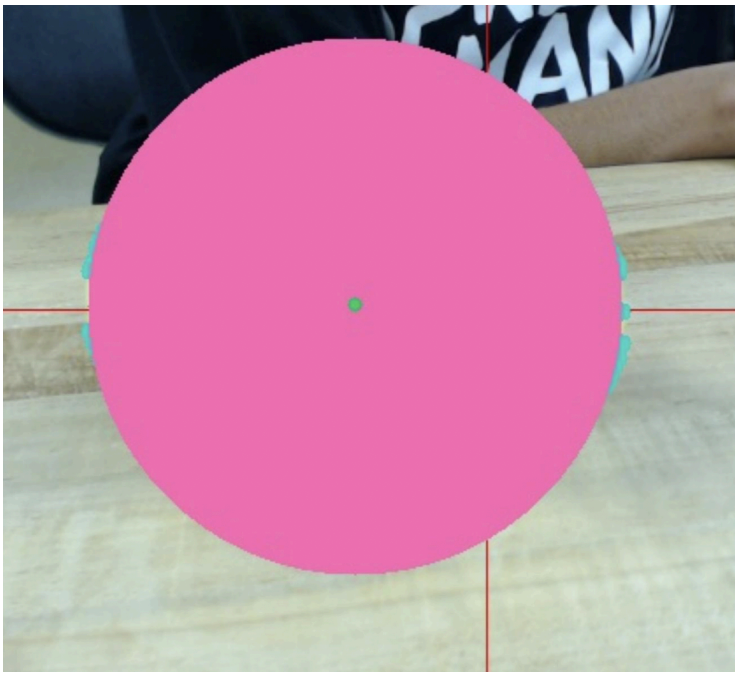

Figure 4. Detection of one large tennis ball when two tennis balls touch.

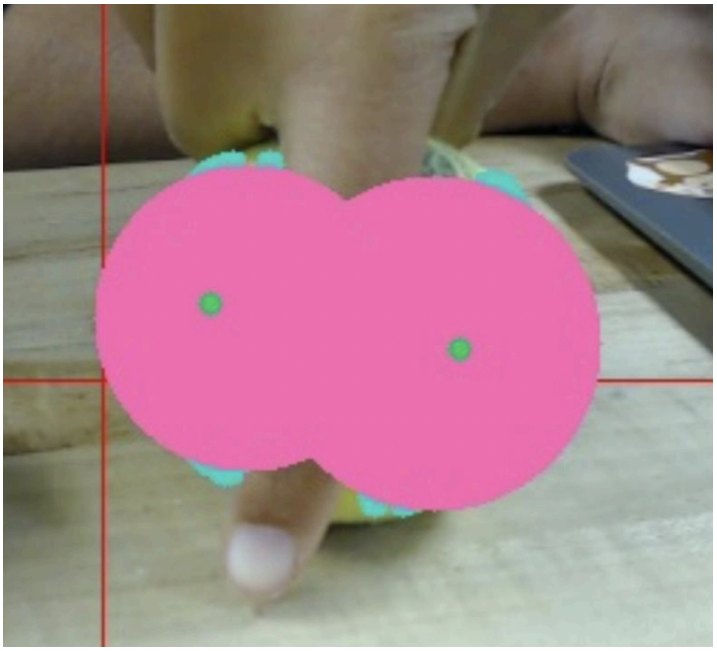

Figure 5. Detection of two tennis balls when a singular tennis ball is covered by a finger.

Furthermore, the detector's contouring limits the distance that the tennis ball can be from the camera in order to be properly detected. In the future, we may want to increase this limit in exchange for less accurate contouring.

Code Appendix

```python
"""goals6simple.py

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

"""

# Import OpenCV
import cv2

# 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 = 330
wb = 1311
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()
    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
    bgr = frame[W//2,H//2]
    hsv2 = hsv[W//2,H//2]
    #print(f"BGR = {bgr}")
    #print(f"HSV = {hsv2}")

    # Cross hair on center pixel
    cv2.line(frame, (0, H//2), (W-1, H//2), (0,0,255), 1)
    cv2.line(frame, (W//2, 0), (W//2, H-1), (0,0,255), 1)
    binary = cv2.inRange(hsv, (7, 61, 174), (50, 173, 255))
    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)
    xc=0
```

```
81      yc=0
82      #down-select contours
83      for contour in contours:
84          if cv2.contourArea(contour) > 1800:
85              cv2.drawContours(frame, contour, -1, (200, 213, 48), 10)
86
87
88              M=cv2.moments(contour)
89              area=M['m00']
90              xc = int(M['m10']/M['m00'])
91              yc = int(M['m01']/M['m00'])
92
93              ((xr,yr), radius) = cv2.minEnclosingCircle(contour)
94              cv2.circle(frame, (int(xr),int(yr)), int(radius), (180, 105, 255), -1)
95              cv2.circle(frame, (xc,yc), 4, (100, 205, 25), -1)
96
97
98      print(f"(x, y) for the centroid is ({xc},{yc})")
99      print(f"(x, y) for the Enclosing circle is ({int(xr)},{int(yr)})")
100
101     # Show the processed image with the given title.  Note this won't
102     # actually appear (draw on screen) until the waitKey(1) below.
103     cv2.imshow('Processed Image', frame)
104     cv2.imshow('Binary Image', binary)
105
106     # Check for a key press IN THE IMAGE WINDOW: waitKey(0) blocks
107     # indefinitely, waitkey(1) blocks for at most 1ms.  If 'q' break.
108     # This also flushes the windows and causes it to actually appear.
109     if (cv2.waitKey(1) & 0xFF) == ord('q'):
110         break
111
112 # Close everything up.
113 camera.release()
114 cv2.destroyAllWindows()
```