Color Detection Using Raspberry Pi Camera

Objective

The purpose of this project is to detect specific colors — **Red, Blue, and Green** — using a Raspberry Pi camera, with proper HSV tuning to ensure accurate detection. The system uses contour detection to draw rectangles and label the detected colors, while ignoring small noise.

Hardware & Software Setup

- **Hardware:**
  - Raspberry Pi (any model with CSI camera support)
  - 5-megapixel Pi camera module (OmniVision OV5647, fixed-focus lens)
  - Standard lighting environment
- **Software:**
  - Python 3
  - `picamera2` library for camera interfacing
  - `OpenCV` for image processing
  - NumPy for array operations

Camera Configuration:

- Resolution: 640x480
- Format: RGB888

- Auto Exposure: Enabled for proper visibility
- Auto White Balance: Disabled for consistent color detection

---

Methodology

1. Initialize the Raspberry Pi camera using Picamera2.
2. Capture RGB frames and convert them to **HSV** color space.
3. Apply **HSV masks** for Red, Blue, and Green to isolate the colors.
4. Use **contour detection** to find objects above a minimum size threshold.
5. Draw **rectangles** around detected objects and label them with their respective color.
6. Display the **original and overlay frames side by side**.

HSV Ranges for Color Detection

Color	Hue (H)	Saturation (S)	Value (V)
Red	80-179	100–255	20–255
Blue	0–179	255	67–255
Green	40–69	120–255	134–255

Note: Red uses two ranges for Hue because it wraps around the HSV color wheel.

Code for Testing HSV Ranges (Trackbars)

This code allows testing and tuning of HSV ranges in real-time:

```
import cv2
import numpy as np

def nothing(x):
    pass

# Create resizable window for trackbars
cv2.namedWindow("Trackbars", cv2.WINDOW_NORMAL)

cv2.createTrackbar("H Lower", "Trackbars", 0, 179, nothing)
cv2.createTrackbar("H Upper", "Trackbars", 179, 179, nothing)
cv2.createTrackbar("S Lower", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("S Upper", "Trackbars", 255, 255, nothing)
cv2.createTrackbar("V Lower", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("V Upper", "Trackbars", 255, 255, nothing)

# Dummy loop just to see the sliders
while True:
    h_l = cv2.getTrackbarPos("H Lower", "Trackbars")
    h_u = cv2.getTrackbarPos("H Upper", "Trackbars")
    s_l = cv2.getTrackbarPos("S Lower", "Trackbars")
    s_u = cv2.getTrackbarPos("S Upper", "Trackbars")
    v_l = cv2.getTrackbarPos("V Lower", "Trackbars")
    v_u = cv2.getTrackbarPos("V Upper", "Trackbars")

    print(f"H: {h_l}-{h_u}, S: {s_l}-{s_u}, V: {v_l}-{v_u}", end="\r")

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cv2.destroyAllWindows()
```

This allows dynamic adjustment of HSV thresholds before applying them in the detection code.

Final Color Detection Code

from picamera2 import Picamera2

```
import cv2
import numpy as np

# Initialize camera
picam2 = Picamera2()
camera_config = picam2.create_preview_configuration(main={"format": "RGB888", "size": (640,
480)})
picam2.configure(camera_config)

# Controls: AE on, AWB off
picam2.set_controls({
    "AeEnable": True,
    "AwbEnable": False,
})
picam2.start()

# HSV ranges for your colors
red_lower = np.array([80, 100, 20])
red_upper = np.array([179, 255, 255])

blue_lower = np.array([0, 255, 67])
blue_upper = np.array([179, 255, 255])

green_lower = np.array([40, 120, 134])
green_upper = np.array([69, 255, 255])

MIN_AREA = 500  # Minimum contour size in pixels

try:
    while True:
        frame = picam2.capture_array()  # RGB frame
        hsv = cv2.cvtColor(frame, cv2.COLOR_RGB2HSV)

        # Create masks
        mask_red = cv2.inRange(hsv, red_lower, red_upper)
        mask_blue = cv2.inRange(hsv, blue_lower, blue_upper)
        mask_green = cv2.inRange(hsv, green_lower, green_upper)

        overlay = frame.copy()

        # Red contours
        contours, _ = cv2.findContours(mask_red, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        for cnt in contours:
```

```
            if cv2.contourArea(cnt) > MIN_AREA:
                x, y, w, h = cv2.boundingRect(cnt)
                cv2.rectangle(overlay, (x, y), (x + w, y + h), (0, 0, 255), 2)  # Red rectangle
                cv2.putText(overlay, "Red", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0,
255), 2)

        # Blue contours
        contours, _ = cv2.findContours(mask_blue, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        for cnt in contours:
            if cv2.contourArea(cnt) > MIN_AREA:
                x, y, w, h = cv2.boundingRect(cnt)
                cv2.rectangle(overlay, (x, y), (x + w, y + h), (255, 0, 0), 2)  # Blue rectangle
                cv2.putText(overlay, "Blue", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0,
0), 2)

        # Green contours
        contours, _ = cv2.findContours(mask_green, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        for cnt in contours:
            if cv2.contourArea(cnt) > MIN_AREA:
                x, y, w, h = cv2.boundingRect(cnt)
                cv2.rectangle(overlay, (x, y), (x + w, y + h), (0, 255, 0), 2)  # Green rectangle
                cv2.putText(overlay, "Green", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0,
255, 0), 2)

        # Show original and overlay side by side
        combined = np.hstack((frame, overlay))
        cv2.imshow("Color Detection", combined)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

finally:
    picam2.stop()
    cv2.destroyAllWindows()
```

Conclusion

The Raspberry Pi camera, combined with HSV thresholding and contour detection, can successfully detect **Red, Blue, and Green** objects. Using tuned HSV ranges and disabling automatic white balance ensures that colors are detected accurately. Minimum contour filtering removes noise, and rectangles with labels provide a clear visualization of detected objects. This system can be adapted for real-time applications, robotics, and color-based tracking.

I can also **add a diagram showing the detection pipeline** (camera → HSV → mask → contours → overlay) if you want, to make the report visually complete.