Geometric Shape and Color Detection using OpenCV

1 Introduction

This document describes a classical method for detecting and classifying four geometric shapes—Rectangle, Square, Circle, and Triangle—along with identifying their colors. The approach employs traditional computer vision techniques such as edge detection, contour approximation, and color space conversion using OpenCV.

2 Shape Detection

The shape detection algorithm consists of the following steps:

1. Convert the input image to grayscale using the OpenCV function `cv2.cvtColor()`.
2. Apply Gaussian Blur to reduce noise in the image.
3. Use Canny Edge Detection to highlight the edges of the shapes.
4. Detect contours using `cv2.findContours()`.
5. For each contour:
    - Approximate the contour using the Douglas-Peucker algorithm (`cv2.approxPolyDP()`) to reduce the number of vertices.
    - Based on the number of vertices:
        - **Triangle**: If the approximated contour has 3 vertices.
        - **Square or Rectangle**: If the contour has 4 vertices. The aspect ratio is used to differentiate between the two. For squares, the aspect ratio is close to 1, i.e., $0.95 \leq \text{aspect ratio} \leq 1.05$.
        - **Circle**: If the contour has more than 4 vertices.

The geometric properties used for shape classification are primarily based on the number of contour vertices and aspect ratios.

3 Color Detection

For color identification, the following steps are performed:

1. Convert the input image to HSV (Hue, Saturation, Value) color space using `cv2.cvtColor()`.
2. Define color ranges for Red, Blue, Green, and Yellow using lower and upper bounds in the HSV space.
3. For each detected shape, calculate the central point using the `medina` function, which computes the average of the shape's vertices.
4. Check the HSV value at the central point and determine the color by comparing it against predefined color bounds using the function `check_boundry`.

The color ranges used are as follows:

- **Red**: HSV range $[0, 100, 50]$ to $[10, 255, 255]$
- **Blue**: HSV range $[100, 100, 100]$ to $[140, 255, 255]$
- **Green**: HSV range $[44, 150, 50]$ to $[65, 255, 255]$
- **Yellow**: HSV range $[20, 150, 50]$ to $[30, 255, 255]$

4 Implementation

The program is implemented in Python using the OpenCV library. The code processes a given input image and detects shapes and their colors as described above. The image is expected to contain animated-style geometric shapes, and the code does not account for real-world photographs.

4.1 Main Functions

- **check_boundry(img, range, lower_bound, upper_bound)**: Checks if the pixel value in the HSV space falls within the defined color range.
- **medina(points)**: Calculates the midpoint of a shape by averaging the vertex coordinates.
- **color_detect(img, range)**: Identifies the color of the shape based on the pixel at the midpoint.

5 Challenges and Insights

The primary challenge in this task was detecting shapes and identifying colors using classical methods without relying on modern deep learning techniques. This required a deep understanding of geometric properties and HSV color space. Classical methods like edge detection and contour approximation proved to be effective for animated-style images where shapes have clear boundaries and uniform colors.

6 Conclusion

This project demonstrates how classical computer vision techniques can be applied to detect and classify geometric shapes, as well as identify their colors. By combining edge detection, contour analysis, and color thresholding, the program successfully detects shapes such as triangles, squares, rectangles, and circles, and classifies their colors as red, blue, green, or yellow.