LOYOLA INSTITUTE OF TECHNOLOGY AND SCIENCE

NM1093-OPEN SOURCE COMPUTER VISION LIBRARY

REAL-TIME TRAFFIC SIGN DETECTION

Submited by

L.MANCY-961222104034

REAL-TIME TRAFFIC SIGN DETECTION

Introduction

This project demonstrates a basic computer vision system using OpenCV that detects the color of a traffic light (Red, Yellow, Green) in a static image. Based on the detected color, it identifies and displays a corresponding action, such as “STOP,” “WAIT,” or “START.” This approach forms a foundational step toward more advanced real-time traffic monitoring or autonomous driving systems.

Objective

To develop a Python program that:
Detects traffic light colors in a static image.
Identifies which light (Red, Yellow, or Green) is active.
Maps detected colors to specific actions.
Visually annotates and outputs the result.

Tools and Technologies

Language: Python
Library: OpenCV (cv2), NumPy
Image Format: PNG (e.g., traffic.png)
4. Methodology
4.1. Image Preprocessing
The input image is loaded using cv2.imread().
It’s resized for easier visualization.
Gaussian Blur is applied to reduce noise.
The image is converted to the HSV color space for better color segmentation.

Color Detection

HSV thresholds are defined for three primary traffic light colors:
Red: [0, 100, 100] to [10, 255, 255]
Yellow: [20, 100, 100] to [30, 255, 255]

Green: [40, 50, 50] to [90, 255, 255]
Masks are created using cv2.inRange() to isolate regions matching each color.

Contour Detection and Filtering

cv2.findContours() is used to locate color regions in the masks.
Each contour's area is calculated; only those above a threshold (500 pixels) are considered significant.
A bounding box is drawn around the detected area, and the corresponding action is labeled on the image.

Output

The processed image with annotations is displayed using cv2.imshow().
Console output prints detected colors and their corresponding actions.

Results

When executed on an image containing a traffic light:
The system accurately identifies the active light color.
Displays bounding rectangles around detected lights.
Annotates the image with color and action labels like:
vbnet
Copy
Edit
Red Light Detected → Action: STOP

Limitations

Only works on static images, not live video.
HSV range sensitivity may vary with lighting conditions or image quality.
Red color detection could be improved by handling both hue ranges (around 0 and 180).
May misidentify regions with similar colors (e.g., billboards or vehicles with red/yellow/green).

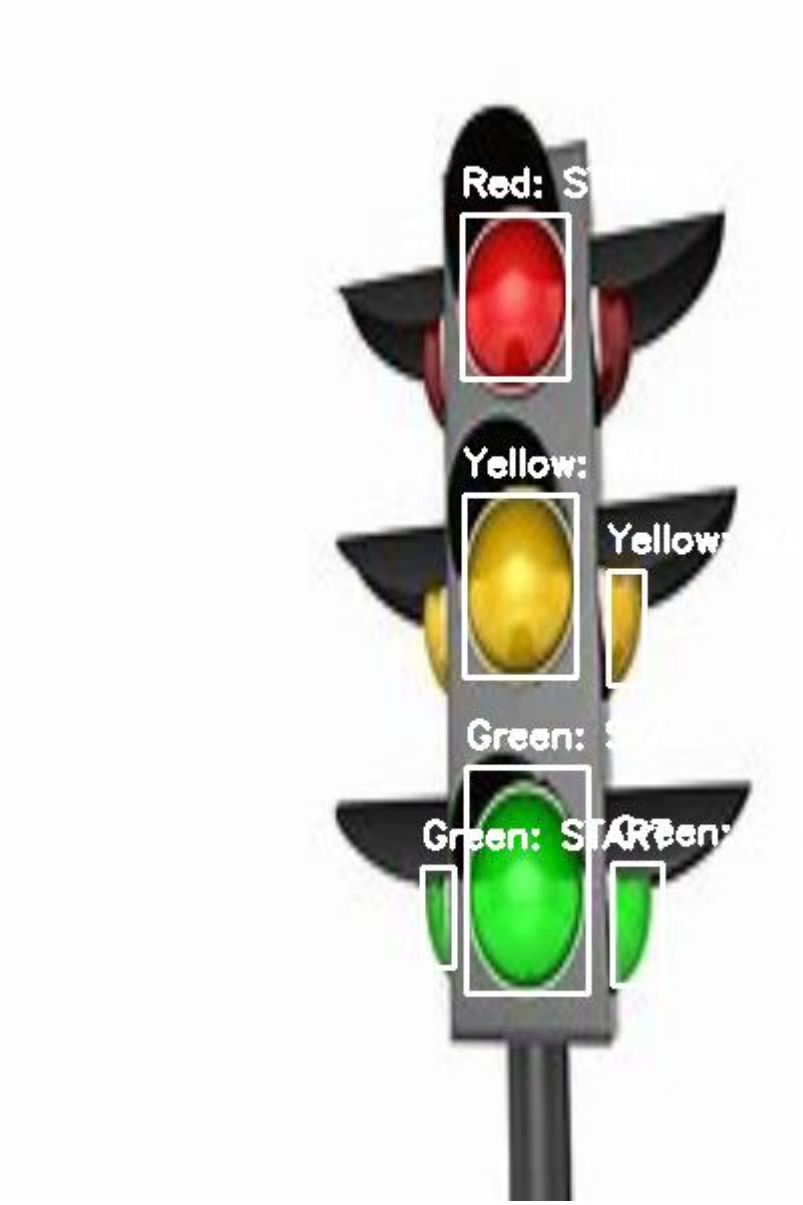
Future Improvements

Extend to real-time video using a webcam feed.
Improve robustness using morphological operations.
Add machine learning or deep learning for shape detection (e.g., detecting actual traffic light shape).
Integrate sound alerts for each action (e.g., beeping on RED).

Deploy on embedded systems like Raspberry Pi for mobile use.

```
# # Real time Traffic sign detection using image
import cv2
import numpy as np

# Load the input image
image = cv2.imread('traffic.png')
if image is None:
    print("Image not found!")
    exit()

# Resize for easier visualization (optional)

image = cv2.resize(image, (400, 600))

# Blur and convert to HSV
blurred = cv2.GaussianBlur(image, (5, 5), 0)
hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)

# Define HSV color ranges
color_ranges = {
    'Red': ([0, 100, 100], [10, 255, 255]),
    'Yellow': ([20, 100, 100], [30, 255, 255]),
    'Green': ([40, 50, 50], [90, 255, 255])
}

# Action labels
actions = {
    'Red': 'STOP',
    'Yellow': 'WAIT',
    'Green': 'START'
}

for color, (lower, upper) in color_ranges.items():
    lower = np.array(lower)
    upper = np.array(upper)
    mask = cv2.inRange(hsv, lower, upper)

    contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    for cnt in contours:
        area = cv2.contourArea(cnt)
        if area > 500:  # filter small noise
            x, y, w, h = cv2.boundingRect(cnt)
            cv2.rectangle(image, (x, y), (x+w, y+h), (255, 255, 255), 2)
            cv2.putText(image, f"{color}: {actions[color]}", (x, y - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)
            print(f"{color} Light Detected → Action: {actions[color]}")

cv2.imshow("Traffic Light Detection", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:



Conclusion

This project illustrates how basic image processing and color segmentation in OpenCV can be effectively used to simulate a traffic light detection system. While currently limited to static images, the approach serves as a stepping stone toward real-time, intelligent traffic systems in autonomous vehicles.