HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

--------o0o--------

BÁO CÁO BÀI TẬP LỚN

CHỦ ĐỀ: Traffic violation detection and management

Giảng viên bộ môn: Phạm Văn Sự

Mã nhóm: DIPFL23PRJG41

Mã đề: STC18

Nhóm tín chỉ: 9

Sinh viên thực hiện	Mã sinh viên
Nguyễn Phúc Sơn	B20DCCN581
Nguyễn Văn Cường	B20DCCN103
Nguyễn Văn Nghiêm	B19DCCN470

Hà Nội – 2023

Danh mục

A. Đánh giá đóng góp các thành viên

Sinh viên	MSV	Phần việc	Đánh giá
Nguyễn Phúc Sơn	B20DCCN581	- Tổng hợp nội dung, viết báo cáo - Tìm kiếm, cài đặt và chạy source code - Phân tích chi tiết chức năng: + nhận diện đèn giao thông + nhận diện vạch kẻ giao thông + cập nhật biển phạt & hàm main Nội dung làm: 3.1, 3.2, 3.5, 3.6, 4	42.5%
Nguyễn Văn Cường	B20DCCN103	- Mô tả bài toán, giới thiệt thư viện - Tham gia tìm kiếm source code - Phân tích chi tiết chức năng nhận diện biển số xe & nhận dạng văn bản - Nội dung làm: 1, 2, 3.3, 3.4	42.5%
Nguyễn Văn Nghiêm	B19DCCN470	- Mô tả sơ lược nhận diện vạch kẻ - Nội dung làm: 3.4	15%

B. Mô tả bài tập lớn

1. Nêu vấn đề

Hiện nay tại các nút giao thông trọng yếu tại các đô thị lớn thường có mật độ phương tiện giao thông rất là lớn từ đó gây ra tình trạng hỗn loạn, khó kiểm soát tới từ các cơ quan giám sát, điều khiển giao thông, người tham gia giao thông có thể lợi dụng các tình huống khó kiểm soát như vậy để thực hiện những hành vi phạm luật giao thông điển hình nhất là vượt đèn đỏ từ đó có thể gây ra những rủi ro lớn về vấn đề tài nạn giao thông gây thiệt hại về người và tài sản. Từ đó nhu cầu cấp thiết về một hệ thống tự động phát hiện vi phạm vượt đèn đỏ của người tham giao thông thông qua các cảnh quay theo thời gian thực được thực hiện để có thể đảm bảo xác định và xử phạt những người vi phạm.

2. Công nghệ được sử dụng

2.1 Thư viện openCV

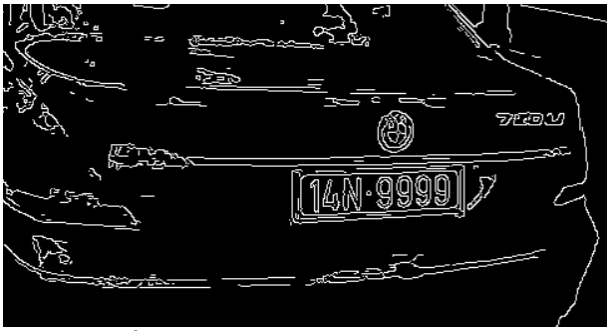
OpenCV là thư viện mã nguồn mở cho thị giác máy tính, xử lý ảnh và học máy, các tính năng tăng tốc GPU trong hoạt động thời gian thực. OpenCV được phát hành theo giấy phép BSD, do đó nó hoàn toàn phí cho cả học thuật và thương mại, cũng như được viết bằng C/C++ nên thư viện có thể tận dụng lợi thể xử lý đa lõi vậy nên rất phù hợp cho dự án này của nhóm trong việc nghiên cứu và phát triển khi ngồi trên ghế nhà trường.

Thư viện này sẽ được sử dụng cho việc xử lý video, nhận diện đèn giao thông và theo đó là xử lý biển số xe.

OpenCV có thể dễ dàng đọc các file video để xử lý khung hình với hàm cv2.VideoCapture(), và hiển thị video với hàm cv2.imshow() để xử lý trên giao diện video.

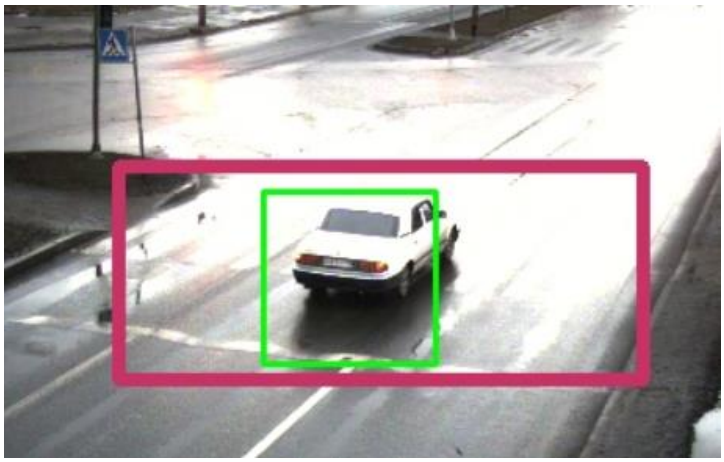
OpenCV có thể xử lý phát hiện các cạnh biên của ảnh:

Ảnh gốc



Ảnh sau khi phát hiện cạnh

Ngoài ra OpenCV còn có thể thao tác trên Video để từ đó phục vụ cho việc kẻ một đường thẳng và giúp phát hiện xe vượt qua đó:



Tuỳ chỉnh trên một frame với opencv

2.2 Thư viện Numpy

Thư viện này có thể thực hiện các kỹ thuật ảnh đơn giản, như lật ảnh, trích xuất các tính năng và phân tích chúng.

Với ảnh màu là một mảng 3 chiều, bằng cách cắt mảng đa chiều, các kênh RGB có thể được tách ra.

Dưới đây là một số thao tác có thể được thực hiện bằng NumPy trên hình ảnh (hình ảnh được tải trong một biến có tên test_img bằng cách sử dụng imread).

- Để lật hình ảnh theo hướng dọc, hãy sử dụng np.flipud (test_img).

- Để lật hình ảnh theo hướng ngang, hãy sử dụng np.fliplr (test_img).
- Để đảo ngược hình ảnh, hãy sử dụng test_img [:: – 1] (hình ảnh sau khi lưu trữ dưới dạng mảng numpy được đặt tên là <img_name>).
- Và nhiều ứng dụng khác nữa

Ngoài ra Numpy cũng được sử dụng để làm việc với mảng số học trong ngữ cảnh của OpenCV, ví dụ mỗi hình ảnh được biểu diễn dưới dạng Numpy và các phép toán số học sẽ được thực hiện trên mảng này

Với những tiện ích như vậy thư viện Numpy sẽ giúp tối ưu hóa hiệu suất và cung cấp các công cụ hiệu quả để xử lý các phép toán số học.

2.3 Thư viện PIL

PIL là viết tắt của Python Image Library. Đây là một trong những thư viện mạnh mẽ, hỗ trợ một loạt các định dạng hình ảnh như PPM, JPEG, TIFF, GIF, PNG và BMP.

Trong quá trình xử lý hình ảnh và nhận diện biển số xe, nhóm đã tích hợp thư viện PIL để thực hiện các thao tác chuyển đổi và tương tác với hình ảnh. PIL cung cấp một loạt các tính năng mãnh mẽ, đặc biệt là thao tác với đối tượng ảnh.

Một trong những tính năng chính của PIL là khả năng chuyển đổi giữa các định dạng hình ảnh khác nhau và tương tác linh hoạt với các đối tượng ảnh. Ta có thể chuyển đổi mảng Numpy từ OpenCV thành đối tượng hình ảnh PIL qua hàm Image.formarray()

Như vậy, nhóm có thể dễ dàng tích hợp Pillow vào quy trình xử lý hình ảnh và tận dụng các tính năng mạnh mẽ mà thư viện này mang lại. Thư viện này không chỉ giúp chúng em chuyển đổi đối tượng hình ảnh giữa các nguồn khác nhau mà còn hỗ trợ trong quá trình hiển thị và tương tác với hình ảnh, làm cho nó trở thành một công cụ quan trọng trong dự án.

2.4 Thư viện matplotlib

Thư viện Matplotlib là một thư viện mạnh mẽ cho việc tạo đồ thị và biểu đồ trong ngôn ngữ lập trình Python. Trong dự án của nhóm Matplotlib được sử dụng để hiển thị ảnh của biển số xe khi việc nhận diện và xử lý vi phạm được thực hiện.

Một Matplotlib figure có thể được phân loại thành nhiều phần như dưới đây:

- **Figure:** Như một cái cửa sổ chứa tất cả những gì bạn sẽ vẽ trên đó.
- **Axes:** Thành phần chính của một figure là các axes (những khung nhỏ hơn để vẽ hình lên đó). Một figure có thể chứa một hoặc nhiều axes. Nói cách khác, figure chỉ là khung chứa, chính các axes mới thật sự là nơi các hình vẽ được vẽ lên.

- **Axis:** Chúng là dòng số giống như các đối tượng và đảm nhiệm việc tạo các giới hạn biểu đồ.
- **Artist:** Mọi thứ mà bạn có thể nhìn thấy trên figure là một artist như Text objects, Line2D objects, collection objects. Hầu hết các Artists được gắn với Axes.

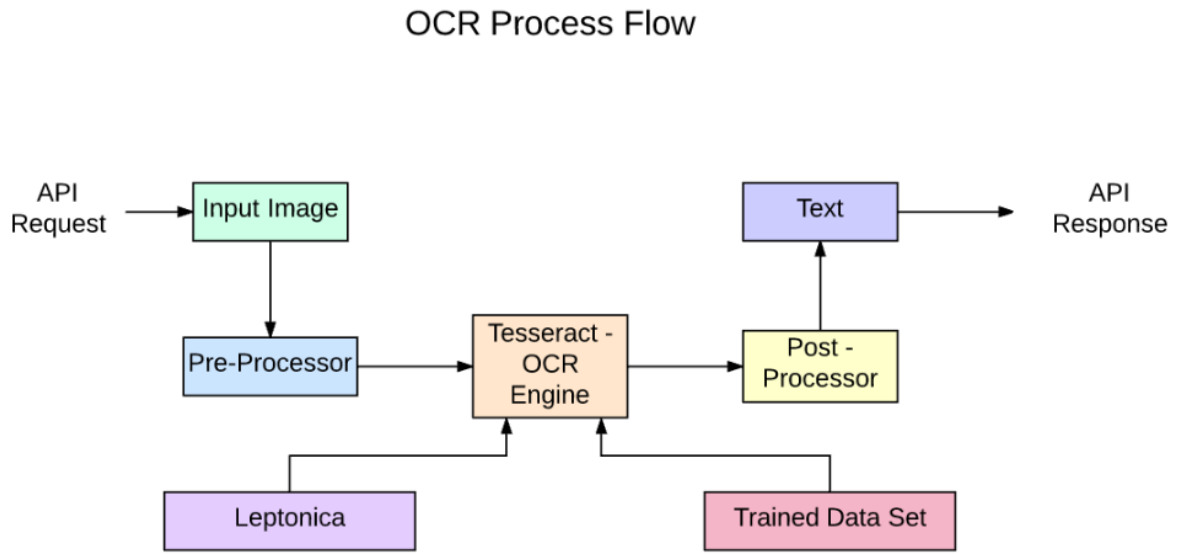
Matplotlib có thể giúp hiển thị biển số xe phạm lỗi, giúp kiểm tra trực quan chất lượng của việc nhận diện biển số xe và xác minh kết quả.

2.5 Thư viện pytesseract

Ở thư viện pytesseract chúng ta chỉ tập trung vào mô hình ORC (Optical Character Recognition) một hệ thống chuyển đổi hình ảnh văn bản hai chiều , có thể chứa văn bản được in bằng máy hoặc viết tay từ biểu diễn hình ảnh của nó thành văn bản có thể đọc được bằng máy. OCR là một quy trình thường bao gồm một số quy trình phụ để thực hiện chính xác nhất có thể. Các quy trình con là:

- Tiền xử lý hình ảnh
- Bản địa hóa văn bản
- Phân đoạn ký tự
- Nhận dạng ký tự
- Xử lý bài đăng

Tất nhiên, các quy trình phụ trong danh sách trên có thể khác nhau, nhưng đây là những bước cơ bản cần thiết để tiếp cận tính năng nhận dạng ký tự tự động. Trong phần mềm OCR, mục đích chính là xác định và nắm bắt tất cả các từ duy nhất sử dụng các ngôn ngữ khác nhau từ các ký tự văn bản viết.



Để nhận dạng hình ảnh chứa một ký tự đơn, ORC thường sử dụng Mạng thần kinh chuyển đổi (CNN). Văn bản có độ dài tùy ý là một chuỗi ký tự và những vấn đề như vậy được giải quyết bằng RNN và LSTM là một dạng RNN phổ biến.

2.6 Thư viện re

Thư viện re trong Python (Regular Expressions) được sử dụng để thực hiện các phép toán xử lý chuỗi dựa trên các biểu thức chính quy (regular expressions). Trong dự án, đã sử dụng re để kiểm tra xem văn bản từ biển số xe có khớp với một mẫu cụ thể hay không.

Ví dụ ta có thể kiểm tra biển số xe có đúng mẫu:

```
if text is not None \
    and re.match("^[A-Z]{2}\s[0-9]{3,4}$", text): ...
```

Thư viện re giúp bạn thực hiện các phép toán kiểm tra, trích xuất, hoặc thay thế chuỗi dựa trên các mẫu chính quy, là một công cụ hữu ích khi làm việc với dữ liệu chuỗi có cấu trúc nhất định.

2.7 Thư viện collection

Thư viện collection là một bộ chứa được sử dụng để lưu trữ các bộ sưu tập dữ liệu, ví dụ: list, dict, set và tuple, ... Được giới thiệu để cải thiện các chức năng của bộ chứa bộ sưu tập tích hợp.

- Hàm python namedtuple() trả về một đối tượng giống như tuple với tên cho từng vị trí trong bộ dữ liệu. Nó được sử dụng để loại bỏ vấn đề ghi nhớ chỉ số của từng trường của một đối tượng bộ dữ liệu trong các bộ dữ liệu thông thường.

- Python OrderedDict() tương tự như một đối tượng Dictionary trong đó các khóa duy trì thứ tự chèn. Nếu chúng ta cố gắng chèn khóa một lần nữa, giá trị trước đó sẽ bị ghi đè cho khóa đó.

- Python defaultdict() được định nghĩa là một đối tượng giống như dictionary. Nó là một lớp con của lớp dict. Nó cung cấp tất cả các phương thức được cung cấp bởi dictionary nhưng lấy đối số đầu tiên làm kiểu dữ liệu mặc định.

- Python Count() là một lớp con của đối tượng từ điển giúp đếm các đối tượng hashtable.

- Python deque() là hàng đợi hai đầu cho phép chúng ta thêm và xóa các phần tử ở cả hai đầu.

3. Mô hình hệ thống

3.1 Nhận diện biển báo giao thông và trạng thái đèn

3.1.1. Giới thiệu

Trong phần demo của nhóm, biển báo giao thông sẽ được chỉnh sửa trong video và được cố định tại vị trí góc trên cùng bên phải

Mục đích là để nhận diện trạng thái đèn thời gian thực, từ đó xét đến việc xác định các xe đi qua vạch kẻ đường và thu thập biển số của các xe đó sau này. Dưới đây là các bước thực hiện:

- Region of Interest – vùng quan tâm (ROI): ta bắt đầu bằng cách xác định vùng quan tâm trong hình ảnh. Đây là nơi ta mong đợi đèn giao thông sẽ xuất hiện và tách phần này khỏi phần còn lại để tập trung phân tích vào phần đó.
- Chuyển đổi HSV: ROI sau đó được chuyển đổi từ không gian màu RGB sang HSV (Hue, Saturation, Value)

Việc chuyển đổi sang HSV sẽ tách nội dung màu sắc (màu sắc) khỏi nội dung độ chói *Saturation* (khả năng kết hợp màu gốc với màu trắng) và độ sáng *value* (khả năng sinh ra màu sáng hoặc tối hơn). Từ đó việc đọc thông tin của màu của ảnh trở nên dễ dàng hơn, thay vì sự kết hợp của 3 màu R, G, B.

Các giá trị màu cơ bản của thành phần *hue:*

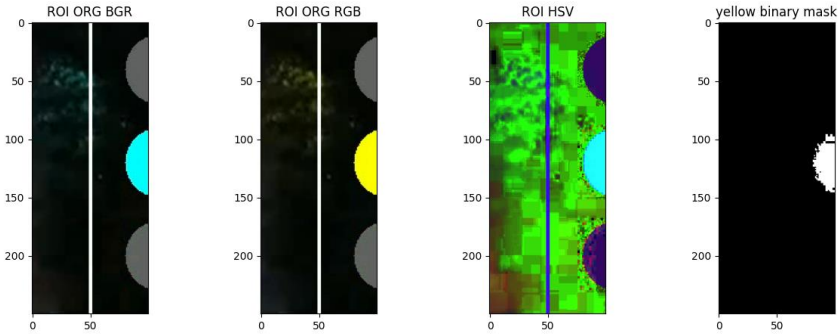
- Orange 0-22
- Yellow 22- 38

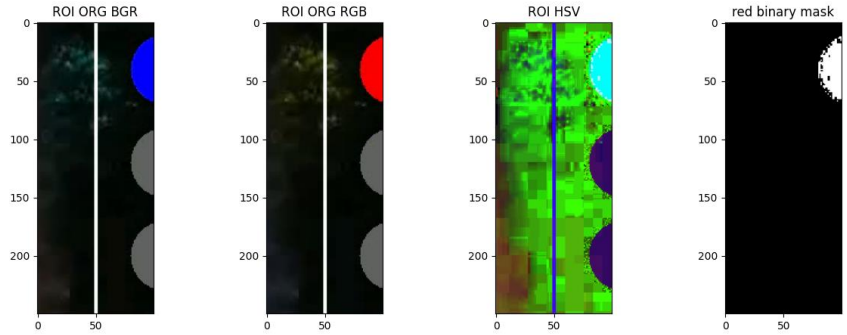
- Green 38-75
- Blue 75-130
- Violet 130-160
- Red 160-179

- Định nghĩa dải màu: ta xác định phạm vi giá trị màu sắc tương ứng với màu đỏ, màu để phát hiện xem đèn giao thông đang hiển thị màu đỏ, vàng hay xanh lục. Các giá trị màu dựa trên dải đỏ và vàng trên lý thuyết
  - Dải đỏ: giới hạn dưới [0, 100, 100], giới hạn trên [10, 255, 255]
  - Dải vàng: giới hạn dưới [20, 100, 100], giới hạn trên [30, 255, 255]
- Đặt mặt nạ màu: Sử dụng các phạm vi được xác định trước đó, mặt nạ (hình ảnh nhị phân) được tạo để làm nổi bật các vùng trong ROI nơi phát hiện màu đỏ và vàng.

Mặt nạ nhị phân là một ma trận 2 chiều gồm các phần tử 0 và 1. Sử dụng hàm cv2.inRange(roi, upper, lower) với upper và lower là màu giới hạn màu đã được định nghĩa, kết quả trả về sẽ là mặt nạ nhị phân với chứa các phần tử 1 tại vị trí thuộc vùng (lower, upper), 0 ở các vị trí còn lại.

- Nhận biết màu sắc: Sau đó ta kiểm tra các mặt nạ. Nếu có bất kỳ giá trị nào khác 0 trong mặt nạ màu đỏ, chúng ta suy ra đèn có màu đỏ. Nếu có các giá trị khác 0 trong mặt nạ màu vàng thì đèn có màu vàng. Nếu không phát hiện thấy màu đỏ hay màu vàng, chúng ta kết luận đèn có màu xanh lục.





ROI, hsv và mặt nạ nhị phân tương ứng với màu vàng/đỏ nhận diện được với cv2

- Thông tin lớp overlay: Cuối cùng, ta phủ một thông báo văn bản lên hình ảnh chính cho biết trạng thái đèn giao thông được phát hiện. Điều này sẽ thông báo một cách trực quan nếu tín hiệu cho biết "Stop", "Caution", hoặc "Go".

3.1.2 Hàm nhận diện

```
def detect_traffic_light_color(image, rect) -> image, color
```

Đầu vào:

- image: ảnh đọc được từ video thông qua Opencv
- rect: vùng roi biển báo giao thông mà ta quan tâm, là một hình chữ nhật có dạng (x, y, w, h)

Đầu ra:

- image: ảnh mới với sự thay đổi ở lớp overlay text dựa trên roi
- color: màu sắc từ vùng đèn giao thông - roi nhận diện được

3.2 Nhận diện vạch kẻ đèn giao thông

Ta sẽ định nghĩa một lớp trong phần này. Mục tiêu chính của lớp LineDetector là phát hiện đường dừng trong khung hình video. Sau khi xác định được vị trí, ta làm nổi bật đường thẳng một cách trực quan, theo trạng thái đèn giao thông được phát hiện (đỏ, vàng hoặc xanh lục)

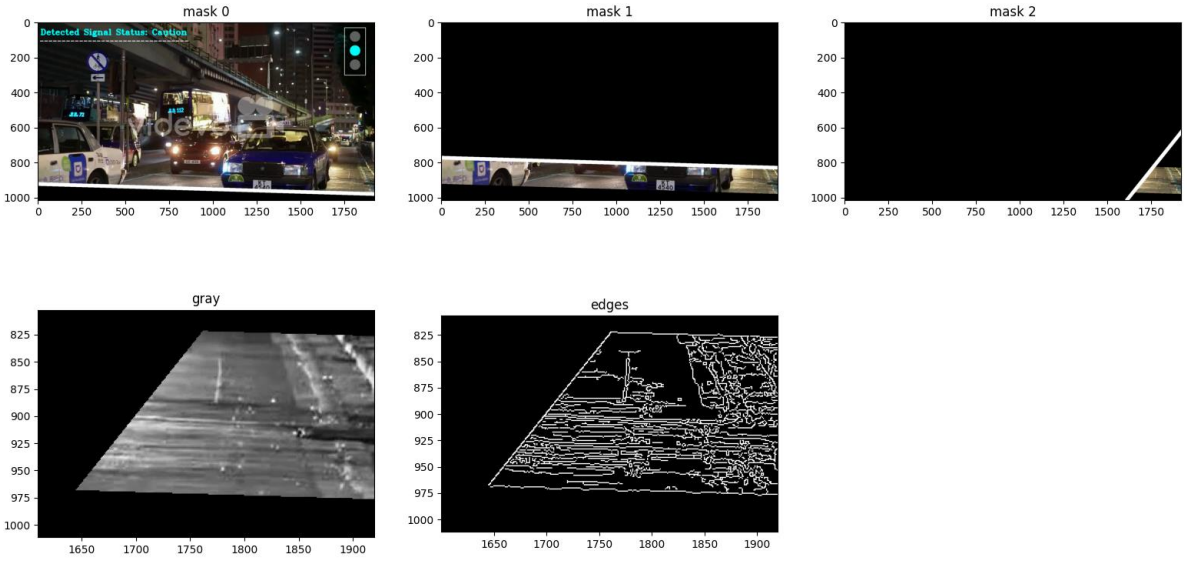
Do hiện tượng tự nhiên là hiện tượng giật (khựng) hình và biến động (rung lắc nhẹ) trên các khung hình video nên việc duy trì hình ảnh ổn định là rất quan trọng. Đây là lúc khái niệm về hàng đợi - deque, phát huy tác dụng. ta có thể theo dõi vị

trí của đường trên một số khung hình trước đó, giúp làm mượt các thay đổi đột ngột và đảm bảo trình bày hình ảnh nhất quán hơn.

Cách lớp LineDetector vận hành:

1. Khởi tạo: bắt đầu với hai hàng đợi (y_start_queue & y_end_queue) lưu trữ tọa độ y của điểm bắt đầu và điểm kết thúc của vạch kẻ dừng trên các khung.
2. Ánh xạ mã màu: Hàm lồng get_color_code <-detect_white_line, liên kết tên màu (‘red’, ‘green’, ‘yellow’) với mã màu BGR của chúng. Việc ánh xạ này được sử dụng để tự động đánh dấu vạch dừng dựa trên màu của đèn giao thông - đạt được bằng *nhận diện biển báo giao thông và trạng thái đèn*
3. Phương trình đường thẳng: Dựa trên các giá trị độ dốc và giao điểm được cung cấp, ba phương trình đường được xây dựng để tập trung vào đoạn quan tâm (segment of interest).

   Các giá trị độ dốc vào giao điểm được hard-coded dựa trên video demo
4. Tạo mặt nạ: Bắt đầu với các phương trình đường, nhiều mặt nạ được tạo ra để thu hẹp vùng quan tâm. Cuối cùng, những thứ này được kết hợp để tạo ra một mặt nạ cuối cùng làm nổi bật vạch dừng đồng thời giảm thiểu sự xao lãng từ các yếu tố khác trong khung hình.



Vùng quan tâm để nhận diện vạch dừng và xử lý cạnh

- Ở mask 1, ta kẻ đường thẳng và đưa các pixel dưới vùng này thành 0
- Ở mask 2, ta kẻ đường thẳng và đưa các pixel trên vùng này thành 0

- Ở mask 3, ta kẻ đường thẳng và đưa các pixel phần bên trái vùng này thành 0

5. Xử lý hình ảnh: Mặt nạ tinh chỉnh trải qua các biến đổi, bao gồm:
   - Chuyển đổi thang độ xám với **`cv2.cvtColor(mask3, cv2.COLOR_BGR2GRAY)`**
     Công thức chuyển đổi với mỗi pixel R, G, B sang thang độ xám tương ứng là:
     Y=0.299*R + 0.587*G + 0.114*B
   - Phát hiện cạnh sử dụng cv2.Canny(frame, threshold1, threshold2)
     Canny Edge detect là một thuật toán phát hiện cạnh phổ biến. Nó được phát triển bởi John F. Canny.
     Đặc điểm của `cv2.Canny()`:
     - Là một thuật toán nhiều giai đoạn
     - Loại bỏ nhiễu trong ảnh bằng bộ lọc Gaussian 5x5.
     - Tìm độ dốc cường độ (Intensity Gradient) của hình ảnh
       Hình ảnh được làm mịn được lọc bằng kernel Sobel theo cả hướng ngang và hướng dọc để lấy đạo hàm bậc nhất theo hướng ngang (Gx) và hướng dọc (Gy). Từ hai hình ảnh này, chúng ta có thể tìm độ dốc cạnh và hướng cho từng pixel.

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

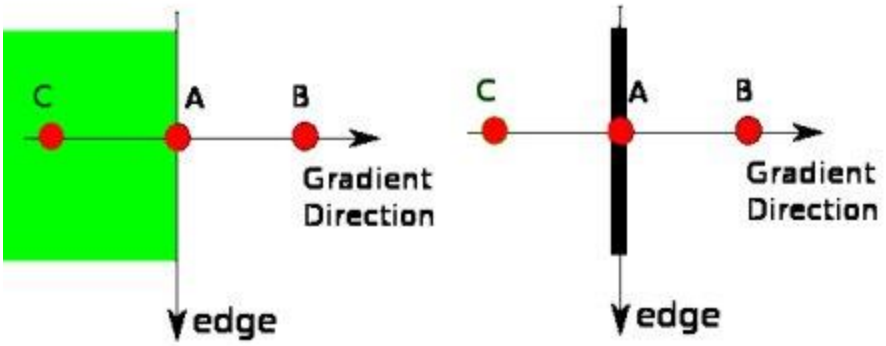
X – Direction Kernel

-1	0	1
-2	0	2
-1	0	1

Y – Direction Kernel

-1	-2	-1
0	0	0
1	2	1

       Hướng gradient luôn vuông góc với các cạnh. Nó được làm tròn thành một trong bốn góc tượng trưng cho các hướng dọc, ngang và hai đường chéo.
     - Non-maximum Suppression
       Sau khi nhận được độ lớn và hướng gradient, quá trình quét toàn bộ hình ảnh được thực hiện để loại bỏ bất kỳ pixel không mong muốn nào có thể không tạo thành cạnh. Đối với điều này, tại mỗi pixel, pixel sẽ được kiểm tra xem nó có phải là mức tối đa cục bộ trong vùng lân cận của nó theo hướng gradient hay không.

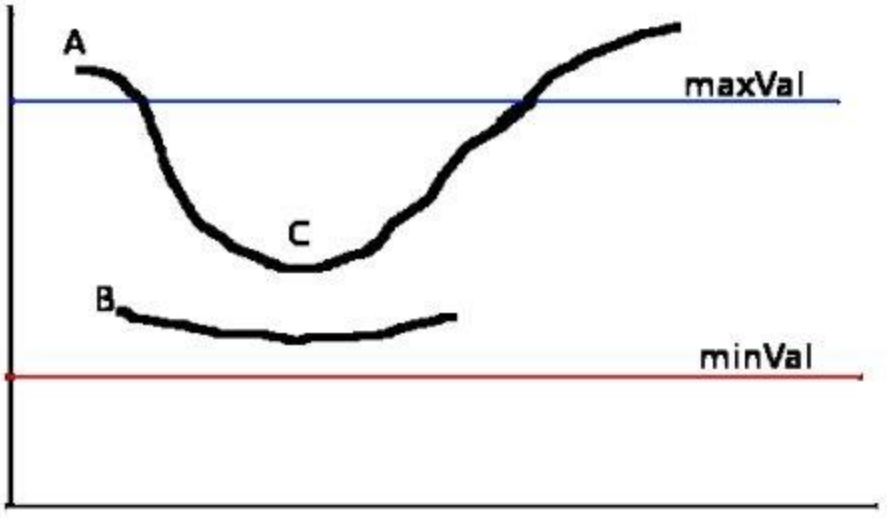


Điểm A nằm trên cạnh theo hướng thẳng đứng. Hướng gradient là bình thường đối với cạnh. Điểm B và C có hướng dốc. Vì vậy, điểm A được kiểm tra với điểm B và C để xem liệu nó có tạo thành cực đại cục bộ hay không. Nếu vậy, nó được xem xét cho giai đoạn tiếp theo, nếu không, nó sẽ bị loại bỏ (đưa về 0).

Nói tóm lại, kết quả nhận được là một hình ảnh nhị phân có "cạnh mỏng".

- Hysteresis Thresholding - Ngưỡng trễ

  Giai đoạn này quyết định tất cả các cạnh đều là cạnh thực sự và cạnh nào không. Để làm được điều này, chúng ta cần hai giá trị ngưỡng, threshold1 và threshold2. Bất kỳ cạnh nào có gradient cường độ lớn hơn threshold2 chắc chắn là cạnh và những cạnh dưới threshold1 chắc chắn không phải là cạnh, do đó bị loại bỏ. Những người nằm giữa hai ngưỡng này được phân loại là cạnh hoặc không cạnh dựa trên khả năng kết nối của chúng. Nếu chúng được kết nối với các pixel "chắc chắn" thì chúng được coi là một phần của các cạnh. Nếu không, chúng cũng bị loại bỏ.



Giai đoạn này cũng loại bỏ nhiễu pixel nhỏ với giả định rằng các cạnh là các đường dài.

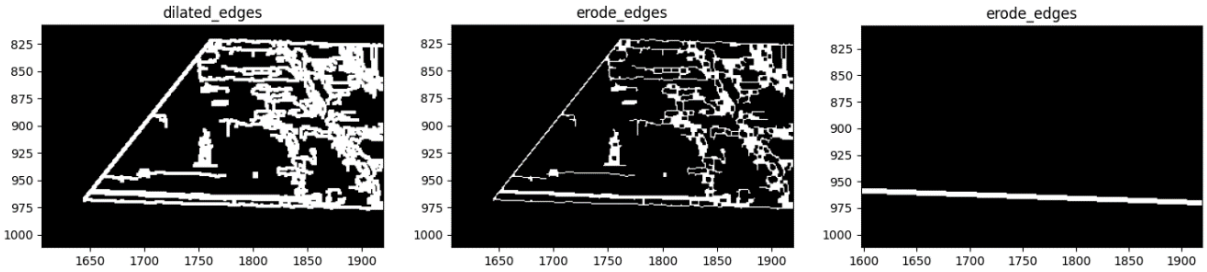
Kết quả thu được cuối cùng sẽ là ảnh với các cạnh chắc chắn (axis *edges* ở hình trên)

- Biến đổi hình học trên ảnh phát hiện cạnh Canny
  - **`dilated_edges = cv2.dilate(edges, None, iterations=1)`**

    Dilation có tác dụng làm tăng kích thước của các vùng có giá trị 1 trong ảnh nhị phân. Nó giúp kết hợp các cạnh gần nhau và làm đậm các cấu trúc trong hình ảnh.
    - edges: Là ảnh kết quả sau bước phát hiện cạnh (Canny). Đây là ảnh nhị phân chứa các cạnh đã được xác định.
    - None: Là kernel hoặc structuring element. Khi None được sử dụng, OpenCV tự động tạo kernel có kích thước 3x3 với tất cả các giá trị bằng 1.
    - iterations=1: Đây là số lần lặp để thực hiện phép dilation. Mỗi lần lặp sẽ tăng kích thước của các cạnh và đóng các khoảng trống giữa chúng.
  - **`edges = cv2.erode(dilated_edges, None, iterations=1)`**

    Erosion có tác dụng làm giảm kích thước của các vùng có giá trị 1 trong ảnh nhị phân. Nó giúp loại bỏ nhiễu và giữ lại các đặc trưng chính của các cạnh. Mục tiêu của nó là mở rộng các vùng tối

Các tham số đều đặt giống **`cv2.dilate()`**, với `dilated_edges`: Là ảnh đã được thực hiện phép dilation. Ảnh này chứa các cạnh đã được mở rộng.



6. Hough Line Transform:

`cv2.HoughLinesP(edges, 1, np.pi/180, 100, minLineLength=160, maxLineGap=5)`

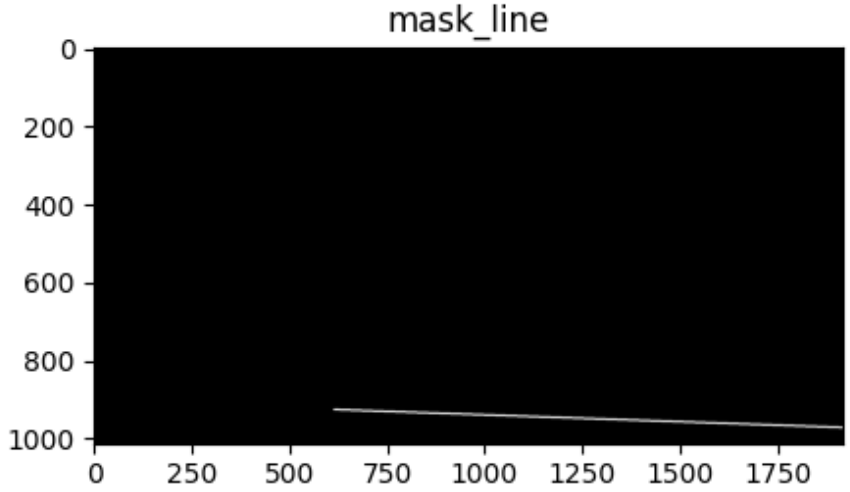
- Input:
  - dst: Kết quả của bộ phát hiện cạnh. Nó nên là một ảnh mức xám.
  - lines: Một vector sẽ lưu trữ các tham số (xstart, ystart, xend, yend) của các đoạn thẳng được phát hiện.
  - rho: Độ phân giải của tham số r tính bằng pixel. Chúng ta sử dụng 1 pixel.
  - theta: Độ phân giải của tham số θ tính bằng radian. Chúng ta sử dụng 1 độ (CV_PI/180).
  - threshold: Số tối thiểu của sự giao nhau để phát hiện một đoạn thẳng. Thay đổi tham số này dẫn đến số lượng đường thẳng phát hiện được
  - minLineLength: Số điểm tối thiểu có thể tạo thành một đoạn thẳng. Các đoạn thẳng có ít hơn số điểm này sẽ không được xem xét. *Đây là thành phần hard-coded, dựa theo độ dài của vạch trên vùng quan tâm. Từ đó ngăn các nhiễu tạo nên đường thẳng không phù hợp*
  - maxLineGap: Khoảng trống tối đa giữa hai điểm để được coi là cùng một đoạn thẳng.
- Output: cung cấp đầu ra các điểm cực trị của các dòng được phát hiện (x0, y0, x1, y1)

  Ví dụ: `[[[1650  961 1852  968]]]`

Sử dụng bước quan trọng này, Biến đổi đường Hough được áp dụng để phát hiện đường dừng dựa trên các cạnh đã xác định trong ảnh đã xử lý.

Các tham số của vạch dừng được phát hiện sau đó sẽ được tích lũy trong hàng đợi.

Vị trí vạch dừng sẽ được đo và hard-coded vị trí nó sẽ hiển thị đối với video demo, bằng cách kéo dài vạch kẻ dựa trên phương trình đường thẳng đi khắp màn hình theo chiều ngang, và cắt 30% vạch kẻ ở phía bên trái –bên đường ngược chiều



Đôi khi ta sẽ không nhận diện được tại một số ít các khung hình do các yếu tố bên ngoài. Để cải thiện điều đó, ta sử dụng deque để luôn tính giá trị trung bình các các vị trí đã phát hiện được trong quá khứ và hiện tại. Việc xử lý vị trí của điểm đầu và cuối đường thẳng tại mỗi khung hình luôn được lấy theo giá trị trung bình của deque. Do đó các khung hình không thể nhận diện vạch dừng sẽ không làm gián đoạn quá trình của cả bài toán

7. Vẽ đường & tô màu: Đường được phát hiện, bắt nguồn từ bước trước, được phác họa trên khung ban đầu. Màu của đèn giao thông xác định màu sắc của nó, làm thay đổi các kênh của khung hình ban đầu.
8. Tạo mặt nạ cuối cùng: Để kết luận, mặt nạ cuối cùng được tạo, chuyển các pixel phía trên đường được phát hiện thành màu đen. Mặt nạ này đóng vai trò là kết quả đầu ra thứ hai trong phương pháp của chúng tôi và đóng vai trò then chốt trong các bước tiếp theo, đặc biệt là trong việc cách ly biển số của những ô tô nằm dưới vạch — nhắm mục tiêu hiệu quả vào các phương tiện đã vượt qua vạch dừng khi có tín hiệu đèn đỏ.

Vùng ảnh nhận diện biển số xe nếu có bất kì phương tiện nào tiến đến khi đèn báo đỏ

Phương thức chính của LineDetector là detect_white_line() -> frame, mask_line - trả về khung hình chính sau khi nhận diện được vạch kẻ, và một vùng mask_line nhận diện biển số xe, sau các bước xử lý ở trên

3.3 Nhận diện biển số xe

Hàm nhận diện biển số xe `extract_license_plate(frame, mask_line):`

Đầu vào gồm:

- frame: là ảnh gốc,
- mask_line là ảnh lấy được khi xe đi qua vạch kẻ ở hàm `linedetector()`

Xử lý tiền dữ liệu: đầu tiên chuyển đổi ảnh khi xe đi qua vạch kẻ sang màu xám để xử lý với các giá trị cường độ kênh đơn, sau đó áp dụng `CLAHE` để cân bằng biểu đồ, từ đó có thể nâng cao độ tương phản của hình ảnh. Do sử dụng Haar cascades để phát hiện biển số xe nên ảnh cần được chuyển sang màu xám, áp dụng `CLAHE` để tặng độ tương phản cho độ sáng tối giúp nổi bận các chi tiết trong ảnh.

Chuyển sang màu xám:

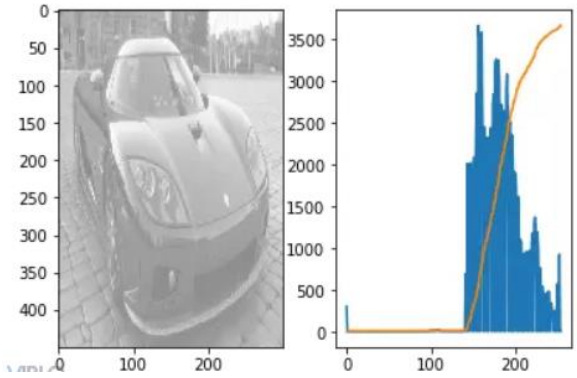
```
gray = cv2.cvtColor(mask_line, cv2.COLOR_BGR2GRAY)
```

Với `mask_line` là ảnh gốc truyền vào, `cv2. COLOR_BGR2GRAY` là mã màu chỉ định chuyển mã màu BGR thành màu xám.

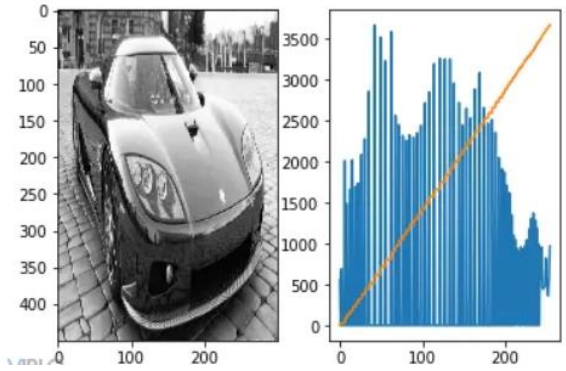
Thêm `CLAHE` cho ảnh:

- `clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))`, với clipLimit là giới hạn của độ tương phản của hình ảnh, còn tileGridSize=(8,8) nghĩa là kích thước các vùng nhỏ mà hình ảnh được chia thành, ví dụ với hình ảnh có kích thước 640x480 thì hình ảnh sẽ được chia thành 80 vùng nhỏ với mỗi vùng có kích thước 80x60

  Hàm CLAHE áp dụng thuật toán cân bằng histogram áp dụng hàm phân phối tích lũy (CDF)
  Với ví dụ như hình dưới ta thấy giá trị các pixel trong ảnh phân phối không đồng đều, để giúp cải thiện cân bằng độ tương phản, ta sẽ chia nhỏ mỗi vùng, sau đó tính histogram của vùng đó rồi dùng hàm CDF để tính giá trị ngưỡng.



Ảnh trước khi cân bằng



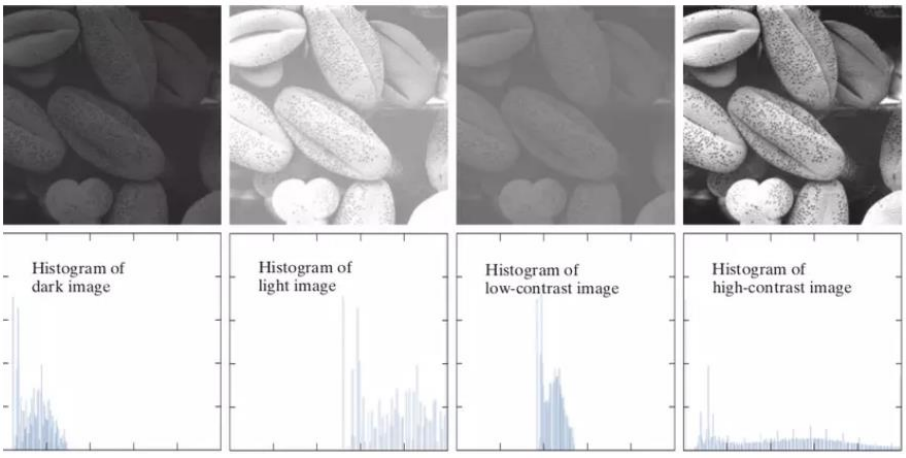
Ảnh sau khi cân bằng

  Công thức của hàm CDF:

$$New\_value = \frac{255}{max(Z(i)) - \alpha} * (Z(i) - \alpha)$$

  - Z(i) là giá trị pixel gốc
  - Max(Z(i)) là giá trị pixel lớn nhất vùng
  - α là ngưỡng

  Nếu chúng ta thay đổi giá trị ngưỡng tùy vào vùng giá trị sẽ cho độ phân bố pixel khác nhau dẫn đến các độ tương phản khác nhau.



- `clipped_gray = clahe.apply(gray)` gán CLAHE cho ảnh

Sau đó chúng ta có thể xóa các đốm đen lẻ để tăng độ chính xác cho ảnh

```
kernel = np.ones((2, 2), np.uint8)

clipped_gray = cv2.erode(clipped_gray, kernel, iterations=1)
```

Loại bỏ nhiễu bằng phương pháp bỏ nhiễu erosion, với cách hoạt động sử dụng một ma trận kernel 2x2 màu đen và di chuyển nó trong hình ảnh để tìm các điểm trắng và biến nó thành các điểm đen

Với các này chúng ta có thể giúp ảnh chống bị các chấm đen li ti trên ảnh tránh ảnh bị nhiễu.

Ảnh xám

Anh thêm CLAHE

Tiếp theo chúng ta sẽ cắt xén vùng ROI: Sau việc xử lý tiền dữ liệu từ các bước trước, trong đó các pixel phía trên vạch dừng có màu đen, mã xác định các pixel không phải màu đen cho biết đường bên dưới vạch dừng. Nó tính toán một khung giới hạn xung quanh khu vực này và sau đó cắt nó, đồng thời loại trừ một chút phần ở phía bên trái thuộc làn đường đối diện. Vùng bị cắt tập trung này trở thành ROI chính cho việc phát hiện biển số xe, vì bộ phân loại tầng Haar hoạt động hiệu quả hơn trên các ROI nhỏ hơn, được xác định rõ ràng.

Ta sử dụng biến `non_black_points = cv2.findNonZero(clipped_gray)` để tìm tất cả các pixel bằng 0 trong ảnh, sau đó trả về mảng Numpy gồm các điểm 2D.

Sau đó ta gán `x, y, w, h = cv2.boundingRect(non_black_points)` tìm hình chữ nhật bao quanh tất cả các pixel không bằng 0 trong ảnh, với hàm cv2.boundingRect giúp tính toán và trả về hình chữ nhật bao quanh tập hợp các điểm ảnh thì sẽ các biến kia thành tọa độ trên ảnh.

Và rồi cắt ảnh để thu hẹp vùng ROI `cropped_gray = gray[y:y+h, x:x+w]`

Sau khi cắt

- y:y+h biểu thị cho một slice bắt đầu từ tọa độ y tới y+h với y là tọa độ góc trên cùng bên trái của hình chữ nhật và h là chiều cao của hình chữ nhật.

- x:x+w biểu thị cho một slice bắt đầu từ tọa độ x tới x+w với x là tọa độ góc trên cùng bên trái và w là chiều rộng của hình chữ nhật.

Tiếp theo là bộ phân loại tầng Haar: Hình ảnh thang độ xám đã cắt sau đó được chuyển đến bộ phân loại tầng Haar, một phương pháp học tập không sâu cổ điển có khả năng nhận dạng mẫu. Trình phân loại này đã được đào tạo trước để nhận dạng các mẫu biển số xe và nó trả về vị trí của các biển số xe tiềm năng ở dạng hình chữ nhật.

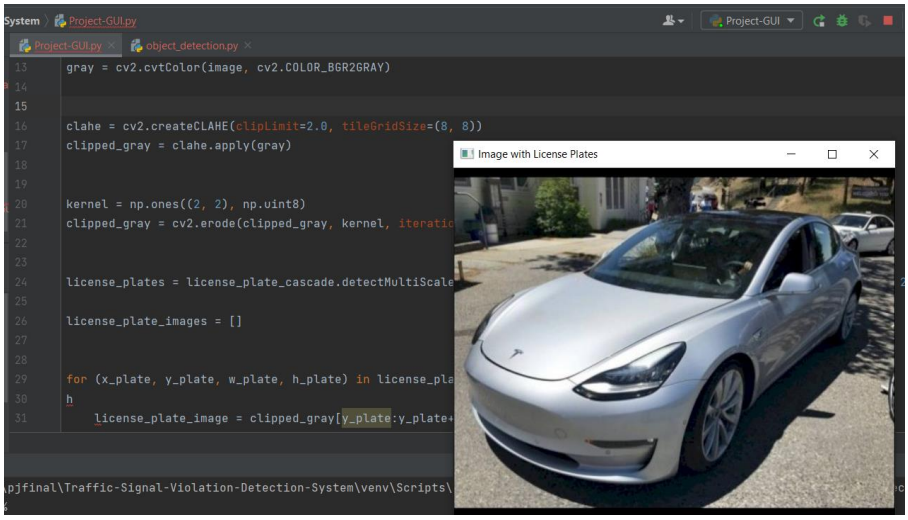
Với

```
license_plates =
license_plate_cascade.detectMultiScale(cropped_gray,scaleFactor=1.07
,minNeighbors=15,minSize=(20, 20))
```

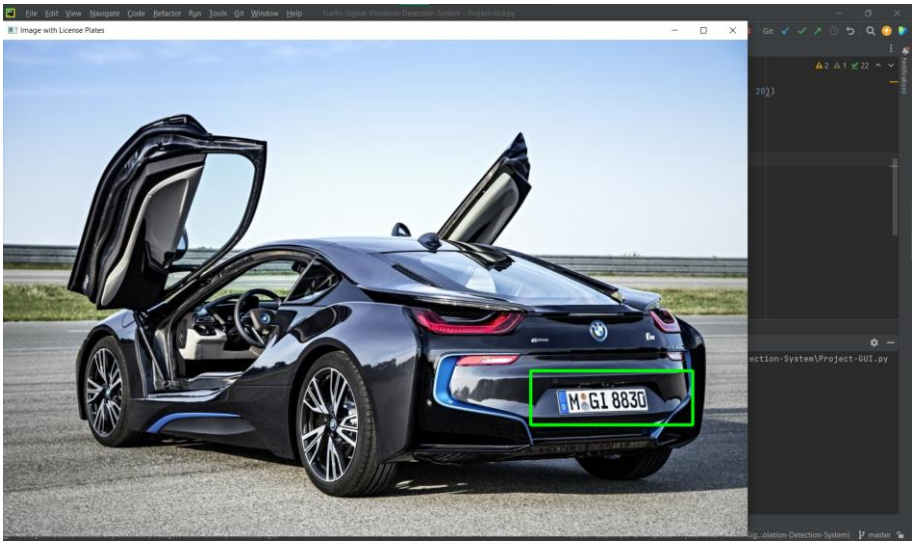
trong đó scaleFator là mức độ thu nhỏ hình ảnh sau mỗi lần phát hiện hình chữ nhật, minNeighbors xác định số lượng hình chữ nhật lân cận tối thiểu cần thiết để xác định một hình chữ nhật là biển số xe hợp lệ ,Minsize là kích thước tối thiểu của biển số xe sau đó lưu danh sách vào `license_plate_images = []`

Cuối cùng là trích xuất biển số xe trên ảnh gốc.

Đánh giá độ chính xác, thử nghiệm qua với ảnh có biển và ảnh không biển:



Đối với ảnh không biển số



Đối với ảnh có biển số

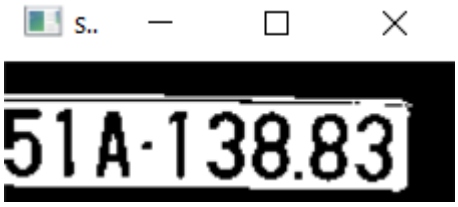
3.4 Nhận dạng văn bản trên biển số

Hàm nhận dạng văn bản trên biển số xe

```
def apply_ocr_to_image(license_plate_image)
```

Tham số đầu vào là biển số xe đã được tách ở phần nhận dạng biển số.

Đầu tiên chúng ta ngưỡng hóa hình ảnh: Hình ảnh được ngưỡng, chuyển đổi nó sang định dạng nhị phân. Điều này giúp phân biệt ký tự với nền, đảm bảo chữ trên biển số xe có màu đen và nền màu trắng.



Sau khi ngưỡng ảnh

```
img = cv2.threshold(license_plate_image, 120, 255, cv2.THRESH_BINARY)
```

Hàm ngưỡng hóa với các giá trị 120 nếu các pixel có giá trị cường độ dưới 120 thì sẽ được gán 0 tức màu đen còn lớn hơn 120 và nhỏ hơn 255 sẽ là 1 tức màu trắng, `cv2.THRES_BINARY` sẽ giúp so sánh để gán nhãn

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

Tiếp theo ta sử dụng PyTesseract OCR để trích xuất văn bản, nhưng vì PyTesseract OCR yêu cầu định dạng PIL nên ta phải chuyển dạng OpenCV qua nó.

Cuối cùng ta dùng ORC để nhận diện văn bản trên nền trắng:

```
full_text = pytesseract.image_to_string(pil_img, config='--psm 6')
```

với tham số `config='–psm 6'` tức cấu hình quy trình phân đoạn ảnh trong thư viện pytesseract xử lý trên một dòng duy nhất.

```
Text extracted from license plate: | 30H-999.99
```

3.5 Cập nhật các biển phạt trên màn hình

`def draw_penalized_text(frame) -> None`

Hàm này đơn giản chỉ nhận 1 frame và ghi các thông tin phát trên cửa sổ theo dõi. Đây là tính năng quan trong cho dù xử lý thời gian thực hay xử lý trên video.

3.6 Tổng hợp và theo dõi

`main()`

Sau khi định nghĩa được toàn bộ các quá trình xử lý của từng thành phần, ta sẽ ghép các thành phần đó và xử lý trên video demo. Dưới đây là bản phân rã tóm tắt của quá trình theo dõi:

1. Cấp dữ liệu video: Chức năng đọc video ngoại tuyến Traffic_video.mp4, từng khung hình và xử lý từng khung hình để phát hiện mọi hành vi vi phạm giao thông.
2. Phát hiện đèn giao thông: Màu của đèn giao thông cho khung nhất định được phát hiện bằng chức năng detect_traffic_light_color
3. Phát hiện vạch trắng: Hàm gọi lớp LineDetector để xác định vạch dừng màu trắng trên đường, làm nổi bật vạch dừng đó dựa trên màu đèn giao thông được phát hiện.
4. Phát hiện & xử lý vi phạm: Nếu màu được phát hiện là màu đỏ, chương trình sẽ sử dụng các hàm `extract_license_plate` và `apply_ocr_to_image` để xác định các phương tiện vượt vạch trắng, sau đó trích xuất và đọc biển số xe của chúng.
   Do có thể có hình ảnh chất lượng thấp được cung cấp cho hàm `apply_ocr_to_image` nên một số thông tin đọc biển số xe có thể không

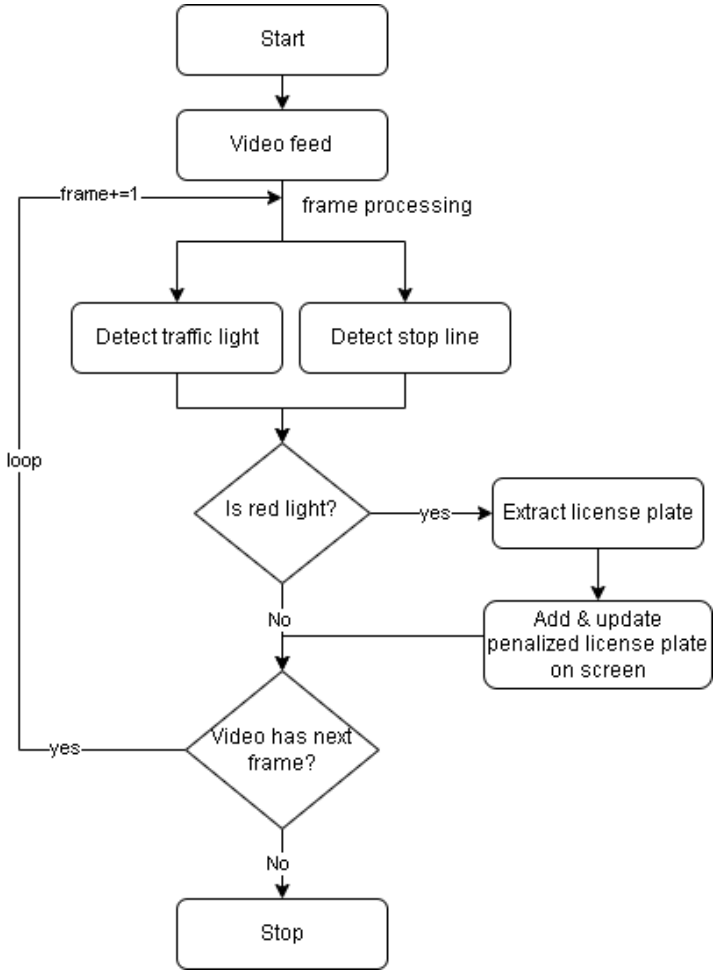
chính xác. Để khắc phục điều này, biển số được xác minh theo định dạng xác định trước, đảm bảo chúng khớp với định dạng tiêu chuẩn của biển số ô tô trong video.

Chỉ những nội dung phù hợp và chưa được ghi lại trước đó mới được ghi là vi phạm. Ta nội dung biển số xe bằng regrex `"^[A-Z]{2}\s[0-9]{3,4}$"` và một danh sách ghi lại trường hợp vi phạm `penalized_texts`

```
if text is not None \
   and re.match("^[A-Z]{2}\s[0-9]{3,4}$", text) \
   and text not in penalized_texts:
      ...
```

5. Cập nhật hiển thị: Hàm `draw_penalized_text` được gọi, hiển thị biển số xe đã bị phạt theo thứ tự trên khung.
6. Đóng chương trình: Quá trình xử lý tiếp tục cho đến khi nguồn cấp dữ liệu video kết thúc hoặc cho đến khi người dùng kết thúc bằng phím ESC. Sau đó, video sẽ dừng và tất cả các cửa sổ GUI sẽ đóng lại.

Tóm tắt quá trình thực hiện:



4. Một số kết quả ghi lại được từ demo

- Đèn xanh

...

```
Lines detected: [[[1668  939 1858  949]]]
```

- Đèn vàng

...

```
Lines detected: [[[1706  891 1906  905]]]
```

- Đèn đỏ


```
...
Lines detected: [[[1710  886 1875  895]]]
Lines detected: None
Lines detected: None
Lines detected: None
Lines detected: None
Fined license plate: YB 6433
```


```
...
Lines detected: None
```

```
Lines detected: None

Fined license plate: NN 773
```

C. Các nguồn tham khảo

- Nguồn gốc: https://www.kaggle.com/code/farzadnekouei/traffic-red-light-running-violation-detection
- Tài liệu cơ sở lý thuyết: OpenCV tutorial, Internet…