TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

NGUYỄN THỊ THÚY VY - 232805404
HOÀNG ĐÌNH QUÝ VŨ - 521H0517

BÁO CÁO CUỐI KỲ

XỬ LÝ ẢNH SỐ

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

NGUYỄN THỊ THÚY VY - 232805404
HOÀNG ĐÌNH QUÝ VŨ - 521H0517

BÁO CÁO CUỐI KỲ

XỬ LÝ ẢNH SỐ

Người hướng dẫn
TS. Trịnh Hùng Cường

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

LỜI CẢM ƠN

Chúng em xin bày tỏ lòng biết ơn chân thành đến thầy Trịnh Hùng Cường vì những kiến thức quý báu mà thầy đã truyền đạt và sự tận tâm trong việc giảng dạy môn Xử lý ảnh số. Chúng em cảm nhận được sự chuyên nghiệp và đam mê của thầy trong việc truyền đạt tri thức, và chúng em rất biết ơn vì thầy đã dành thời gian và công sức để hướng dẫn chúng em trong quá trình học tập và tìm hiểu về lĩnh vực này.

Thầy đã truyền đạt những kiến thức sâu sắc và chi tiết về Xử lý ảnh số, giúp chúng em hiểu rõ hơn về khung phát triển này và cách áp dụng vào thực tế. Nhờ những điều thầy đã truyền dạy, chúng em đã nắm vững cách xử lý dữ liệu một cách hiệu quả, đem lại kết quả tốt trong ứng dụng thực tế.

Chúng em cũng biết ơn vì sự quan tâm và hỗ trợ tận tình của thầy trong quá trình học tập. Thầy đã luôn sẵn sàng trả lời các câu hỏi của chúng em và giúp đỡ chúng em vượt qua những khó khăn trong quá trình nắm bắt kiến thức. Nhờ đó, chúng em đã có thêm niềm tin và động lực để tiếp tục khám phá và phát triển trong lĩnh vực Xử lý ảnh số.

Chúng em cảm nhận được sự chuyên nghiệp và đam mê của thầy trong việc giảng dạy. Sự cống hiến và tâm huyết của thầy đã giúp chúng em có được nền tảng vững chắc, đồng thời truyền cảm hứng để chúng em tiếp tục theo đuổi đam mê và ước mơ của bản thân.

Với tấm lòng biết ơn sâu sắc, chúng em xin kính chúc thầy Trịnh Hùng Cường sức khỏe dồi dào, hạnh phúc và ngày càng thành công trong việc truyền tải tri thức và hỗ trợ sinh viên. Mong rằng những đóng góp của thầy sẽ tiếp tục lan tỏa và mang lại những thành tựu to lớn cho thầy và cả khoa Công nghệ thông tin.

TP. Hồ Chí Minh, ngày 30 tháng 07 năm 2024

Tác giả

(ký tên và ghi rõ họ tên)

VY

Nguyễn Thị Thúy Vy

VŨ

Hoàng Đình Quý Vũ

CÔNG TRÌNH ĐƯỢC HOÀN THÀNH
TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của TS. Trịnh Hùng Cường. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong Dự án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung Dự án của mình. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 30 tháng 07 năm 2024

Tác giả

(ký tên và ghi rõ họ tên)

VY

Nguyễn Thị Thúy Vy

VŨ

Hoàng Đình Quý Vũ

TÊN ĐỀ TÀI

TÓM TẮT

Bài báo cáo trình bày các nghiên cứu về những vấn đề sau:

Vấn đề 1: Phương pháp giải quyết bài toán Tự động vẽ các hình chữ nhật bao quanh và xuất ra nội dung của biển báo

Vấn đề 2: Kết quả bài toán

TÓM TẮT

Trong bài toán nhận dạng nội dung biển báo giao thông, chúng tôi đã phát triển một phương pháp tự động phát hiện, phân loại và trích xuất nội dung từ 15 loại biển báo giao thông khác nhau. Hệ thống bao gồm các bước chính: phát hiện biển báo, vẽ hình chữ nhật bao quanh từng biển báo, và nhận dạng nội dung.

Cụ thể, hình ảnh đầu vào được chuyển đổi sang không gian màu **HSV** để thực hiện lọc màu, từ đó phát hiện các vùng có màu sắc đặc trưng của biển báo như đỏ và xanh dương. Sau đó, các biển báo hình tròn được phát hiện bằng kỹ thuật **Hough Circle Transform**, kết hợp với việc loại bỏ các đối tượng nhiễu dựa trên kích thước và vị trí của các hình tròn. Những vùng quan tâm (ROI) được trích xuất từ các hình tròn này để tiếp tục xử lý và nhận dạng nội dung.

Để nhận dạng nội dung biển báo, các kỹ thuật xử lý ảnh như chuyển đổi sang thang độ xám, áp dụng **adaptive thresholding**, và các phép biến đổi hình thái học như **closing** được sử dụng để làm nổi bật các chi tiết trên biển báo. Dựa trên tỷ lệ các màu đỏ, xanh, trắng và đặc trưng hình học như các đường chéo hoặc thẳng trong biển báo, hệ thống phân loại và xác định nội dung. Cuối cùng, các thông tin này được hiển thị trực tiếp trên hình ảnh đầu ra cùng với vị trí biển báo.

MỤC LỤC

CHƯƠNG 1. PHƯƠNG PHÁP XỬ LÝ BÀI TOÁN

1.1 Bài toán

Trong lĩnh vực nhận dạng biển báo giao thông, việc phát hiện và phân vùng các biển báo trong hình ảnh là một bước quan trọng nhằm xác định và trích xuất nội dung từ các biển báo. Bài toán đặt ra là tự động xác định vị trí của biển báo trong hình ảnh đầu vào, vẽ các hình chữ nhật bao quanh từng biển báo, và nhận diện nội dung biển báo. Sau khi xử lý, hình ảnh đầu ra sẽ được lưu lại với các biển báo được bao quanh bởi các hình chữ nhật, cùng nội dung đã được nhận diện và trích xuất, giúp xác định rõ vị trí và loại biển báo.

Hình ảnh đầu vào bao gồm các biển báo giao thông có đặc trưng màu sắc như đỏ và xanh dương, được phát hiện thông qua các kỹ thuật xử lý ảnh và nhận diện đối tượng.

1.2 Phương pháp xử lý

1.2.1 Xác định vị trí của biển báo cấm trong tấm hình

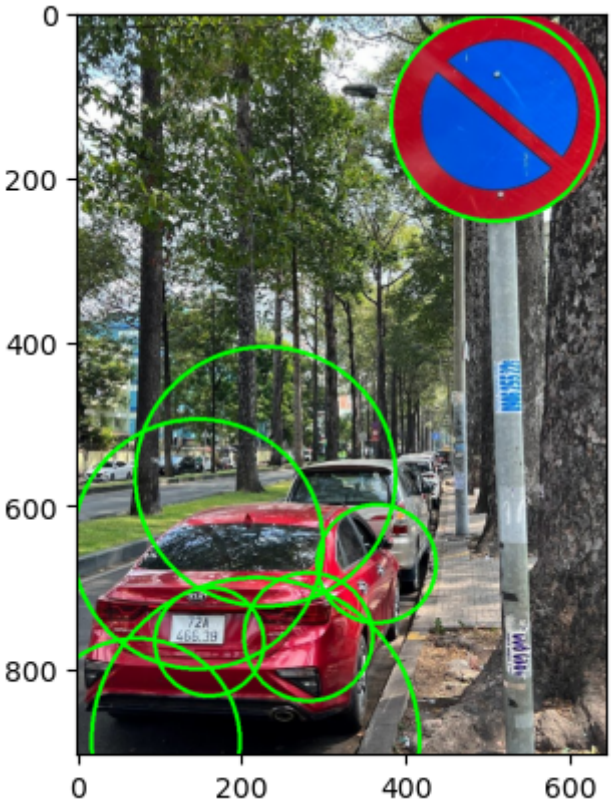
Trong việc phát hiện và xác định vị trí của biển báo giao thông là một trong những nhiệm vụ quan trọng, đặc biệt là các biển báo cấm thường có hình tròn và màu đỏ. Phương pháp được sử dụng để thực hiện nhiệm vụ này bao gồm các bước như sau:

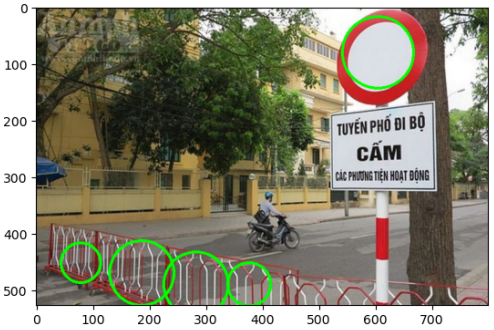
1.2.1.1 Kiểm tra khoảng cách giữa các tâm

- Hàm **are_centers_close** được sử dụng để kiểm tra xem hai tâm của các hình tròn có gần nhau không, dựa trên khoảng cách ngưỡng được cung cấp. Khoảng cách này được tính bằng cách sử dụng công thức khoảng cách Euclid. Việc kiểm tra này giúp xác định các hình tròn có vị trí gần hoặc trùng nhau để xử lý chính xác hơn.

1.2.1.2 Lọc và loại bỏ các hình tròn nhỏ

- Hàm **remove_smaller_circles** nhận vào danh sách các hình tròn đã phát hiện và loại bỏ những hình tròn có kích thước nhỏ hoặc những hình tròn có tâm gần nhau. Nếu hai hình tròn có tâm gần nhau, hình tròn lớn hơn sẽ được giữ lại. Ngoài ra, chỉ các hình tròn có bán kính lớn hơn một giá trị ngưỡng nhất định (mặc định là 20) mới được giữ lại.

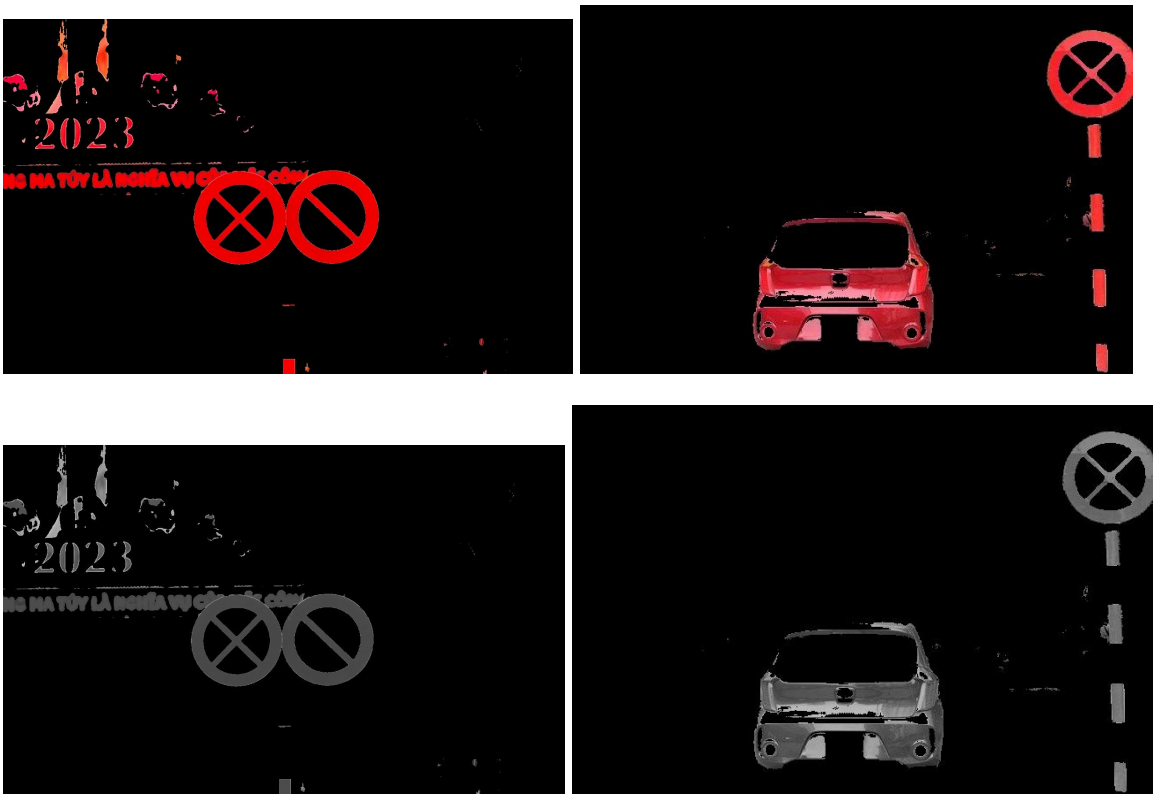




1.2.1.3 Phát hiện biển báo giao thông

- Hàm fine_num_sign xử lý hình ảnh đầu vào để phát hiện các biển báo giao thông có hình tròn (thường là biển báo cấm màu đỏ). Các bước thực hiện bao gồm:
    - Chuyển đổi ảnh từ định dạng RGB sang HSV để dễ dàng lọc màu đỏ và chuyển nó về ảnh xám để xử lý.
    - Sử dụng các mặt nạ để tìm ra các vùng màu đỏ trong ảnh.
    - Sử dụng kỹ thuật Hough Circle Transform để phát hiện các hình tròn trong ảnh dựa trên các đặc điểm như kích thước và khoảng cách.

- Áp dụng hàm **remove_smaller_circles** để loại bỏ những hình tròn nhỏ hoặc trùng nhau, giúp kết quả phát hiện chính xác hơn.
- Tùy thuộc vào kích thước của ảnh đầu vào, các tham số phát hiện hình tròn sẽ được điều chỉnh để phù hợp với từng loại ảnh



Sau khi chạy xong thì hàm sẽ trả ra vị trí x,y đó là tọa độ của biển báo. Thì mình sẽ vẽ được vị trí biển báo.

1.2.2 Xử lý nội dung của biển báo cấm

Quá trình nhận dạng nội dung biển cấm được thực hiện trong hàm **recognize_sign_content()**. Cụ thể, hệ thống sử dụng các đặc điểm hình học và tỷ lệ màu sắc để phân loại nội dung biển báo.

Các bước chính:

- Trích xuất vùng quan tâm (ROI):
    - Sau khi phát hiện các hình tròn trong ảnh (tương ứng với biển báo giao thông), hệ thống sẽ trích xuất vùng chứa biển báo, gọi là vùng quan tâm (ROI). Việc phát hiện hình tròn được thực hiện thông qua Hough Circle Transform, sử dụng các giá trị màu sắc đặc trưng của biển báo (đỏ và xanh).
    - ROI sau đó được chuyển đổi sang ảnh xám bằng lệnh gray_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY) để loại bỏ thông tin màu và làm nổi bật các chi tiết cần thiết cho quá trình phân đoạn và nhận dạng.
- Phân tích tỷ lệ màu:
    - Tách riêng các kênh đỏ (red_channel), kênh xanh (blue_channel), và kênh xám (gray_roi).
    - **Thresholding** (ngưỡng nhị phân) được áp dụng cho từng kênh để làm nổi bật các vùng màu sắc chính như đỏ, xanh và trắng. Sau đó, hệ thống tính toán **tỷ lệ màu đỏ**, **tỷ lệ màu xanh**, và **tỷ lệ màu trắng** dựa trên số lượng pixel không phải là đen trong các vùng này.
- Phân tích hình học:
    - Sử dụng các kỹ thuật phát hiện đường thẳng và đường chéo (HoughLinesP) để xác định các đặc trưng như đường chéo hoặc mũi tên trên biển báo.

```
lines = cv2.HoughLinesP(edges, rho=1, theta=np.pi/180,
threshold=40, minLineLength=80, maxLineGap=10)
```

- Phát hiện đường chéo: Nếu góc nghiêng của đường thẳng nằm trong các khoảng từ 30 đến 60 độ hoặc từ 120 đến 150 độ, hệ thống coi đây là các đường chéo dùng để xác định biển báo như cấm đỗ xe hoặc 1 phần điều kiện của cấm đỗ xe ngày lẻ hoặc cấm dừng và đỗ xe, …

```
angle = np.degrees(np.arctan2(y2 - y1, x2 - x1))

if (30 < abs(angle) < 60 or 120 < abs(angle) < 150):

    diagonal_lines.append(((x1, y1), (x2, y2), angle))
```

- Phát hiện đường ngang: Đường thẳng có góc gần 0 độ được coi là đường ngang để phát hiện biển cấm ngược chiều với điều kiện:

```
if aspect_ratio > 4.0 and h > 10 and h < 20 and w > 80 and w
< 100:  # Horizontal lines

            horizontal_lines += 1
```

- Phát hiện đường dọc: Đường thẳng có góc gần 90 độ được coi là đường dọc để phát hiện các biển cấm như cấm đỗ xe ngày chẵn với điều kiện

```
if 0.1 < aspect_ratio < 0.5 and w > 10 and w <= 40 and h > 50:

      ...
```

- Phát hiện mũi tên: sử dụng hàm **detect_arrow()** để phát hiện các biển báo liên quan mũi tên như biển báo cấm rẽ trái, rẽ phải, cấm quay đầu xe. Áp dụng Canny Edge Detection sẽ trả về một ảnh nhị phân và sử dụng hàm findContours() được sử dụng để tìm các đường viền trong

ảnh. Sau khi loại bỏ nhiễu (cv2.contourArea(contour) > 50) và số lượng đỉnh (vertices) của hình dạng phải lớn hơn hoặc bằng 7 (vì mũi tên có nhiều điểm góc). Kết hợp hàm **check_arrow_direction()** để xác định hướng mũi tên (trái/phải), cụ thể:

- Tìm điểm bên trái và bên phải nhất của mũi tên bằng cách so sánh các tọa độ của các đỉnh (vertices) trong contour:

```
leftmost = points[np.argmin(points[:, 0])]  # Điểm x nhỏ nhất

rightmost = points[np.argmax(points[:, 0])]  # Điểm x lớn nhất
```

- **Tính toán độ dốc (slope)** giữa hai điểm này để xác định hướng. Nếu độ dốc của đoạn thẳng giữa hai điểm là dương và lớn hơn 0.6, mũi tên được xác định là chỉ về bên phải, còn lại là bên trái:

```
if rightmost[0] != leftmost[0]:  # Đảm bảo không chia cho 0

    slope = (rightmost[1] - leftmost[1]) / (rightmost[0] - leftmost[0])

else:

    slope = float('inf')  # Nếu scope là vô cùng

if slope >= 0.6 and len(points) >= 9:  # Thêm kiểm tra cho số lượng điểm góc

    return 'right'
```

1.2.3 Kết hợp lại các hàm và xử lý các ảnh

Trong phần 3 này nhằm mục đích kết hợp các hàm đã được phát triển để xử lý hình ảnh, phát hiện và nhận diện nội dung của biển báo giao thông. Dưới đây là mô tả chi tiết từng bước xử lý qua 2 hàm:

1.2.3.1 Hàm reviewSign

Hàm **reviewSign** thực hiện nhiệm vụ gọi đến các hàm trước đó để sử dụng và vẽ ra vùng vị trí biển báo, cùng với viết tên biển báo lên ảnh gốc. Cụ thể đi chi tiết vào từng đoạn code chính:

- **Chuyển đổi ảnh từ BGR sang RGB**

```
img_rgb = cv2.cvtColor(image_original, cv2.COLOR_BGR2RGB)
```

Ảnh gốc đầu vào thường ở định dạng BGR (Blue-Green-Red), do đó cần chuyển đổi sang định dạng RGB để phù hợp với việc xử lý và hiển thị.

- **Gọi hàm phát hiện hình tròn**

```
filtered_circles = fine_num_sign(img_rgb)
```

Hàm **fine_num_sign** được gọi để phát hiện và lọc ra các hình tròn có thể là biển báo trong ảnh. Hàm sẽ ra ra 1 list gồm tọa độ x, y và bán kính r của biển báo. Và dựa vào đó cũng biết số lượng biển báo cấm đã nhận diện.

```
numSign = len(filtered_circles)
```

- **Vẽ hình tròn và chú thích biển báo**

```
if filtered_circles is not None:
        for i in filtered_circles:
```

Tiếp theo chạy vòng for lần lượt vào các vị trí biển báo đã xác định. Đối với mỗi hình tròn được phát hiện, hàm sẽ vẽ đường viền của hình tròn và thực hiện chú thích lên ảnh.Có thể nhận diện nội dung của biển báo bằng cách cắt vùng quan tâm (ROI - Region of Interest) và sử dụng các hàm nhận diện như **recognize_sign_content.**

  - **Trích xuất ROI:**

```
roi = img_rgb[center[1] - radius:center[1] + radius,
center[0] - radius:center[0] + radius]
```

Vùng hình tròn được trích xuất từ ảnh để nhận dạng nội dung biển báo bên trong.

- Lấy nội dung biển báo

```
if numSign == 1:
    sign_content = recognize_sign_content(roi, numSign)
else: # numSign == 2
      sign_content = recognize_sign_content_2
                                (sign_content,roi, numSign)
signList.append(sign_content)
```

Dựa vào số lượng biển báo mà chạy vô các hàm tương ứng để có được nội dung của các biển và rồi add vào list **signList.**

- **Vẽ vùng bao quanh biển báo bằng màu xanh**

```
cv2.circle(img_rgb, center, radius, (0, 255, 0), 2)
```

Tiếp theo, dựa vào số lượng biển (numSign) mà ta có 2 cách vẽ khác nhau. Với loại ảnh có 1 biển thì nhóm em vẽ nằm ở góc dưới bên trái biển. Còn với loại 2 biển thì sẽ đánh số vị trí biển và để nội dung ở góc trái ảnh.

```
if(numSign == 1):
    ......
else:
    ......
```

Với numSign ==1, cụ thể chi tiết như sau:

- **Thiết lập vị trí vẽ ban đầu, font chữ, kích thước và độ dày**:

```
text_x = max(center[0] - radius - 100, 30)
text_y = max(center[1] + radius + 30 , 20)
# Split the text if it exceeds a certain width
```

```
font = cv2.FONT_HERSHEY_SIMPLEX
font_scale = 0.8
thickness = 3
if(y < 600):
    font_scale = 0.4
    thickness = 2
```

- **font = cv2.FONT_HERSHEY_SIMPLEX**: Chọn font chữ "Hershey Simplex" từ thư viện OpenCV để vẽ văn bản.
- **font_scale = 0.8**: Đặt kích thước font chữ là 0.8. Đây là một thông số để điều chỉnh kích thước văn bản.
- thickness = 3: Đặt độ dày của chữ là 3.
- **if(y < 600)**: Kiểm tra chiều cao của ảnh. Nếu chiều cao ảnh nhỏ hơn 600 pixel, thì giảm kích thước font và độ dày (font_scale = 0.4 và thickness = 2) để phù hợp với ảnh nhỏ hơn, giúp văn bản không chiếm quá nhiều không gian.

- **Chia văn bản thành nhiều dòng (nếu cần):**

```
max_width = 200  # Max text width
wrapped_text = wrap_text(signList[0], max_width, font,
font_scale, thickness)
```

- **max_width = 200**: Đặt giới hạn chiều rộng tối đa của văn bản là 200 pixel. Nếu văn bản dài hơn chiều rộng này, nó sẽ được chia thành nhiều dòng.
- **wrap_text(signList[0], max_width, font, font_scale, thickness)**: Hàm wrap_text sẽ chia văn bản signList[0] (chứa nội dung biển báo) thành các dòng, sao cho mỗi dòng không vượt quá max_width là 200 pixel. Hàm này sử dụng thông tin về font chữ, kích thước và độ dày để đảm bảo việc chia dòng diễn ra chính xác.

- **Vẽ từng dòng của văn bản lên ảnh**

```
line_height = 30
```

```
for i, line in enumerate(wrapped_text):
    cv2.putText(img_rgb, line, (text_x, text_y + i * line_height), font, font_scale, (0, 255, 0), thickness)
```

- line_height = 30: Khoảng cách giữa các dòng văn bản là 30 pixel.
- for i, line in enumerate(wrapped_text): Vòng lặp này duyệt qua từng dòng văn bản đã được chia (wrapped_text).
- cv2.putText(...): Dùng hàm putText của OpenCV để vẽ từng dòng văn bản lên ảnh.
    - img_rgb: Ảnh đầu vào (đã được chuyển đổi sang định dạng RGB).
    - line: Dòng văn bản cần vẽ.
    - (text_x, text_y + i * line_height): Vị trí để vẽ dòng văn bản trên ảnh. Vị trí y được cộng thêm với i * line_height để vẽ các dòng tiếp theo bên dưới dòng trước đó.
    - font, font_scale, (0, 255, 0), thickness: Các thông số liên quan đến font chữ, kích thước, màu sắc (màu xanh lá cây (0, 255, 0)), và độ dày của văn bản.

Tiếp tục trường hợp else, với NumSign = 2:

- **Thiết lập font, màu chữ và các thông số hiển thị văn bản**

```
font = cv2.FONT_HERSHEY_SIMPLEX
font_scale = 0.6
font_thickness = 2
bottom = 30
if(y < 600):
    font_scale = 0.35
    font_thickness = 1
    bottom = 20
font_color = (0, 255, 0)  # Green text
```

- font = cv2.FONT_HERSHEY_SIMPLEX: Chọn font chữ "Hershey Simplex" từ OpenCV để vẽ văn bản.
- font_scale = 0.6: Đặt kích thước font chữ là 0.6.
- font_thickness = 2: Đặt độ dày của chữ là 2.
- bottom = 30: Đặt khoảng cách giữa các dòng văn bản.
- if(y < 600): Nếu ảnh có chiều cao nhỏ hơn 600 pixel, điều chỉnh kích thước chữ (font_scale) và độ dày (font_thickness) để phù hợp với ảnh nhỏ hơn. Đồng thời giảm khoảng cách giữa các dòng văn bản (bottom).
- Màu chữ được đặt là màu xanh lá cây (0, 255, 0).

- **Đánh số thứ tự cho các hình tròn (biển báo) đã phát hiện**

```
text = str(count_circle)
text_x = int(center[0])
text_y = int(center[1])
cv2.putText(img_rgb, text, (text_x, text_y), font,
font_scale, font_color, font_thickness, cv2.LINE_AA)
```

- text = str(count_circle): Chuyển số thứ tự của hình tròn (biển báo) thành chuỗi.
- text_x = int(center[0]), text_y = int(center[1]): Xác định tọa độ trung tâm của hình tròn để hiển thị số thứ tự.
- cv2.putText(...): Hàm này dùng để vẽ số thứ tự của hình tròn lên ảnh tại vị trí trung tâm, với các thông số về font, kích thước, màu sắc và độ dày.

- **Hiển thị danh sách các biển báo đã nhận diện ở góc trên bên trái**

```
y_start = img_rgb.shape[0] - bottom
line_height = bottom
# Draw the list of detected signs
for idx, sign in enumerate(signList):
    text = f"{idx + 1}. {sign}"
```

```
    cv2.putText(img_rgb, text, (10, y_start - idx * line_height),
font, font_scale, font_color, font_thickness, cv2.LINE_AA)
```

- y_start = img_rgb.shape[0] - bottom: Xác định vị trí y ban đầu để bắt đầu vẽ danh sách các biển báo, cách cạnh dưới của ảnh một khoảng là bottom.
- line_height = bottom: Đặt khoảng cách giữa các dòng văn bản trong danh sách là bottom.
- for idx, sign in enumerate(signList): Vòng lặp duyệt qua từng biển báo trong signList.
    - text = f"{idx + 1}. {sign}": Tạo văn bản hiển thị cho mỗi biển báo, bao gồm số thứ tự và nội dung biển báo.
    - cv2.putText(...): Hàm này vẽ danh sách các biển báo lên góc trên bên trái ảnh. Tọa độ y của mỗi dòng được điều chỉnh theo thứ tự (idx) và khoảng cách dòng (line_height).

Và cuối cùng hàm đã trả về ảnh đã được vẽ bao quanh biển cấm và viết nội dung của từng biển báo.

1.2.3.2 Hàm process_image

Hàm **process_images** có nhiệm vụ xử lý toàn bộ các ảnh trong một thư mục đầu vào, phát hiện và nhận diện các biển báo giao thông, sau đó lưu kết quả đã xử lý vào thư mục đầu ra. Dưới đây là giải thích chi tiết từng phần của hàm:

- **Khởi tạo thư mục đầu ra**

```
ver2_folder = os.path.join(output_folder, check)

if not os.path.exists(ver2_folder):

    os.makedirs(ver2_folder)
```

- ver2_folder = os.path.join(output_folder, check): Tạo đường dẫn cho thư mục con trong thư mục đầu ra, sử dụng tên được truyền vào từ tham số check.

- if not os.path.exists(ver2_folder): os.makedirs(ver2_folder): Kiểm tra xem thư mục đã tồn tại chưa. Nếu chưa, tạo mới thư mục.

- Duyệt qua tất cả các ảnh trong thư mục đầu vào

```
for filename in os.listdir(input_folder):
    if filename.endswith('.jpg') or filename.endswith('.png'):
        print(filename)
```

- for filename in os.listdir(input_folder): Duyệt qua từng tệp tin trong thư mục đầu vào.
- if filename.endswith('.jpg') or filename.endswith('.png'): Chỉ xử lý các tệp ảnh có đuôi .jpg hoặc .png. Có thể mở rộng để xử lý các định dạng ảnh khác.
- print(filename): In ra tên tệp ảnh hiện đang được xử lý.

- Đọc và xử lý ảnh

```
image_path = os.path.join(input_folder, filename)
image = cv2.imread(image_path)
if image is not None:
    processed_image = reviewSign2(image, filename)
```

- image_path = os.path.join(input_folder, filename): Tạo đường dẫn đầy đủ đến ảnh.
- image = cv2.imread(image_path): Sử dụng OpenCV để đọc ảnh từ đường dẫn.
- if image is not None: Kiểm tra xem ảnh có được đọc thành công hay không. Nếu có, tiếp tục xử lý.
- processed_image = reviewSign2(image, filename): Gọi hàm reviewSign2 để phát hiện và nhận diện biển báo giao thông trong ảnh, và trả về ảnh đã xử lý.

- Lưu ảnh đã xử lý

```
output_path = os.path.join(ver2_folder, filename)

cv2.imwrite(output_path, processed_image)

print(f"Processed and saved {filename} to {ver2_folder}")

print("--------------------------------")
```

- output_path = os.path.join(ver2_folder, filename): Tạo đường dẫn lưu ảnh sau khi xử lý vào thư mục đầu ra.
- cv2.imwrite(output_path, processed_image): Lưu ảnh đã xử lý vào đường dẫn được chỉ định.
- print(f"Processed and saved {filename} to {ver2_folder}"): In thông báo xác nhận rằng ảnh đã được xử lý và lưu thành công.
- print("--------------------------------"): Tạo đường phân cách cho các lần xử lý ảnh để dễ quan sát khi chạy chương trình.

CHƯƠNG 2. HIỆN THỰC HÓA BẰNG MÃ CODE

2.1 Đoạn mã đầy đủ để xử lý bài toán

```
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt


############################ Part1
def are_centers_close(center1, center2, threshold=5):
    """
    Check if the centers of two circles are close to each other within a
given threshold.

    Parameters:
        center1 (tuple): Coordinates of the first circle's center (x1,
y1).
        center2 (tuple): Coordinates of the second circle's center (x2,
y2).
        threshold (int, optional): The maximum allowed distance between
the  centers to consider them close. Default is 5.

    Returns:
        bool: True if the centers are within the threshold distance,
False otherwise.
    """
    # Calculate the Euclidean distance between the two centers and
compare with the threshold
    return np.linalg.norm(np.array(center1) - np.array(center2)) <
threshold


def remove_smaller_circles(circles, min_radius=20):
    """
```

```
    Remove circles that are smaller or have close centers to larger
circles from a list of detected circles.

    Parameters:
        circles (ndarray): Array of detected circles, where each circle
is represented as [x_center, y_center, radius].
        min_radius (int, optional): The minimum radius required for a
circle to be kept. Default is 20.

    Returns:
        list: A list of unique circles, each represented as [x_center,
y_center, radius].
    """
    if circles is None:
        return []

    # Convert circles to integer values and round them
    circles = np.uint16(np.around(circles))
    unique_circles = []

    for current_circle in circles[0, :]:
        add_circle = True

        for unique_circle in unique_circles:
            # Check if the current circle's center is close to any
unique circle's center
            if are_centers_close(current_circle[:2], unique_circle[:2]):
                # If current circle is larger, replace the smaller one
                if current_circle[2] > unique_circle[2]:
                    unique_circle[:] = current_circle
                add_circle = False
                break
```

```
        # Add the circle if it is unique and meets the minimum radius
requirement
        if add_circle and current_circle[2] >= min_radius:
            unique_circles.append(current_circle)

    return unique_circles

def fine_num_sign(img_rgb):
    """
    Detect circular traffic signs in an image using color filtering and
Hough Circle Transform.

    Parameters:
        img_rgb (ndarray): The input image in RGB format.

    Returns:
        ndarray: Filtered list of detected circles with center
coordinates and radius.
    """

    # Get image dimensions
    x, y, _ = img_rgb.shape

    # Convert the image from RGB to HSV for easier color filtering
    img_hsv = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2HSV)

    # Define the HSV color range for red (traffic signs are typically
red)
    lower_red1 = np.array([0, 100, 100])
    upper_red1 = np.array([10, 255, 255])
    lower_red2 = np.array([170, 100, 100])
    upper_red2 = np.array([180, 255, 255])
```

```
    # Create masks for red regions in the image
    mask1 = cv2.inRange(img_hsv, lower_red1, upper_red1)
    mask2 = cv2.inRange(img_hsv, lower_red2, upper_red2)
    red_mask = cv2.bitwise_or(mask1, mask2)

    # Extract red regions from the original image
    red_regions = cv2.bitwise_and(img_rgb, img_rgb, mask=red_mask)

    # Convert the RGB image to BGR to comply with OpenCV's color
format
    red_regions_bgr = cv2.cvtColor(red_regions, cv2.COLOR_RGB2BGR)

    # Convert the image to grayscale for contour detection
    gray = cv2.cvtColor(red_regions_bgr, cv2.COLOR_BGR2GRAY)

    # Apply a median blur to reduce noise
    grayBlur = cv2.medianBlur(gray, 5)

    circles = None

    # Detect circles using HoughCircles based on specific image
dimensions
    if (x, y) == (1706, 2560):  # Case for image size m5
        rows = gray.shape[0]
        circles = cv2.HoughCircles(grayBlur, cv2.HOUGH_GRADIENT, dp=1.2,
minDist=rows/8,
                                   param1=50, param2=40, minRadius=100,
maxRadius=200)
    elif (x, y) == (526, 800):  # Case for image size m6
        circles = cv2.HoughCircles(grayBlur, cv2.HOUGH_GRADIENT_ALT,
                                   1.2, 60, param1=100, param2=0.85,
minRadius=10)
```

```
    elif (x, y) == (903, 645):  # Case for image size m8
        circles = cv2.HoughCircles(grayBlur, cv2.HOUGH_GRADIENT_ALT,
                                   2, 30, param1=200, param2=0.85,
minRadius=30)
    elif (x, y) == (1333, 2000):  # Case for image size m10
        rows = grayBlur.shape[0]
        circles = cv2.HoughCircles(grayBlur, cv2.HOUGH_GRADIENT, dp=1.5,
minDist=rows/8,
                                   param1=50, param2=60, minRadius=100,
maxRadius=330)

    # More specific circle detection cases based on image dimensions
    elif (x, y) == (188, 268):  # m11
        circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT_ALT, 1.2,
10, param1=100, param2=0.2, minRadius=20)
    elif (x, y) == (177, 285):  # m12
        circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT_ALT, 1.5,
10, param1=150, param2=0.2, minRadius=10)
    elif (x, y) == (193, 261):  # m14
        gray = cv2.medianBlur(gray, 3)
        circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT_ALT, 1.5,
10, param1=150, param2=0.5, minRadius=22)
    elif (x, y) == (398, 600):  # m15
        circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT_ALT, 1.5,
10, param1=150, param2=0.4, minRadius=22)
    elif (x, y) == (800, 1280):  # m13
        circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT_ALT, 1.5,
10, param1=150, param2=0.5, minRadius=22)
    else:  # General case for other image sizes
        circles = cv2.HoughCircles(grayBlur, cv2.HOUGH_GRADIENT_ALT, 2,
30, param1=200, param2=0.85, minRadius=10)

    # Remove smaller circles for better accuracy
    filtered_circles = remove_smaller_circles(circles)
```

```
    return filtered_circles


############################ Part 2
def detect_arrow(gray_roi):
    """
    Detect the direction of an arrow in a given grayscale region of interest (ROI).

    Parameters:
        gray_roi (ndarray): Grayscale image of the region of interest where the arrow is expected.

    Returns:
        str: The detected direction of the arrow ('left', 'right', 'down', or '').
             Returns an empty string if no arrow direction is detected.
    """
    direction = ""
    # Apply Canny edge detection to find the edges
    edges = cv2.Canny(gray_roi, 50, 150)

    # Find contours in the image after applying Canny
    contours, _ = cv2.findContours(edges, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    for contour in contours:
        # Reduce the area threshold to avoid missing small arrow details
        if cv2.contourArea(contour) > 50:  # Reduced threshold from 100 to 50
            # Identify if the contour is an arrow by calculating the length-to-width ratio
```

```
            approx = cv2.approxPolyDP(contour, 0.02 *
cv2.arcLength(contour, True), True)


            # Check if the contour is an arrow based on the number of
sides
            if len(approx) >= 7:  # Added condition on the area
                x, y, w, h = cv2.boundingRect(approx)
                aspect_ratio = float(w) / h


                if 0.5 < aspect_ratio < 1.5:
                    direction = check_arrow_direction(approx)


                    if direction:  # If the direction is determined
                        if h > 100:
                            direction = "down"
                        break
    return direction


def check_arrow_direction(approx):
    """
    Determine the direction of an arrow based on its contour
approximation.


    Parameters:
        approx (ndarray): Array of points approximating the contour of
the arrow.


    Returns:
        str: The detected direction of the arrow ('left' or 'right').
             Returns 'left' if the slope is not steep enough or if no
clear direction is found.
    """
    # Get all points from the contour
```

```
    points = approx.reshape(-1, 2)

    # Find the leftmost and rightmost points
    leftmost = points[np.argmin(points[:, 0])]  # Point with the smallest x
    rightmost = points[np.argmax(points[:, 0])]  # Point with the largest x

    # Calculate the slope between leftmost and rightmost
    if rightmost[0] != leftmost[0]:  # Ensure no division by 0
        slope = (rightmost[1] - leftmost[1]) / (rightmost[0] - leftmost[0])
    else:
        slope = float('inf')  # If the slope is infinite

    # If the slope is not clear, check additional conditions
    if slope >= 0.6 and len(points) >= 9:  # Additional check for the number of corner points
        return 'right'
    return 'left'

def check_blue_area_symmetry(blue_mask):
    """
    Check if the blue areas in a given mask are approximately symmetrical across four quadrants.

    Parameters:
        blue_mask (ndarray): Binary mask where the blue areas are highlighted (non-zero).

    Returns:
        bool: True if the areas in the four quadrants are approximately symmetrical, False otherwise.
```

```
    """
    height, width = blue_mask.shape
    half_height = height // 2
    half_width = width // 2

    # Divide the mask into 4 parts
    top_left = blue_mask[0:half_height, 0:half_width]
    top_right = blue_mask[0:half_height, half_width:width]
    bottom_left = blue_mask[half_height:height, 0:half_width]
    bottom_right = blue_mask[half_height:height, half_width:width]

    # Calculate the area of the blue region in each part
    area_top_left = cv2.countNonZero(top_left)
    area_top_right = cv2.countNonZero(top_right)
    area_bottom_left = cv2.countNonZero(bottom_left)
    area_bottom_right = cv2.countNonZero(bottom_right)

    areas = [area_top_left, area_top_right, area_bottom_left,
area_bottom_right]

    # Check if the areas are approximately equal (allowing for some
small error)
    max_area = max(areas)
    min_area = min(areas)

    if min_area > 0 and (max_area / min_area) < 1.65:  # Allowing for a
maximum error of 65%
        return True
    return False

def detect_diagonal_lines(image):
    """
```

```
    Detect diagonal lines in the given image.

    Parameters:
        image (ndarray): Input image in BGR format.

    Returns:
        int: The number of unique diagonal lines detected.
    """
    diagonal_lines = 0
    # Convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Enhance the contrast
    gray = cv2.equalizeHist(gray)

    # Apply CLAHE for better contrast enhancement
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    gray = clahe.apply(gray)

    # Preprocess the image with GaussianBlur and Canny Edge Detection
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    edges = cv2.Canny(blurred, 50, 150, apertureSize=3)

    # Apply dilation to enhance diagonal lines
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
    edges = cv2.dilate(edges, kernel, iterations=1)

    # Detect lines using HoughLinesP
    lines = cv2.HoughLinesP(edges, rho=1, theta=np.pi/180, threshold=40,
minLineLength=80, maxLineGap=10)
```

```
    # Store the detected diagonal lines
    diagonal_lines = []

    if lines is not None:
        for line in lines:
            x1, y1, x2, y2 = line[0]
            angle = np.degrees(np.arctan2(y2 - y1, x2 - x1))  # Calculate the angle of the line

            # Group lines with similar angles (i.e., diagonal directions)
            if (30 < abs(angle) < 60 or 120 < abs(angle) < 150):
                diagonal_lines.append(((x1, y1), (x2, y2), angle))

    # Merge similar lines based on angle and distance
    merged_lines = merge_similar_lines(diagonal_lines)

    # Return the number of unique diagonal lines
    return len(merged_lines)

def merge_similar_lines(lines, angle_threshold=15, distance_threshold=30, overlap_threshold=0.5):
    """
    Merge lines that are similar in angle and close in distance.

    Parameters:
        lines (list): List of lines where each line is represented as ((x1, y1), (x2, y2), angle).
        angle_threshold (int): Angle difference threshold to consider lines similar.
        distance_threshold (int): Distance threshold to consider lines close.
```

```
        overlap_threshold (float): Overlap ratio threshold for merging
lines.

    Returns:
        list: List of merged lines.
    """
    merged_lines = []
    for line in lines:
        (x1, y1), (x2, y2), angle = line
        merged = False

        for i, merged_line in enumerate(merged_lines):
            (mx1, my1), (mx2, my2), mangle = merged_line

            # Check if angles are similar and lines are close enough
            if abs(angle - mangle) < angle_threshold:
                merged = True
                break

        if not merged:
            merged_lines.append(line)

    return merged_lines

def recognize_sign_content(roi, numSign):
    """
    Recognize the content of a traffic sign based on the region of
interest (ROI).

    Parameters:
        roi (ndarray): Region of interest in the image where the traffic
sign is located.
```

```
        numSign (int): Indicator used to distinguish between different
types of signs.

    Returns:
        str: The recognized traffic sign content.
    """

    # Convert ROI to grayscale
    gray_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)

    # Extract red and blue channels from ROI
    red_channel = roi[:, :, 2]
    blue_channel = roi[:, :, 0]

    # Apply threshold to highlight red and blue areas in the sign
    _, red_thresh = cv2.threshold(red_channel, 100, 255,
cv2.THRESH_BINARY)
    _, blue_thresh = cv2.threshold(blue_channel, 100, 255,
cv2.THRESH_BINARY)
    _, white_thresh = cv2.threshold(gray_roi, 150, 255,
cv2.THRESH_BINARY)

    # Apply adaptive threshold to highlight characters in the sign
    adaptive_thresh = cv2.adaptiveThreshold(gray_roi, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)

    # Use Morphological Transformations to clean white mask
    kernel = np.ones((5, 5), np.uint8)
    white_thresh = cv2.morphologyEx(white_thresh, cv2.MORPH_CLOSE,
kernel)
    adaptive_thresh = cv2.morphologyEx(adaptive_thresh, cv2.MORPH_CLOSE,
kernel)
```

```
    # Remove small noise in blue_mask
    blue_thresh_cleaned = cv2.morphologyEx(blue_thresh, cv2.MORPH_CLOSE,
kernel)

    # Remove white areas that might affect blue_ratio
    blue_thresh_cleaned = cv2.bitwise_and(blue_thresh_cleaned,
cv2.bitwise_not(white_thresh))
    blue_thresh_cleaned = cv2.bitwise_and(blue_thresh_cleaned,
cv2.bitwise_not(red_thresh))

    red_thresh_cleaned = cv2.morphologyEx(red_thresh, cv2.MORPH_CLOSE,
kernel)

    # Calculate total number of pixels in ROI
    total_area = red_thresh.shape[0] * red_thresh.shape[1]

    # Calculate red, blue, and white ratios in ROI
    red_area = cv2.countNonZero(red_thresh_cleaned)
    red_ratio = red_area / total_area
    blue_area = cv2.countNonZero(blue_thresh_cleaned)
    blue_ratio = blue_area / total_area
    white_area = cv2.countNonZero(white_thresh)
    white_ratio = white_area / total_area

    diagonal_lines = detect_diagonal_lines(roi)

    # Analyze the number of white lines
    contours, _ = cv2.findContours(white_thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # Filter contours based on width and height size
    vertical_lines = 0
    vertical_lines_nhoHon100 = 0
```

```
    horizontal_lines = 0
    h_prev = 0
    for cnt in contours:
        x, y, w, h = cv2.boundingRect(cnt)
        aspect_ratio = float(w) / h

        if 0.1 < aspect_ratio < 0.5 and w > 10 and w <= 40 and h > 50:
# Vertical lines - for even/odd restriction signs
            if h_prev == 0:
                h_prev = h
                continue
            if h == h_prev:
                if h_prev < 100:
                    vertical_lines_nhoHon100 += 1
                else:
                    vertical_lines += 2
            else:
                h_prev = h

        if aspect_ratio > 4.0 and h > 10 and h < 20 and w > 80 and w <
100:  # Horizontal lines
            horizontal_lines += 1

    if h_prev != 0 and vertical_lines == 0:
        if h_prev > 100:
            vertical_lines = 1
        else:
            vertical_lines_nhoHon100 = 1

    if len(contours) == 30 and 0.36 < red_ratio < 0.37 and 0.46 <
blue_ratio < 0.47 and 0.34 < white_ratio < 0.344:
```

```
        return "Bien cam xe tai tren 4 tan va xe o to khach tu 16 cho
tro len"


    elif diagonal_lines == 1 and 0.4 < red_ratio < 0.5 and 0.4 <
blue_ratio < 0.7 and white_ratio < 0.1:
        return "Bien cam do xe"


    elif horizontal_lines == 1 and vertical_lines == 0:
        return "Bien cam nguoc chieu"


    # Check red and white ratio for "No Vehicles" sign
    elif 0.5 < red_ratio < 0.7 and 0.5 < white_ratio < 0.6 and
vertical_lines == 0 and diagonal_lines == 0 and horizontal_lines == 0:
        return "Bien duong cam"


    elif diagonal_lines == 1 and red_ratio > 0.5 and blue_ratio < 0.35
and white_ratio > 0.5:
        return "Bien cam nguoi di bo"


    elif diagonal_lines == 2 and vertical_lines_nhoHon100 == 1 and 0.3 <
blue_ratio < 0.7 and white_ratio < 0.3 and not
check_blue_area_symmetry(blue_thresh_cleaned):
        return "Bien cam do xe ngay le"


    # Check arrow shape in ROI
    elif diagonal_lines == 1 and vertical_lines == 0 and
detect_arrow(gray_roi) == "left":
        return "Bien cam re trai"


    elif vertical_lines == 0 and detect_arrow(gray_roi) == "right":
        if numSign == 1:
            return "Bien cam re phai"
        else:
```

```
            return "Bien cam o to"

    elif vertical_lines == 0 and detect_arrow(gray_roi) == "down":
        return "Bien cam quay dau xe"

    elif diagonal_lines == 2 and vertical_lines == 0 and
check_blue_area_symmetry(blue_thresh_cleaned):
        if numSign == 1:
            return "Bien cam dung va do xe"
        else:
            return "Bien cam quay dau xe"

    elif vertical_lines == 2 and diagonal_lines == 3 and 0.3 <
blue_ratio < 0.7:
        return "Bien cam do xe ngay chan"

    elif 0.5 < red_ratio < 0.56 and 0.4 < blue_ratio < 0.46 and 0.49 <
white_ratio < 0.5:
        return "Bien toc do toi da cho phep 40 km/h"

    elif vertical_lines == 0 and diagonal_lines == 0 and
horizontal_lines == 0 and vertical_lines_nhoHon100 == 0:
        if numSign == 1:
            return "Bien toc do toi da cho phep 50 km/h"
        else:
            return "Bien cam vuot"

    return ""

def wrap_text(text, max_width, font, font_scale, thickness):
    """
    Wrap text into multiple lines to fit within a specified width.
```

```
    Parameters:
        text (str): The text string to be wrapped.
        max_width (int): The maximum width (in pixels) each line can
occupy.
        font (int): The font type used for the text.
        font_scale (float): The scale of the font.
        thickness (int): The thickness of the text stroke.

    Returns:
        list of str: A list of lines where each line fits within the
given width.
    """
    if not text:
        return []

    words = text.split(' ')
    lines = []
    current_line = words[0] if words else ''

    for word in words[1:]:
        # Calculate the size of the text if the word is added to the
current line
        line_size = cv2.getTextSize(current_line + ' ' + word, font,
font_scale, thickness)[0][0]

        # If the size exceeds the maximum width, start a new line
        if line_size > max_width:
            lines.append(current_line)
            current_line = word
        else:
            current_line += ' ' + word
```

```
    # Append the last line
    lines.append(current_line)
    return lines

def recognize_sign_content_2(sign_content, roi, numSign):
    """
    Update the traffic sign content based on previously recognized signs.

    This function modifies the recognized traffic sign content if two signs are detected.
    If `sign_content` is already identified, it checks for specific cases and updates the
    content accordingly.

    Parameters:
        sign_content (str): The content of the previously recognized sign.
        roi (ndarray): The region of interest (ROI) containing the detected traffic sign.
        numSign (int): The number of signs detected in the image.

    Returns:
        str: The updated sign content.
    """

    # If the first sign is already recognized
    if sign_content != "":
        # Case when the first sign is "No Overtaking" -> update to "No Parking or Stopping"
        if sign_content == "Bien cam vuot":  # m12
            sign_content = "Bien cam dung va do xe"
```

```
        # Case when the first sign is "No Pedestrian Crossing" ->
update to "Max Speed Limit 40 km/h"
        elif sign_content == "Bien cam nguoi di bo":  # m15
            sign_content = "Bien toc do toi da cho phep 40 km/h"
        # Case when the first sign is "Max Speed Limit 40 km/h" ->
update to "No Parking"
        elif sign_content == "Bien toc do toi da cho phep 40 km/h":  #
m11
            sign_content = "Bien cam do xe"
        # Case when the first sign is "No Cars" -> update to "No Taxi"
        elif sign_content == "Bien cam o to":  # m14
            sign_content = "Bien cam taxi"
        # Default case -> update to "No Parking or Stopping"
        else:  # m13
            sign_content = "Bien cam dung va do xe"
    else:
        # If no sign content is recognized yet, call the main
recognition function
        sign_content = recognize_sign_content(roi, numSign)

    return sign_content

############################ Part 3
def reviewSign(image_original):
    """
    Process an image to detect and recognize traffic signs.

    Parameters:
        image_original (ndarray): The input image in BGR format.
        filename (str): The name of the image file for logging or saving
purposes.

    Returns:
```

```
        ndarray: The image with detected circles and recognized traffic
signs.
    """
    # Convert the image from BGR to RGB
    img_rgb = cv2.cvtColor(image_original, cv2.COLOR_BGR2RGB)
    x, y, _ = img_rgb.shape

    filtered_circles = fine_num_sign(img_rgb)

    numSign = len(filtered_circles)
    sign_content = ""
    count_circle = 0
    signList = []

    # Draw the detected circles and annotate the signs
    if filtered_circles is not None:
        for i in filtered_circles:
            count_circle +=1
            if count_circle>2:
              break;

            center = (i[0], i[1])
            radius = i[2]

            # Extract the region of interest (ROI) containing the
circle
            roi = img_rgb[center[1] - radius:center[1] + radius,
center[0] - radius:center[0] + radius]

            if numSign == 1:
                sign_content = recognize_sign_content(roi, numSign)
            else: # numSign == 2
```

```
                sign_content = recognize_sign_content_2(sign_content,
roi, numSign)

            signList.append(sign_content)

            # Draw the circle on the image
            cv2.circle(img_rgb, center, radius, (0, 255, 0), 2) # Circle
boundary

            if(numSign == 1):
                text_x = max(center[0] - radius - 100, 30)
                text_y = max(center[1] + radius + 30 , 20)

                # Split the text if it exceeds a certain width
                font = cv2.FONT_HERSHEY_SIMPLEX
                font_scale = 0.8
                thickness = 3
                if(y < 600):
                    font_scale = 0.4
                    thickness = 2

                max_width = 200  # Max text width
                wrapped_text = wrap_text(signList[0], max_width, font,
font_scale, thickness)

                # Draw each line of the text
                line_height = 30

                for i, line in enumerate(wrapped_text):
                    cv2.putText(img_rgb, line, (text_x, text_y + i *
line_height), font, font_scale, (0, 255, 0), thickness)
            else:
```

```
                font = cv2.FONT_HERSHEY_SIMPLEX
                font_scale = 0.6
                font_thickness = 2
                bottom = 30

                if(y < 600):
                    font_scale = 0.35
                    font_thickness = 1
                    bottom = 20

                font_color = (0, 255, 0)  # Green text

                # Number the circles
                text = str(count_circle)
                text_x = int(center[0])
                text_y = int(center[1])
                cv2.putText(img_rgb, text, (text_x, text_y), font,
font_scale, font_color, font_thickness, cv2.LINE_AA)

                if(len(signList) == 2):
                    y_start = img_rgb.shape[0] - bottom
                    line_height = bottom
                    # Draw the list of detected signs
                    for idx, sign in enumerate(signList):
                        text = f"{idx + 1}. {sign}"
                        cv2.putText(img_rgb, text, (10, y_start - idx *
line_height), font, font_scale, font_color, font_thickness, cv2.LINE_AA)

    # plt - RGB , cv2 - BGR
    return cv2.cvtColor(img_rgb, cv2.COLOR_RGB2BGR)

def process_images(input_folder, output_folder, check):
```

```
    """
    Process all images in the input folder by detecting and recognizing
traffic signs,
    then save the processed images to the output folder.

    Parameters:
        input_folder (str): Path to the folder containing the input
images.
        output_folder (str): Path to the folder where processed images
will be saved.
        check (str): Sub-folder name to store processed images.

    Returns:
        None
    """
    # Ensure output folder exists
    ver2_folder = os.path.join(output_folder, check)
    if not os.path.exists(ver2_folder):
        os.makedirs(ver2_folder)

    # List all images in the input folder
    for filename in os.listdir(input_folder):
        if filename.endswith('.jpg') or filename.endswith('.png'):  #
Add other image formats if needed
            print(filename)
            image_path = os.path.join(input_folder, filename)
            image = cv2.imread(image_path)
            if image is not None:
                processed_image = reviewSign(image)
                output_path = os.path.join(ver2_folder, filename)
                cv2.imwrite(output_path, processed_image)
                print(f"Processed and saved {filename} to
{ver2_folder}")
```

```
                print("--------------------------------")


if __name__ == "__main__":
    # Define paths
    current_folder = os.getcwd()
    input_folder = os.path.join(current_folder, 'NewData')
    check = "ver2"
    # Process images
    process_images(input_folder, current_folder, check)
```

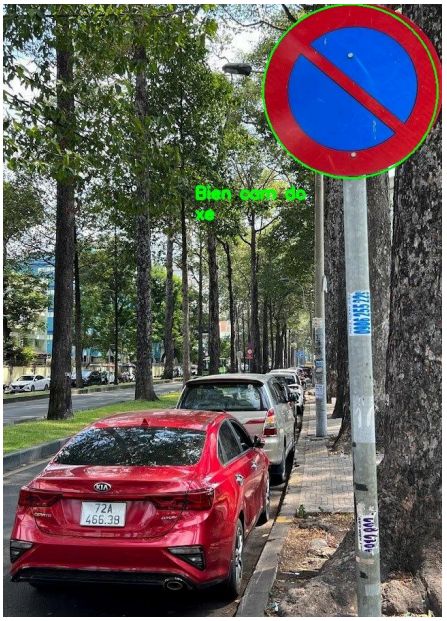
2.2 Môi trường cần để thực thi đoạn mã

- Môi trường thực thi: Google Colab, Local
- Ngôn ngữ lập trình: Python

CHƯƠNG 3. KẾT QUẢ

Trong chương này, chúng em trình bày kết quả thu được từ quá trình xử lý và nhận dạng biển báo cấm. Các biển báo này thường có hình dạng tròn và nội dung bên trong mô tả các quy định về giao thông. Sử dụng phương pháp phát hiện hình tròn và nhận dạng vùng quan tâm (ROI), hệ thống đã phân tích và dự đoán thành công nội dung của các biển báo trên 15 bức ảnh đầu vào. Kết quả cho ra là 15 ảnh đã được tìm ra vị trí biển cấm và nội dung của các biển.


Biển cấm
TUYẾN PHỐ ĐI BỘ
CẤM
CÁC PHƯƠNG TIỆN HOẠT ĐỘNG


Biển cấm đỗ xe


50
Biển báo tốc độ tối đa cho phép
50 km/h


Biển cấm rẽ trái


40
2
16:30 - 18:30
2. Bien cam do xe


1
2


2023
06: 00 - 18 : 00
TUYẾN ĐƯỜNG CÓ
GIÁM SÁT, XỬ PHẠT
BẰNG CAMERA


TAXI
6:00 - 21:00
XE HỢP ĐỒNG
DƯỚI 09 CHỖ NGỒI
6:00 - 21:00


1
40
2. Bien toc do toi da cho phep 40 km/h
1. Bien cam nguoi di bo

Hình - Kết quả

TÀI LIỆU THAM KHẢO

OpenCV Documentation: https://docs.opencv.org/