

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH  
KHOA CÔNG NGHỆ THÔNG TIN



**TIỂU LUẬN CUỐI KỲ**

**Môn học: XỬ LÝ ẢNH**

**Tên tiểu luận: CUỐI KỲ MÔN XỬ LÝ ẢNH**

# ỨNG DỤNG XỬ LÝ ẢNH VÀO PHÁT HIỆN LÀN ĐƯỜNG VÀ ĐÁNH LÁI

Giảng viên: PGS.TS. Hoàng Văn Dũng

**Danh sách sinh viên thực hiện**

<b>Mã số SV</b>	<b>Họ và tên</b>
20110043	ĐỊNH QUÂN
20110526	ĐỖ THỊ BÍCH NGỌC
20110408	LÊ CÔNG TRÌNH

## Acknowledgment

Nhóm em xin chân thành cảm ơn thầy Hoàng Văn Dũng, người phụ trách dự án của nhóm em, đã hướng dẫn nhóm em trong quá trình thực hiện dự án. Thầy đã đưa ra những đề xuất và hướng dẫn quý giá để hoàn thành dự án, giúp nhóm em hiểu rõ các vấn đề phức tạp liên quan đến việc làm dự án và trình bày nó một cách hiệu quả. Những vấn đề phức tạp này sẽ không thể giải quyết nếu không có sự hướng dẫn của thầy. Dự án của nhóm em đã thành công chỉ nhờ có sự hướng dẫn của thầy.

Dự án được thực hiện trong vòng năm tuần, đủ để hoàn thành. Tuy nhiên, do có quá nhiều kiến thức mới và thời gian mỗi tuần không đủ tối ưu, dự án của nhóm em sẽ có nhiều lỗi, điều không tránh khỏi. Nhóm em rất mong nhận được tất cả những ý kiến của giáo viên để giúp kiến thức hạn chế của nhóm em được cải thiện tốt hơn. Chân thành cảm ơn.

## Lời mở đầu

Mục đích và mục tiêu của khóa đào tạo này chủ yếu là cho thời điểm này, và với khóa đào tạo này, nhóm em đã có một số tự tin khi giới thiệu ứng dụng. Nhóm em cũng tin rằng nhóm em đã thu được một số kiến thức IT, và nếu nhóm em luyện tập nhiều và có một chút chuyên môn trong lĩnh vực này, thì nhóm em sẽ có thể tồn tại thông minh trong môi trường cạnh tranh ngày nay.

Nỗ lực viết báo cáo là một phần để hoàn thành khóa học. Trong báo cáo, nhóm em đã cố gắng đại diện cho toàn bộ nội dung mà nhóm em đã học trong chương trình một cách có hệ thống và có thể trình bày được. Nhóm em chia từng chủ đề thành một chương độc lập để phản ánh toàn bộ chủ đề một cách rõ ràng và rõ ràng hơn. Trong tài liệu tham khảo, nhóm em đã sử dụng phương pháp trích dẫn trong toàn bộ báo cáo. Cuối cùng, nhóm em rất hy vọng cấu trúc và chủ đề của báo cáo sẽ là tài liệu hữu ích cho tất cả độc giả, đặc biệt là người dùng.

## MỤC LỤC

Acknowledgment.....	2
Lời mở đầu .....	3
I. Mô tả dự án .....	6
1. Mục tiêu.....	6
2. Phạm vi và đối tượng.....	6
II. Cơ sở lý thuyết.....	6
Cơ sở lý thuyết dùng để thực hiện project lane detection bao gồm các nội dung sau	6
1. Công cụ và môi trường để lập trình: .....	6
2. Thư viện hỗ trợ lập trình:.....	7
3. Phương pháp và kỹ thuật được sử dụng: .....	7
4. Đối tượng sử dụng ứng dụng: .....	7
III. Phân tích, thiết kế giải pháp .....	8
1. Mô tả quy trình .....	8
1.1. Filter_colors.....	9
1.2. Grayscale .....	10
1.3. Gaussian_blur .....	10
1.4. Canny .....	10
1.5. Region_of_interest.....	11
1.6. Hough_lines .....	12
1.7. Draw_lines .....	13
1.8. Weighted_img.....	15
1.9. Annotate_image_array .....	16
1.10. Annotate_video .....	17
1.11. GUI .....	18

2. Giao diện người dùng đồ họa .....	21
IV. Test cases .....	23
V. Tổng kết .....	24
1. Sinh Viên tự đánh giá .....	24
2. Khó khăn.....	24
3. Lợi ích.....	24
4. Bất lợi .....	24
5. Ý tưởng phát triển.....	24
References .....	25

# I. Mô tả dự án

## 1. Mục tiêu

Lane detection là một trong những công nghệ quan trọng trong lĩnh vực xe tự lái và hệ thống giám sát lái xe. Mục tiêu chính của lane detection là xác định vị trí của các làn đường trên đường để giúp cho hệ thống xe tự lái hoặc hệ thống giám sát lái xe có thể điều khiển xe một cách an toàn và chính xác hơn.

Các thuật toán lane detection sử dụng các kỹ thuật xử lý ảnh và máy học để phân tích hình ảnh hoặc video và xác định vị trí của các làn đường. Các thông tin đó sẽ được truyền đến hệ thống xe tự lái hoặc hệ thống giám sát lái xe để giúp cho hệ thống có thể đưa ra quyết định và hành động phù hợp.

Mục tiêu của lane detection không chỉ là giúp cho hệ thống xe tự lái và hệ thống giám sát lái xe hoạt động an toàn hơn, mà còn giúp tài xế lái xe phát hiện và tránh được các nguy hiểm trên đường như lái xe vượt đèn đỏ, lái xe lấn làn, và xử lý các tình huống khẩn cấp khác. Điều này giúp cải thiện đáng kể độ an toàn và tiện ích trong việc lái xe và giảm nguy cơ tai nạn giao thông..

## 2. Phạm vi và đối tượng

Phạm vi của ứng dụng lane detection là phát hiện và vẽ các đường làn trên một video đầu vào. Ứng dụng này sử dụng các kỹ thuật xử lý ảnh để phát hiện các đường làn và vẽ chúng lên video đầu vào để tạo ra một video mới được chú thích với các đường làn đã được vẽ lên.

Đối tượng của ứng dụng này là tài xế và những người sử dụng đường. Khi lái xe, các đường làn có vai trò quan trọng để giữ cho xe đi đúng hướng và tránh va chạm với các xe khác. Ứng dụng lane detection này giúp tài xế và những người sử dụng đường có thể nhận biết các đường làn một cách dễ dàng và an toàn hơn, từ đó giúp giảm nguy cơ xảy ra tai nạn giao thông.

# II. Cơ sở lý thuyết

Cơ sở lý thuyết dùng để thực hiện project lane detection bao gồm các nội dung sau

## 1. Công cụ và môi trường để lập trình:

Ngôn ngữ lập trình: Python

Trình biên dịch: không cần do Python là ngôn ngữ thông dịch

Môi trường lập trình: PyCharm hoặc một trình soạn thảo khác

## **2. Thư viện hỗ trợ lập trình:**

OpenCV: thư viện mã nguồn mở được sử dụng để xử lý ảnh và video.

MoviePy: thư viện Python được sử dụng để xử lý video và âm thanh.

tkinter: thư viện Python được sử dụng để tạo giao diện người dùng.

## **3. Phương pháp và kỹ thuật được sử dụng:**

Xử lý ảnh để phát hiện và vẽ các đường làn trên video đầu vào. Các bước xử lý ảnh bao gồm giữ lại các pixel màu trắng và vàng để chỉ tập trung vào các đường làn, chuyển đổi ảnh thành grayscale, áp dụng Gaussian smoothing và Canny edge detection để phát hiện các đường làn trên ảnh, và region of interest để giới hạn phạm vi xử lý.

Sử dụng hàm `weighted_img` để kết hợp ảnh gốc và ảnh với các đường làn đã được vẽ lên để tạo ra một video mới được chú thích với các đường làn đã được vẽ lên.

Sử dụng hàm `annotate_video` để tạo ra một video mới được chú thích với các đường làn đã được vẽ lên trên mỗi khung hình của video đầu vào.

Sử dụng OpenCV để đọc và hiển thị video mới được tạo ra.

Sử dụng phương pháp region of interest để giới hạn phạm vi xử lý, giúp tối ưu hóa thời gian xử lý và giảm thiểu các sai sót không cần thiết.

## **4. Đối tượng sử dụng ứng dụng:**

Tài xế và những người sử dụng đường, giúp họ có thể nhận biết các đường làn một cách dễ dàng và an toàn hơn khi lái xe trên đường.

Tóm lại, để thực hiện ứng dụng lane detection, chúng ta sử dụng ngôn ngữ lập trình Python và các thư viện hỗ trợ xử lý ảnh và video như OpenCV và MoviePy. Chúng ta sử dụng các phương pháp xử lý ảnh để phát hiện và vẽ các đường làn, phương pháp region of interest để giới hạn phạm vi xử lý, và sử dụng hàm `weighted_img` để kết hợp ảnh gốc và ảnh với các đường làn đã được vẽ lên. Đối tượng sử dụng ứng dụng là tài xế và những người sử dụng đường, giúp họ có thể lái xe an toàn hơn trên đường.

### III. Phân tích, thiết kế giải pháp

#### 1. Mô tả quy trình

Đoạn mã này là một ứng dụng lane detection được viết bằng Python và sử dụng thư viện OpenCV. Ứng dụng này cho phép người dùng chọn tệp video đầu vào, chọn vị trí và đặt tên cho tệp video đầu ra và sau đó sử dụng hàm `annotate_video` để tạo ra một video mới được chú thích với các đường làn đã được vẽ lên.

Cụ thể, ứng dụng bao gồm các thành phần chính sau:

- Hàm `weighted_img`: hàm này được sử dụng để kết hợp ảnh gốc và ảnh với các đường làn đã được vẽ lên.
- Hàm `annotate_image_array`: hàm này được sử dụng để tạo ra ảnh được chú thích với các đường làn đã được vẽ lên.
- Hàm `annotate_video`: hàm này được sử dụng để tạo ra một video mới được chú thích với các đường làn đã được vẽ lên trên mỗi khung hình của video đầu vào.
- Giao diện người dùng: ứng dụng này sử dụng thư viện `tkinter` để tạo giao diện người dùng. Nó cho phép người dùng chọn tệp video đầu vào, chọn vị trí và đặt tên cho tệp video đầu ra và sau đó sử dụng hàm `annotate_video` để tạo ra một video mới được chú thích với các đường làn đã được vẽ lên.

Ứng dụng này hoạt động bằng cách sử dụng một loạt các phép xử lý ảnh để phát hiện và vẽ các đường làn trên video đầu vào. Đầu tiên, các pixel màu trắng và vàng được giữ lại trên ảnh để chỉ tập trung vào các đường làn. Sau đó, các bước xử lý ảnh được áp dụng như `grayscale`, `Gaussian smoothing`, `Canny edge detection` và `region of interest` để phát hiện các đường làn trên ảnh. Cuối cùng, hàm sử dụng hàm `weighted_img` để kết hợp ảnh gốc và ảnh với các đường làn đã được vẽ lên để tạo ra một video mới được chú thích với các đường làn đã được vẽ lên trên mỗi khung hình của video đầu vào.

Sau khi người dùng đã chọn tệp đầu vào và đầu ra và nhấp vào nút "Annotate", hàm `annotate_video` được gọi để tạo ra một video mới với các đường làn đã được vẽ lên. Sau đó, video mới này được đọc và hiển thị sử dụng OpenCV.



## 1.1. Filter\_colors

Hàm `filter_colors` được sử dụng trong lane detection để lọc các pixel màu trên ảnh và chỉ giữ lại các pixel màu vàng và trắng. Hàm này giúp tăng độ chính xác của việc phát hiện đường làn bằng cách loại bỏ các pixel màu khác không liên quan.

Cụ thể, hàm `filter_colors` sử dụng phương pháp `inRange()` của OpenCV để tạo các mask cho các pixel màu trắng và vàng trên ảnh. Đối với các pixel màu trắng, ngưỡng được đặt tại giá trị 200 và tất cả các pixel có giá trị màu cao hơn ngưỡng này sẽ được giữ lại. Đối với các pixel màu vàng, hàm sử dụng không gian màu HSV để xác định ngưỡng màu và chỉ giữ lại các pixel nằm trong phạm vi ngưỡng đó.

Sau đó, hàm `filter_colors` sử dụng bit-wise and để kết hợp hai mask trên và chỉ giữ lại các pixel vừa là pixel màu trắng và vừa là pixel màu vàng. Cuối cùng, hàm sử dụng `addWeighted()` để kết hợp hai ảnh kết quả từ mask trên lại với nhau với hệ số trọng số bằng nhau.

Việc lọc các pixel màu không liên quan trên ảnh giúp tăng độ chính xác của việc phát hiện đường làn bằng cách loại bỏ các yếu tố nhiễu trên ảnh, đồng thời giúp tăng độ tương phản của các đường làn so với các yếu tố màu khác trên ảnh..

```
def filter_colors(image):
    # Filter white pixels
    white_threshold = 200 #130
    lower_white = np.array([white_threshold, white_threshold,
white_threshold])
    upper_white = np.array([255, 255, 255])
    white_mask = cv2.inRange(image, lower_white, upper_white)
    white_image = cv2.bitwise_and(image, image, mask=white_mask)

    # Filter yellow pixels
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    lower_yellow = np.array([90,100,100])
    upper_yellow = np.array([110,255,255])
    yellow_mask = cv2.inRange(hsv, lower_yellow, upper_yellow)
    yellow_image = cv2.bitwise_and(image, image, mask=yellow_mask)

    # Combine the two above images
    image2 = cv2.addWeighted(white_image, 1., yellow_image, 1., 0.)

    return image2
```

## 1.2. Grayscale

Hàm grayscale được sử dụng trong lane detection để chuyển đổi ảnh màu đầu vào thành ảnh đen trắng. Chuyển đổi này giúp giảm số lượng thông tin trong ảnh và tăng độ chính xác của việc phát hiện đường làn.

Cụ thể, hàm grayscale sử dụng hàm `cv2.cvtColor` của OpenCV để chuyển đổi ảnh màu đầu vào sang ảnh đen trắng. Chuyển đổi được thực hiện bằng cách tính trung bình cộng của các giá trị màu trong mỗi pixel trên ảnh màu và sử dụng giá trị này làm giá trị cho pixel trên ảnh đen trắng tương ứng.

Việc chuyển đổi ảnh màu sang ảnh đen trắng giúp loại bỏ thông tin màu sắc không cần thiết trên ảnh và chỉ giữ lại thông tin về độ sáng tại mỗi điểm trên ảnh. Điều này giúp giảm nhiễu và tăng độ chính xác của việc phát hiện đường làn trên ảnh.

```
def grayscale(img):  
    return cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

## 1.3. Gaussian\_blur

Hàm `gaussian_blur` được sử dụng trong lane detection để làm mờ ảnh đầu vào trước khi áp dụng các thuật toán phát hiện đường làn. Điều này giúp loại bỏ nhiễu trên ảnh và giảm sự ảnh hưởng của các chi tiết nhỏ không quan trọng đến quá trình phát hiện đường làn.

Cụ thể, hàm `gaussian_blur` sử dụng hàm `cv2.GaussianBlur` của OpenCV để áp dụng bộ lọc Gaussian cho ảnh đầu vào. Bộ lọc Gaussian là một bộ lọc tuyến tính được sử dụng để làm mờ ảnh bằng cách tính toán trung bình trọng số của các pixel xung quanh pixel đang xét. Kích thước của bộ lọc được xác định bởi biến `kernel_size`, và giá trị của `kernel_size` càng lớn thì độ mờ của ảnh sẽ càng cao.

Việc áp dụng bộ lọc Gaussian giúp làm mờ các chi tiết không quan trọng trên ảnh và giảm nhiễu. Điều này giúp tăng độ chính xác của việc phát hiện đường làn trên ảnh, đồng thời giảm thiểu khả năng xảy ra các sai sót do nhiễu trên ảnh.

```
def gaussian_blur(img, kernel_size):  
    return cv2.GaussianBlur(img, (kernel_size, kernel_size), 0)
```

## 1.4. Canny

Hàm `canny` được sử dụng trong lane detection để phát hiện cạnh trên ảnh đầu vào. Việc phát hiện cạnh giúp xác định các đường viền của vật thể trên ảnh, trong trường hợp này là các đường làn đường.

Cụ thể, hàm `canny` sử dụng hàm `cv2.Canny` của OpenCV để tạo ra ảnh chỉ chứa các cạnh trên ảnh ban đầu. Thuật toán Canny được sử dụng để xác định cạnh trên ảnh bằng cách tìm các điểm có độ dốc lớn trên gradient của ảnh. Việc xác định cạnh trên ảnh giúp tách các đường viền của vật thể trên ảnh, giúp tăng độ chính xác của việc phát hiện đường lằn và giảm thiểu khả năng xảy ra các sai sót do các chi tiết không quan trọng trên ảnh.

Hàm `canny` có hai tham số đầu vào là `low_threshold` và `high_threshold`. Các tham số này được sử dụng để xác định ngưỡng dưới và ngưỡng trên cho việc phát hiện cạnh trên ảnh. Các pixel có giá trị gradient lớn hơn ngưỡng trên sẽ được coi là cạnh, trong khi các pixel có giá trị gradient nhỏ hơn ngưỡng dưới sẽ không được coi là cạnh. Các pixel có giá trị gradient nằm giữa ngưỡng dưới và ngưỡng trên sẽ được xem xét là cạnh nếu chúng kết nối với các pixel có giá trị gradient lớn hơn ngưỡng trên.

```
def canny(img, low_threshold, high_threshold):  
    return cv2.Canny(img, low_threshold, high_threshold)
```

### 1.5. Region\_of\_interest

Hàm `region_of_interest` được sử dụng trong lane detection để giới hạn vùng quan tâm trên ảnh, chỉ giữ lại các pixel trong vùng này để phát hiện đường lằn.

Cụ thể, hàm `region_of_interest` sử dụng hàm `cv2.fillPoly` của OpenCV để tạo ra một mask chỉ giữ lại các pixel nằm trong vùng được xác định bởi các đỉnh của một đa giác (polygon) được đưa vào qua biến `vertices`. Sau đó, hàm sử dụng `bitwise_and` để chỉ giữ lại các pixel nằm trong vùng mask trên ảnh đầu vào.

Các tham số đầu vào của hàm `region_of_interest` bao gồm:

- `img`: ảnh đầu vào cần giới hạn vùng quan tâm.
- `vertices`: một danh sách các đỉnh của đa giác xác định vùng quan tâm trên ảnh.

Hàm trả về một ảnh mới chỉ chứa các pixel nằm trong vùng quan tâm đã được xác định bởi các đỉnh của đa giác. Các pixel nằm ngoài vùng này sẽ được đặt giá trị là 0 (đen).

```
def region_of_interest(img, vertices):  
    #defining a blank mask to start with  
    mask = np.zeros_like(img)  
  
    #defining a 3 channel or 1 channel color to fill the mask with depending  
    on the input image  
    if len(img.shape) > 2:  
        channel_count = img.shape[2] # i.e. 3 or 4 depending on your image  
        ignore_mask_color = (255,) * channel_count  
    else:  
        ignore_mask_color = 255
```

```

        #filling pixels inside the polygon defined by "vertices" with the fill
        color
        cv2.fillPoly(mask, vertices, ignore_mask_color)

    #returning the image only where mask pixels are nonzero
    masked_image = cv2.bitwise_and(img, mask)
    return masked_image

```

## 1.6. Hough\_lines

Hàm `hough_lines` được sử dụng trong lane detection để phát hiện các đường thẳng trên ảnh. Hàm sử dụng phương pháp Hough transform để tìm ra các đường thẳng trên ảnh.

Cụ thể, hàm `hough_lines` sử dụng hàm `cv2.HoughLinesP` của OpenCV để tìm ra các đường thẳng trên ảnh. Hàm `cv2.HoughLinesP` sử dụng phương pháp Hough transform để tìm ra các đường thẳng trên ảnh. Trong quá trình này, mỗi điểm trên ảnh được chuyển đổi thành một đường thẳng trên không gian rho-theta. Các đường thẳng này được tìm ra bằng cách tìm kiếm các điểm giao nhau của các đường thẳng trên không gian rho-theta. Các đường thẳng được xác định bởi các tham số rho và theta.

Hàm `hough_lines` có các tham số đầu vào như sau:

- `img`: ảnh đầu vào đã được xử lý bằng phương pháp Canny để phát hiện cạnh.
- `rho`: độ phân giải của không gian rho.
- `theta`: độ phân giải của không gian theta.
- `threshold`: ngưỡng số lượng điểm để xác định một đường thẳng.
- `min_line_len`: độ dài tối thiểu của đường thẳng để được xác định.
- `max_line_gap`: khoảng cách tối đa giữa các đoạn thẳng để được kết hợp lại thành một đường thẳng.

Hàm trả về một ảnh mới với các đường thẳng đã được vẽ lên đó.

```

def hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap):
    lines = cv2.HoughLinesP(img, rho, theta, threshold, np.array([]),
minLineLength=min_line_len, maxLineGap=max_line_gap)
    line_img = np.zeros((*img.shape, 3), dtype=np.uint8) # 3-channel RGB
image
    draw_lines(line_img, lines)
    return line_img

```

## 1.7. Draw\_lines

Hàm `draw_lines` được sử dụng trong lane detection để vẽ đường làn trên ảnh đầu vào. Hàm sử dụng kết quả phát hiện các đường thẳng của phương pháp Hough transform để tìm ra các đường làn trên ảnh.

Cụ thể, hàm `draw_lines` nhận đầu vào là ảnh đã được xử lý bằng phương pháp Hough transform để phát hiện các đường thẳng trên ảnh. Hàm sẽ tìm kiếm các đường thẳng có độ dốc phù hợp để xác định các đường làn riêng biệt bằng cách tính toán độ dốc của các đường thẳng và so sánh với một ngưỡng độ dốc được xác định trước (`slope_threshold`). Sau đó, hàm sẽ sử dụng phương pháp linear regression để tìm ra các đường thẳng tốt nhất để đại diện cho đường làn bên phải và bên trái.

Các tham số đầu vào của hàm `draw_lines` bao gồm:

- `img`: ảnh đầu vào cần vẽ đường làn.
- `lines`: danh sách các đường thẳng được phát hiện bởi Hough transform.
- `color`: màu sắc của đường làn được vẽ.
- `thickness`: độ dày của đường được vẽ.

Hàm sẽ trả về một ảnh mới với các đường làn đã được vẽ lên đó.

```
def draw_lines(img, lines, color=[255, 0, 0], thickness=10):
    if lines is None:
        return
    if len(lines) == 0:
        return
    draw_right = True
    draw_left = True

    # But only care about lines where abs(slope) > slope_threshold
    slope_threshold = 0.5
    slopes = []
    new_lines = []
    for line in lines:
        x1, y1, x2, y2 = line[0] # line = [[x1, y1, x2, y2]]

        # Calculate slope
        if x2 - x1 == 0.: # corner case, avoiding division by 0
            slope = 999. # practically infinite slope
        else:
            slope = (y2 - y1) / (x2 - x1)

    # Filter lines based on slope
```

```

        if abs(slope) > slope_threshold:
            slopes.append(slope)
            new_lines.append(line)

    lines = new_lines

    # Split lines into right_lines and left_lines, representing the right
    and left lane lines
    # Right/left lane lines must have positive/negative slope, and be on the
    right/left half of the image
    right_lines = []
    left_lines = []
    for i, line in enumerate(lines):
        x1, y1, x2, y2 = line[0]
        img_x_center = img.shape[1] / 2 # x coordinate of center of image
        if slopes[i] > 0 and x1 > img_x_center and x2 > img_x_center:
            right_lines.append(line)
        elif slopes[i] < 0 and x1 < img_x_center and x2 < img_x_center:
            left_lines.append(line)

    # Run linear regression to find best fit line for right and left lane
    lines
    # Right lane lines
    right_lines_x = []
    right_lines_y = []

    for line in right_lines:
        x1, y1, x2, y2 = line[0]

        right_lines_x.append(x1)
        right_lines_x.append(x2)

        right_lines_y.append(y1)
        right_lines_y.append(y2)

    if len(right_lines_x) > 0:
        right_m, right_b = np.polyfit(right_lines_x, right_lines_y, 1) # y
= m*x + b
    else:
        right_m, right_b = 1, 1
        draw_right = False

    # Left lane lines
    left_lines_x = []
    left_lines_y = []

    for line in left_lines:
        x1, y1, x2, y2 = line[0]

```

```

        left_lines_x.append(x1)
        left_lines_x.append(x2)

        left_lines_y.append(y1)
        left_lines_y.append(y2)

    if len(left_lines_x) > 0:
        left_m, left_b = np.polyfit(left_lines_x, left_lines_y, 1) # y =
m*x + b
    else:
        left_m, left_b = 1, 1
        draw_left = False

# Find 2 end points for right and left lines, used for drawing the line
# y = m*x + b --> x = (y - b)/m
y1 = img.shape[0]
y2 = img.shape[0] * (1 - trap_height)

right_x1 = (y1 - right_b) / right_m
right_x2 = (y2 - right_b) / right_m

left_x1 = (y1 - left_b) / left_m
left_x2 = (y2 - left_b) / left_m

# Convert calculated end points from float to int
y1 = int(y1)
y2 = int(y2)
right_x1 = int(right_x1)
right_x2 = int(right_x2)
left_x1 = int(left_x1)
left_x2 = int(left_x2)

# Draw the right and left lines on image
if draw_right:
    cv2.line(img, (right_x1, y1), (right_x2, y2), color, thickness)
if draw_left:
    cv2.line(img, (left_x1, y1), (left_x2, y2), color, thickness)

```

## 1.8. Weighted\_img

Hàm `weighted_img` được sử dụng trong lane detection để kết hợp ảnh gốc và ảnh với các đường làn đã được vẽ lên.

Cụ thể, hàm `weighted_img` nhận đầu vào là ảnh với các đường làn đã được vẽ lên (`img`) và ảnh gốc trước khi xử lý (`initial_img`). Hàm sử dụng phương pháp `addWeighted` của OpenCV để kết hợp ảnh gốc và ảnh với các đường làn đã được vẽ lên. Các tham số  $\alpha$ ,  $\beta$  và  $\lambda$  được sử dụng để điều chỉnh sự đóng góp của các ảnh vào ảnh kết quả.

Các tham số đầu vào của hàm `weighted_img` bao gồm:

- `img`: ảnh với các đường lằn đã được vẽ lên.
- `initial_img`: ảnh gốc trước khi xử lý.
- $\alpha$ : hệ số để điều chỉnh sự đóng góp của ảnh gốc (`initial_img`).
- $\beta$ : hệ số để điều chỉnh sự đóng góp của ảnh với các đường lằn đã được vẽ lên (`img`).
- $\lambda$ : hệ số để điều chỉnh độ lệch của ảnh kết quả.

Hàm trả về ảnh kết quả sau khi đã kết hợp ảnh gốc và ảnh với các đường lằn đã được vẽ lên.

```
def weighted_img(img, initial_img,  $\alpha=0.8$ ,  $\beta=1.$ ,  $\lambda=0.$ ):  
    return cv2.addWeighted(initial_img,  $\alpha$ , img,  $\beta$ ,  $\lambda$ )
```

### 1.9. Annotate\_image\_array

Hàm `annotate_image_array` được sử dụng trong lane detection để tạo ra ảnh được chú thích với các đường lằn đã được vẽ lên.

Cụ thể, hàm `annotate_image_array` nhận đầu vào là một mảng numpy của ảnh và sử dụng một loạt các phép xử lý ảnh để phát hiện và vẽ các đường lằn trên ảnh này. Đầu tiên, hàm sử dụng hàm `filter_colors` để chỉ giữ lại các pixel màu trắng và vàng trên ảnh, loại bỏ các pixel màu khác. Sau đó, hàm sử dụng một loạt các bước xử lý ảnh như grayscale, Gaussian smoothing, Canny edge detection và region of interest để phát hiện các đường lằn trên ảnh. Cuối cùng, hàm sử dụng hàm `weighted_img` để kết hợp ảnh gốc và ảnh với các đường lằn đã được vẽ lên.

Các tham số đầu vào của hàm `annotate_image_array` bao gồm: `image_in`: một mảng numpy của ảnh.

Hàm trả về một mảng numpy của ảnh đã được chú thích với các đường lằn đã được vẽ lên.

```
def annotate_image_array(image_in):  
    """ Given an image Numpy array, return the annotated image as a Numpy array """  
    # Only keep white and yellow pixels in the image, all other pixels become black  
    image = filter_colors(image_in)  
  
    # Read in and grayscale the image  
    gray = grayscale(image)  
  
    # Apply Gaussian smoothing  
    blur_gray = gaussian_blur(gray, kernel_size)  
  
    # Apply Canny Edge Detector
```



```

edges = canny(blur_gray, low_threshold, high_threshold)

# Create masked edges using trapezoid-shaped region-of-interest
imshape = image.shape
vertices = np.array([[
    ((imshape[1] * (1 - trap_bottom_width)) // 2, imshape[0]),\
    ((imshape[1] * (1 - trap_top_width)) // 2, imshape[0] - imshape[0] *
trap_height),\
    (imshape[1] - (imshape[1] * (1 - trap_top_width)) // 2, imshape[0] -
imshape[0] * trap_height),\
    (imshape[1] - (imshape[1] * (1 - trap_bottom_width)) // 2,
imshape[0])]])\
    , dtype=np.int32)
masked_edges = region_of_interest(edges, vertices)

# Run Hough on edge detected image
line_image = hough_lines(masked_edges, rho, theta, threshold,
min_line_length, max_line_gap)

# Draw lane lines on the original image
initial_image = image_in.astype('uint8')
annotated_image = weighted_img(line_image, initial_image)
return annotated_image

```

## 1.10. Annotate\_video

Hàm `annotate_video` được sử dụng trong lane detection để tạo ra một video được chú thích với các đường làn đã được vẽ lên trên mỗi khung hình của video đầu vào.

Cụ thể, hàm `annotate_video` nhận đầu vào là tên tệp video đầu vào và tên tệp video đầu ra. Hàm sử dụng thư viện `MoviePy` để đọc tệp video đầu vào và áp dụng hàm `annotate_image_array` cho mỗi khung hình của video đó để tạo ra một video mới với các đường làn đã được vẽ lên. Sau đó, hàm lưu trữ video mới này vào tệp đầu ra.

Các tham số đầu vào của hàm `annotate_video` bao gồm:

- `input_file`: tên tệp video đầu vào.
- `output_file`: tên tệp video đầu ra.

Hàm không trả về giá trị nào mà chỉ tạo ra một tệp video mới với các đường làn đã được vẽ lên.

```

def annotate_video(input_file, output_file):
    """ Given input_file video, save annotated video to output_file """
    video = VideoFileClip(input_file)
    annotated_video = video.fl_image(annotate_image_array)
    annotated_video.write_videofile(output_file, audio=False)

```

## 1.11. GUI

Tạo giao diện người dùng (GUI) cho ứng dụng lane detection. Ứng dụng này cho phép người dùng chọn tệp video đầu vào, chọn vị trí và đặt tên cho tệp video đầu ra và sau đó sử dụng hàm `annotate_video` để tạo ra một video mới được chú thích với các đường làn đã được vẽ lên.

Cụ thể, đoạn mã sử dụng thư viện `tkinter` để tạo giao diện người dùng. Nó bao gồm một số ứng dụng như:

Tạo các trường nhập liệu và nút chọn tệp để cho phép người dùng chọn tệp video đầu vào và đặt tên cho tệp video đầu ra.

Tạo một nhãn để hiển thị đường dẫn tệp video đầu ra đã chọn.

Tạo một nút để bắt đầu quá trình chú thích video, sử dụng hàm `annotate_video`.

Sau khi người dùng đã chọn tệp đầu vào và đầu ra và nhấp vào nút "Annotate", đoạn mã sẽ gọi hàm `annotate_video` để tạo ra một video mới với các đường làn đã được vẽ lên. Sau đó, đoạn mã sử dụng `OpenCV` để đọc và hiển thị video mới này.

Các biến được sử dụng trong đoạn mã này bao gồm:

- `input_file`: một biến `StringVar` để lưu trữ đường dẫn tệp video đầu vào.
- `output_file`: một biến `StringVar` để lưu trữ đường dẫn tệp video đầu ra.
- `image_only`: một biến `BooleanVar` để chỉ định liệu chương trình có nên chú thích ảnh duy nhất hay tạo video chú thích với các đường làn đã được vẽ lên.

Đoạn mã này được chạy trong một vòng lặp vô hạn để hiển thị video mới, và sử dụng phím "q" để thoát khỏi vòng lặp.

```
# Create the GUI
root = tk.Tk()
root.title("ỨNG DỤNG XỬ LÝ ẢNH VÀO PHÁT HIỆN LÀN ĐƯỜNG VÀ ĐÁNH LÁI")

# Get the screen size
screen_width = 100
screen_height = 100

# Define variables to store file paths
input_file_path = tk.StringVar()
output_file_path = tk.StringVar()

# Define a function to select the input file
def select_input_file():
    file_types = [("MP4 files", "*.mp4")]
    input_file_path.set(filedialog.askopenfilename(filetypes=file_types))
```

```

# Define a function to select the output file
def select_output_file():
    output_file_path.set(filedialog.asksaveasfilename(defaultextension=".mp4"))

# Define a function to process the video
def process_video():
    # Get the input and output file paths
    input_path = input_file_path.get()
    output_path = output_file_path.get()
    annotate_video(input_path, output_path)
    # Open the video capture
    cap = cv2.VideoCapture(output_path)

    # Process each frame of the video
    while True:
        # Read the next frame
        ret, frame = cap.read()

        # If there are no more frames, break out of the loop
        if not ret:
            break
        # Process the frame (replace this with your own video processing
code)
        processed_frame = frame

        # Display the processed frame on the GUI
        img = PIL.Image.fromarray(processed_frame)
        img_rgb = img.convert('RGB')
        img_tk = PIL.ImageTk.PhotoImage(img_rgb, gamma=1.0)
        video_label.img_tk = img_tk
        video_label.configure(image=img_tk)
        video_label.update()

    # Release the video capture and writer
    cap.release()

# Create a label for the input file path
input_label = tk.Label(root, text="Input File Path:")
input_label.grid(row=1, column=0)

# Create an entry for the input file path
input_entry = tk.Entry(root, textvariable=input_file_path)
input_entry.grid(row=1, column=1)

```

```

# Create a button to select the input file
input_button = tk.Button(root, text="Select Input File",
command=select_input_file)
input_button.grid(row=1, column=2)

# Create a label for the output file path
output_label = tk.Label(root, text="Output File Path:")
output_label.grid(row=2, column=0)

# Create an entry for the output file path
output_entry = tk.Entry(root, textvariable=output_file_path)
output_entry.grid(row=2, column=1)

# Create a button to select the output file
output_button = tk.Button(root, text="Select Output File",
command=select_output_file)
output_button.grid(row=2, column=2)

#tao label chứa Họ Tên MSSV tên
name_label4 =tk.Label(root,text="      Họ và Tên, MSSV THỰC HIỆN ĐỀ
TÀI",fg="RED",font=("Arial", 12))
name_label4.grid(row=1,column= 5)
name_label11 =tk.Label(root,text="      ĐỊNH QUÂN
20110043",fg="RED",font=("Arial", 12))
name_label11.grid(row=2,column= 5)
name_label12 =tk.Label(root,text="      LÊ CÔNG TRÌNH
20110408",fg="RED",font=("Arial", 12))
name_label12.grid(row=3,column= 5)
name_label13 =tk.Label(root,text="      ĐỖ THỊ BÍCH NGỌC
20110526",fg="RED",font=("Arial", 12))
name_label13.grid(row=4,column= 5)
#TẠO LABEL CHO LOGO
img = Image.open("C://Users//pc//Documents//TaiLieuMonHoc//XLA//logo.png")
img = img.resize((300, 300), resample=Image.BILINEAR)
photo = ImageTk.PhotoImage(img)

# Tạo đối tượng Label để hiển thị hình ảnh
label = tk.Label(root, image=photo, width=300, height=300)
label.grid(row=5, column=1)
# Tạo một khung cho đầu ra video
video_frame = tk.Frame(root)
video_frame.grid(row=5, column=1, columnspan=3)

# Tạo một nhãn cho đầu ra video
video_label = tk.Label(video_frame)
video_label.grid(row=5, column=1)

# Create a button to process the video

```


```

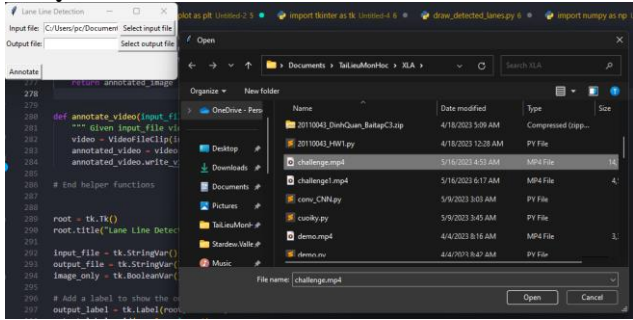
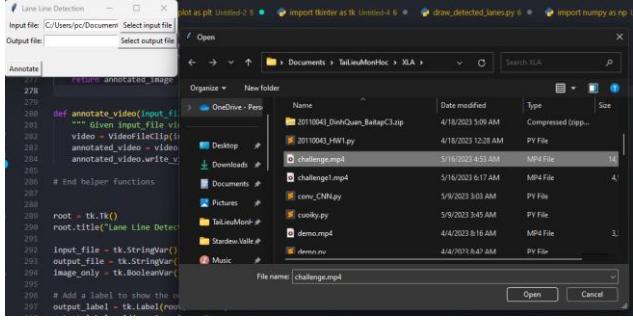
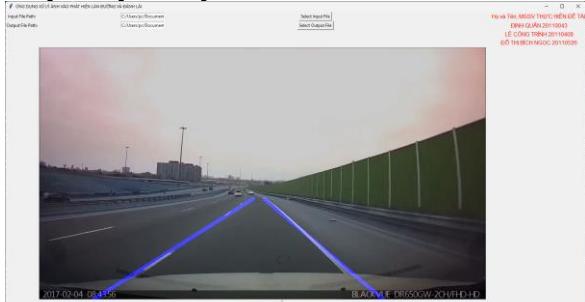
process_button = tk.Button(root, text="Process Video",
command=process_video,width=15, height=2)
process_button.grid(row=6, column=1, columnspan=3)

# Start the GUI main loop
root.mainloop()

```

## 2. Giao diện người dùng đồ họa

No.	GUI	Mục đích	Brief Explanation
1	 <p><b>Main window</b></p>	Giao diện chính của ứng dụng	Ở bên phải ngoài cùng có 2 nút để người dùng chọn video đầu vào và đầu ra sau khi đã vẽ làn đường lên. Dưới cùng có nút “Annoate” để thực hiện.

2	<p><b>Chọn file đầu vào</b></p> 	Người dùng chọn file video đầu vào	Khi người dùng ấn vào nút “Select input file” thì một cửa sổ hiện lên cho phép người dùng chọn video đầu vào
3	<p><b>Chọn file đầu ra</b></p> 	Người dùng chọn nơi chứa file đầu ra và đặt tên cho file đầu ra	Khi người dùng ấn vào nút “Select output file” thì một cửa sổ hiện lên cho phép người dùng chọn nơi chứa file đầu ra và đặt tên cho file đầu ra đầu vào
4	<p><b>Hiện thị kết quả</b></p> 	Hiện thị kết quả sau khi lane-detection cho người dùng xem	Hiện thị kết quả sau khi lane-detection cho người dùng xem

## IV. Test cases

No.	Test cases	Purpose
1	<b>Test case 1:</b> Input: Đầu vào là 1 video xe chạy trên đường vắng Result: Nhận được video xe nhận biết được làn đường đang chạy	Kiểm tra phần mềm có hoạt động bình thường không
2	<b>Test case 2:</b> Input: Đầu vào là 1 xe chạy trên đường đông đúc Result: Nhận được video xe nhận biết được làn đường đang chạy	Kiểm tra xe có nhận biết được làn đường ở thành phố không

## V. Tổng kết

### 1. Sinh Viên tự đánh giá

- Đối với sinh viên mới học lập trình và xử lý ảnh, ứng dụng lane detection có thể là một thử thách khó khăn, nhưng cũng là một cơ hội để họ học thêm về các kỹ thuật xử lý ảnh và video.
- Sinh viên có kinh nghiệm về lập trình và xử lý ảnh có thể tìm thấy ứng dụng này thú vị và hữu ích để nâng cao kỹ năng của mình.

### 2. Khó khăn

- Việc xử lý ảnh và video có thể tốn nhiều thời gian và tài nguyên máy tính, đặc biệt là khi xử lý các video có độ phân giải cao.
- Việc phát hiện và vẽ các đường làn có thể bị ảnh hưởng bởi các yếu tố như ánh sáng, điều kiện thời tiết hoặc các vật cản trên đường..

### 3. Lợi ích

- Ứng dụng lane detection giúp tài xế và những người sử dụng đường có thể nhận biết các đường làn một cách dễ dàng và an toàn hơn, từ đó giúp giảm nguy cơ xảy ra tai nạn giao thông.
- Ứng dụng có thể được sử dụng để phát triển các hệ thống hỗ trợ lái xe tự động trong tương lai..

### 4. Bất lợi

- Việc sử dụng ứng dụng lane detection để thay thế cho quan sát và quyết định của tài xế là không khả thi, do đó việc sử dụng ứng dụng này chỉ nên là một công cụ hỗ trợ cho tài xế và những người sử dụng đường.
- Việc sử dụng ứng dụng lane detection có thể gây phụ thuộc và làm giảm khả năng quan sát và quyết định của tài xế..

### 5. Ý tưởng phát triển

- Cải thiện thuật toán phát hiện và vẽ các đường làn để giảm thiểu các sai sót và tăng độ chính xác của ứng dụng.
- Phát triển các tính năng mới như phát hiện vật cản trên đường và hiển thị biển báo giao thông để tăng tính ứng dụng và hữu ích của ứng dụng trong việc hỗ trợ lái xe.



- Tối ưu hóa hiệu suất của ứng dụng để có thể sử dụng được trên các thiết bị có tài nguyên máy tính thấp hơn. Tối ưu hóa hiệu suất của ứng dụng để có thể sử dụng được trên các thiết bị có tài nguyên máy tính thấp hơn.

## References

1. OpenCV documentation: [https://docs.opencv.org/master/d9/df8/tutorial\\_root.html](https://docs.opencv.org/master/d9/df8/tutorial_root.html)
2. MoviePy documentation: <https://zulko.github.io/moviepy/>
3. tkinter documentation: <https://docs.python.org/3/library/tk.html>
4. "Lane Detection with OpenCV and Python" tutorial by Adrian Rosebrock: <https://www.pyimagesearch.com/2017/02/06/faster-video-file-fps-with-cv2-videocapture-and-opencv/>
5. "Lane Detection for Self-Driving Cars" tutorial by Krish Naik: <https://www.youtube.com/watch?v=eLTLtUVuuy4>