

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH**  
**KHOA CƠ KHÍ CHẾ TẠO MÁY**



**BÁO CÁO CUỐI KÌ**  
**MÔN HỌC: THỊ GIÁC MÁY**  
**ĐỀ TÀI:**

**NHẬN DIỆN ĐƯỜNG LANE**  
**TRONG NHIỀU ĐIỀU KIỆN THỜI TIẾT KHÁC NHAU:**  
**NẮNG, MƯA, NGÀY, ĐÊM ...**

**GVHD : Nguyễn Văn Thái**

**SVTH :**

**Lê Duy Huy-21146467**

**Bùi Trung Kiên-21146115**

**Võ Thành Đạt-21146450**

**TP.HCM, tháng 5 năm 2024**

## HỌC KỲ II- NĂM HỌC 2023-2024

Nhóm 03 (Mã lớp: MAVI332529\_23\_2\_02CLC)

Tên đề tài: Nhận diện đường lane trong nhiều điều kiện thời tiết khác nhau,  
nắng, mưa, ngày, đêm,...

STT	STT	MSSV	Tỉ lệ % hoàn thành công việc
1	Lê Duy Huy	21146467	100%
2	Bùi Trung Kiên	21146115	100%
3	Võ Thành Đạt	21146450	100%

### Ghi chú:

Tỷ lệ % = 100%: Mức độ phần trăm của từng sinh viên tham gia

### Nhận xét của giảng viên:

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Ngày tháng năm 2024

GVHD

## LỜI CẢM ƠN

Trong suốt quá trình học tập vừa qua, chúng em xin chân thành gửi lời cảm ơn sâu sắc tới thầy Nguyễn Văn Thái - người thầy hết lòng quan tâm, hỗ trợ và truyền đạt kiến thức, kỹ năng cho chúng em một cách tận tụy. Nhờ những bài giảng đầy tâm huyết và sự hướng dẫn tận tình của thầy, chúng em đã hiểu rõ hơn về kiến thức chuyên ngành, nắm vững hơn các kỹ năng thực hành, qua đó giúp chúng em xác định được đam mê và hướng đi trong tương lai.

Tuy nhiên, do kiến thức và kỹ năng của bản thân còn nhiều hạn chế, nên chắc chắn đồ án của chúng em không thể tránh khỏi những thiếu sót. Vì vậy, chúng em rất mong nhận được những ý kiến đóng góp quý báu từ thầy cùng các bạn, để đồ án có thể được hoàn thiện một cách tốt nhất.

Để kết thúc, một lần nữa chúng em xin gửi lời cảm ơn chân thành và sâu sắc nhất tới thầy - người đã dày công truyền đạt kiến thức, động viên và khích lệ tinh thần chúng em không ngừng phấn đấu, rèn luyện bản thân. Chúng em kính chúc thầy luôn dồi dào sức khỏe, thành công hơn nữa trên con đường hoàn thành sứ mệnh cao cả là trồng người, và tiếp tục đồng hành cùng nhiều thế hệ sinh viên, đóng góp nhiều hơn nữa cho sự nghiệp giáo dục đào tạo của đất nước.

# MỤC LỤC

CHƯƠNG I: TỔNG QUAN .....	1
1. Giới thiệu đề tài.....	1
2. Vấn đề đặt ra .....	1
3. Mục tiêu đặt ra .....	1
4. Phương pháp nghiên cứu .....	1
5. Phạm vi đề tài.....	1
CHƯƠNG II: CƠ SỞ LÝ THUYẾT .....	2
1. Lý thuyết về xử lý ảnh:.....	2
2. Các thư viện sử dụng trong chương trình.....	2
2.1 OpenCV-Xử lý hình:.....	2
2.2 Numpy – Xử lý mảng đa chiều, ma trận .....	2
2.3 Tkinter – Tạo giao diện người dùng (GUI) .....	3
3. Các thuật toán .....	3
3.1 Canny Edge Detection: .....	3
3.2 Hough Line Transform .....	6
3.3 AddWeighted.....	6
3.4 Display lines .....	6
3.5 Average_slope_intercept .....	7
CHƯƠNG III: NỘI DUNG LẬP TRÌNH .....	8
1. Giao diện theo yêu cầu: .....	8
2. Các bước thực hiện và lưu đồ giải thuật .....	8
3. Code .....	10
CHƯƠNG IV: KẾT QUẢ ĐẠT ĐƯỢC .....	16
1. Kết quả:.....	16
2. Hạn chế: .....	16
3. Hướng phát triển: .....	16
TÀI LIỆU THAM KHẢO .....	17

## CHƯƠNG I: TỔNG QUAN

### 1. Giới thiệu đề tài

Line detection-đây là là một chủ đề quan trọng trong lĩnh vực Xử lý Ảnh và Thị giác Máy tính. Mục đích chính là phát triển các thuật toán và phương pháp để phát hiện và nhận dạng các đường thẳng từ dữ liệu ảnh số hoặc video. Phát hiện đường thẳng có nhiều ứng dụng thực tế như lái xe tự động, kiểm tra chất lượng sản phẩm, phân tích cấu trúc, và điều khiển robot. Đây là một đề tài quan trọng với nhiều tiềm năng nghiên cứu và ứng dụng.

### 2. Vấn đề đặt ra

Nhận dạng đường lane đường để giúp cho các tài xế đi đúng làn đường của mình để tránh tình trạng chạy sai làn đường và gây ra những tai nạn xảy ra không đáng có. Đây là vấn đề ưu tiên trong việc điều khiển xe và chấp hành đúng luật giao thông. Do đã có nhiều vụ tai nạn xảy ra đáng tiếc do tài xế chạy sai làn đường của mình trong quá trình di chuyển trên đường. Từ vấn đề trên, chúng em quyết định ứng dụng xử lý ảnh và trí tuệ nhân tạo dùng để có thể nhận dạng lane đường trong các điều kiện thực tế khác nhau.

### 3. Mục tiêu đặt ra

- Sử dụng thành thạo ngôn ngữ lập trình Python
- Áp dụng các lý thuyết từ môn học
- Vận dụng trí tuệ nhân tạo và xử lý ảnh vào việc nhận dạng đường lane đường thực tế

### 4. Phương pháp nghiên cứu

- Vận dụng các kiến thức đã học ở môn Xử Lý Ảnh kết hợp với Trí Tuệ Nhân Tạo (AI)
- Tham khảo các dự án có sẵn của các bài viết trong và ngoài nước để tìm ra phương pháp tối ưu cho dự án của nhóm

### 5. Phạm vi đề tài

- Ứng dụng rộng rãi hơn trong việc phục vụ cho camera hành trình phục vụ cho các tài xế
- Giúp các tài xế chạy xe đúng làn đường để giảm thiểu tối đa những tai nạn không đáng có

## CHƯƠNG II: CƠ SỞ LÝ THUYẾT

### 1. Lý thuyết về xử lý ảnh:

- Xử lý ảnh (digital image processing) hay xử lý ảnh kỹ thuật số là một phân ngành trong xử lý số tín hiệu với tín hiệu xử lý là ảnh. Đây là một phân ngành khoa học mới rất phát triển trong những năm gần đây. Xử lý ảnh là đối tượng nghiên cứu của lĩnh vực thị giác máy, là quá trình biến đổi từ một ảnh ban đầu sang một ảnh mới với các đặc tính và tuân theo ý muốn của người sử dụng. Xử lý ảnh có thể gồm quá trình phân tích, phân lớp các đối tượng, làm tăng chất lượng, phân đoạn và tách cạnh, gán nhãn cho vùng hay quá trình biên dịch các thông tin hình ảnh của ảnh. Cũng như xử lý dữ liệu bằng đồ họa, xử lý ảnh số là một lĩnh vực của tin học ứng dụng. Xử lý dữ liệu bằng đồ họa đề cập đến những ảnh nhân tạo, các ảnh này được xem xét như là một cấu trúc dữ liệu và được tạo bởi các chương trình.
- Ngày nay xử lý ảnh đã được áp dụng rất rộng rãi trong đời sống như: photoshop, nén ảnh, nén video, nhận dạng biển số xe, nhận dạng khuôn mặt, nhận dạng chữ viết, xử lý ảnh thiên văn, ảnh y tế,... Ngoài ra chúng ta có thể kết hợp với AI để có thể phân tích dự đoán hình ảnh để đưa ra những kết quả chúng ta muốn biết. Có thể liệt kê một số phương pháp nhận dạng cơ bản như nhận dạng ảnh của các đối tượng trên ảnh, tách cạnh, phân đoạn hình ảnh

### 2. Các thư viện sử dụng trong chương trình

#### 2.1 OpenCV-Xử lý hình:

- OpenCV là một gói mô-đun hình ảnh lý tưởng cho phép bạn đọc và ghi, thay đổi dữ liệu nhiều hình ảnh cùng một lúc.
- Tạo ra thị giác máy tính cho phép bạn xây dựng lại, gián đoạn và thông hiểu môi trường 3D từ môi trường 2D tương ứng của nó.
- OpenCV được xử dụng nhiều trong nhận diện vật thể và hình ảnh được
- thiết lập trước, chẳng hạn như khuôn mặt, động vật, cây cối, các vật thể di chuyển,..
- Bạn cũng có thể lưu và chụp bất kỳ khoảnh khắc nào của video và cũng có
- thể phân tích các thuộc tính khác nhau của nó như chuyển động, nền,..
- OpenCV tương thích với nhiều hệ điều hành như Windows, OS-X, Open BSD và nhiều hệ điều hành khác

#### 2.2 Numpy – Xử lý mảng đa chiều, ma trận

- Numpy (Numeric Python): là một thư viện toán học phổ biến và mạnh mẽ của Python. Cho phép làm việc hiệu quả với ma trận và mảng, đặc biệt là dữ liệu ma trận và mảng lớn với tốc độ xử lý nhanh hơn nhiều lần khi chỉ sử dụng code Python đơn thuần
- Là một mô-đun mở rộng mã nguồn mở cho Python, cung cấp các chức năng biên dịch nhanh cho các thao tác toán học và số, thậm chí là với những ma trận và mảng có lượng dữ liệu khổng lồ. Bên cạnh đó các mô-

- NumPy cung cấp một thư viện lớn các chức năng toán học cấp cao để hoạt động trên các ma trận và mảng một cách dễ dàng và thuận tiện
- NumPy cung cấp những masked arrays đồng thời với mảng gốc. Nó cũng đi kèm với các chức năng như thao tác với hình dạng logic, biến đổi Fourier rời rạc, đại số tuyến tính tổng quát, và nhiều hơn nữa
- Gói mô-đun này cung cấp các công cụ hữu ích để tích hợp với các ngôn ngữ lập trình khác. Chẳng hạn như C, C++, và ngôn ngữ lập trình Fortran.
- NumPy cung cấp các chức năng tương đương với MATLAB. Cả hai đều cho phép người dùng thao tác nhanh hơn

### 2.3 Tkinter – Tạo giao diện người dùng (GUI)

Tkinter là thư viện GUI tiêu chuẩn cho Python. Tkinter trong Python cung cấp một cách nhanh chóng và dễ dàng để tạo các ứng dụng GUI. Tkinter cung cấp giao diện hướng đối tượng cho bộ công cụ Tk GUI (Graphic user interface). Tk ban đầu được viết cho ngôn ngữ Tcl. Sau đó Tkinter được viết ra để sử dụng Tk bằng trình thông dịch Tcl trên nền Python. Ngoài Tkinter ra còn có một số công cụ khác giúp tạo một ứng dụng GUI viết bằng Python như wxPython, PyQt, và PyGTK

## 3. Các thuật toán

### 3.1 Canny Edge Detection:

- Phát hiện cạnh Canny (Canny edge detector) là một thuật toán bao gồm nhiều giai đoạn để phát hiện một loạt các cạnh trong hình ảnh. Nó đưa ra một lý thuyết tính toán về phát hiện cạnh giải thích tại sao kỹ thuật này hoạt động
- Thuật toán phát hiện cạnh Canny bao gồm 5 bước:
  - + **Giảm nhiễu:** Việc loại bỏ các yếu tố nhiễu là đặc biệt quan trọng đối với kết quả phát hiện cạnh. Một cách để loại bỏ nhiễu là áp dụng Gaussian Filter giúp làm mịn ảnh. Để làm như vậy, kỹ thuật tích chập hình ảnh được áp dụng với Gaussian Kernel (kích thước 3x3, 5x5, 7x7, v.v.). Kích thước kernel phụ thuộc vào hiệu ứng làm mờ mong muốn. Về cơ bản, kernel càng nhỏ, hiệu ứng mờ càng ít: Câu lệnh trong Python sử dụng thư viện opencv (sử dụng mặt nạ kernel 5x5) với ảnh đầu vào là image, sigmaX và sigmaY là độ lệch chuẩn của mặt nạ chọn là 0 :

*cv2.GaussianBlur(image, ksize=(5,5), sigmaX=0, sigmaY=0)*

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- + **Tính Gradient và hướng gradient:** Bước tính toán Gradient độ xám phát hiện các cạnh thông qua cường độ và hướng của gradient độ xám. Các cạnh tương ứng với sự thay đổi cường độ sáng của pixel. Ta dùng bộ lọc Sobel X và Sobel Y (3x3) để tính được ảnh đạo hàm Gx và Gy. Sau đó, ta tiếp tục tính ảnh Gradient và góc của Gradient theo công thức. Ảnh đạo hàm Gx và Gy là ma trận (ví dụ: 640x640),

thì kết quả tính ảnh đạo hàm Edge Gradient cũng là một ma trận (640x640), mỗi pixel trên ma trận này thể hiện độ lớn của biến đổi mức sáng ở vị trí tương ứng trên ảnh gốc. Tương tự, ma trận Angle cũng có cùng kích thước (640x640), mỗi pixel trên Angle thể hiện góc, hay nói cách khác là hướng của cạnh. Ví dụ dễ hiểu, nếu góc gradient là 0 độ, thì cạnh của ta trên ảnh sẽ là một đường thẳng đứng (tức tạo góc 90 độ so với trục hoành) (vuông góc hướng gradient). Khi tính toán, giá trị hướng gradient sẽ nằm trong đoạn  $[-180, 180]$  độ, ta không giữ nguyên các góc này mà gom các giá trị này về 4 bin đại diện cho 4 hướng: hướng ngang (0 độ), hướng chéo bên phải (45 độ), hướng dọc (90 độ) và hướng chéo trái (135 độ).

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

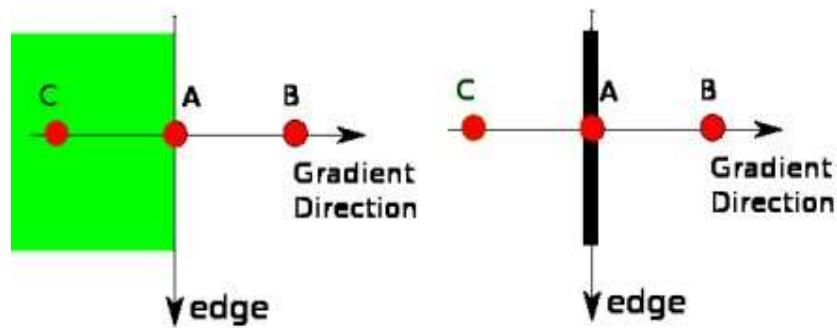
$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

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

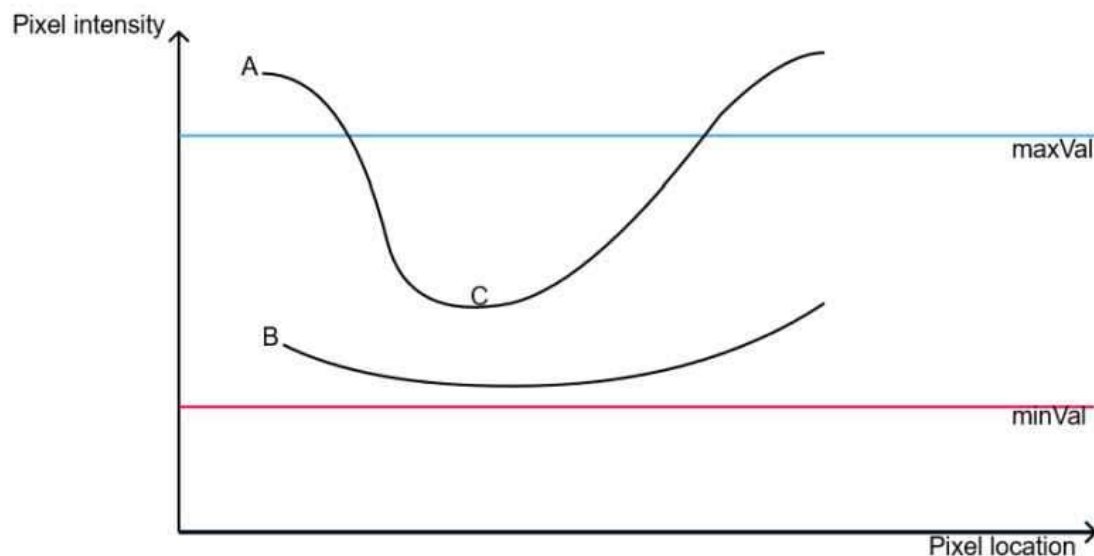
$$Angle(\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

- + **Áp dụng Non-maximum suppression:** Hình ảnh cuối cùng nên có các cạnh mỏng. Vì vậy, ta phải thực hiện thuật toán Non-maximum suppression để làm mỏng chúng. Nguyên tắc rất đơn giản: thuật toán đi qua tất cả các điểm trên ma trận cường độ gradient và tìm các pixel có giá trị lớn nhất theo các hướng cạnh. Loại bỏ các pixel ở vị trí không phải cực đại toàn cục. Ở bước này, ta dùng một filter 3x3 lần lượt chạy qua các pixel trên ảnh gradient. Trong quá trình lọc, ta xem xét xem độ lớn gradient của pixel trung tâm có phải là cực đại (lớn nhất trong cục bộ - local maximum) so với các gradient ở các pixel xung quanh. Nếu là cực đại, ta sẽ ghi nhận sẽ giữ pixel đó lại. Còn nếu pixel tại đó không phải là cực đại lân cận, ta sẽ set độ lớn gradient của nó về zero. Ta chỉ so sánh pixel trung tâm với 2 pixel lân cận theo hướng gradient. Ví dụ: nếu hướng gradient đang là 0 độ, ta sẽ so pixel trung tâm với pixel liền trái và liền phải nó. Trường hợp khác nếu hướng gradient là 45 độ, ta sẽ so sánh với 2 pixel hàng xóm là góc trên bên phải và góc dưới bên trái của pixel trung tâm.





- + **Ngưỡng kép (Double threshold)** : Bước này nhằm mục đích xác định 3 loại pixel: mạnh, yếu và ngoại lai: Pixel mạnh là pixel có cường độ cao và trực tiếp góp phần vào sự hình thành cạnh. Pixel yếu là pixel có giá trị cường độ không đủ để được coi là mạnh nhưng chưa đủ nhỏ để được coi là ngoại lai. Các pixel khác được coi là ngoại lai. Bây giờ bạn có thể thấy vai trò của ngưỡng kép: Ngưỡng cao được sử dụng để xác định các pixel mạnh (cường độ cao hơn ngưỡng cao) Ngưỡng thấp được sử dụng để xác định các pixel ngoại lai (cường độ thấp hơn ngưỡng thấp) Tất cả các điểm ảnh có cường độ giữa cả hai ngưỡng đều được gán là yếu. Cơ chế Độ trễ (bước tiếp theo) sẽ giúp ta xác định xem các Pixel yếu sẽ được giữ lại như các pixel mạnh hay bị loại bỏ như pixel ngoại lai.



- + **Theo dõi cạnh bằng độ trễ (Edge Tracking by Hysteresis)**: Dựa trên kết quả sau khi áp dụng ngưỡng, Hysteresis Tracking chuyển đổi các pixel yếu thành mạnh, khi và chỉ khi ít nhất một trong các pixel xung quanh pixel đang xét là pixel mạnh, các pixel yếu còn lại bị loại bỏ.

### 3.2 Hough Line Transform

- Hough Transform là thuật toán phát hiện đường thẳng khá hiệu quả trong xử lý ảnh
- Ý tưởng chính của giải thuật phát hiện đường thẳng Hough Transform đó là:
  - + Dựa trên kết quả phát hiện cạnh để tiến hành phát hiện đường thẳng. Giải thuật phát hiện cạnh phổ biến là Canny Edge Detection.
  - + Trên mỗi pixel cạnh (pixel thuộc cạnh được phát hiện trong ảnh), ta lần lượt thử các phương trình đường thẳng đi qua pixel đó. Số phương trình đường thẳng ta thử càng nhiều thì sẽ cho ra kết quả phát hiện đường thẳng càng tốt (ít bỏ lỡ đường thẳng có trong ảnh hơn). Pixel cạnh đó sẽ "vote" thêm 1 giá trị vào ma trận thống kê.
  - + Sau khi duyệt hết tất cả các pixel cạnh, ta sẽ lọc theo một giá trị ngưỡng (xác định trước) trên ma trận thống kê để giữ lại (để xác định được) các phương trình đường thẳng có trong ảnh.
- Áp dụng bộ lọc HoughLines để lọc đường thẳng:  
`lines = cv.HoughLines(edges, 1, np.pi / 180, 150, None, 0, 0)`  
Các tham số được sử dụng lần lượt là:
  - + edges: Đầu ra của bộ lọc biên
  - + lines: Vector lưu kết quả dưới dạng (p,θ)

### 3.3 AddWeighted

- AddWeighted là thuật toán cho phép pha trộn ảnh lại với nhau để cho ra một ảnh hoàn chỉnh.
- Thuật toán có cú pháp như sau:  
`cv2.addWeighted(src1, alpha, src2, beta, γ)`

Trong đó:

- + src1: ảnh 1.
- + alpha: trọng số mức sáng của ảnh 1
- + src2: ảnh 2.
- + beta: trọng số mức sáng của ảnh 2.
- + γ (Gamma): điều chỉnh sáng pha trộn

### 3.4 Display lines

- Chương trình con Display lines, dùng để vẽ đường kẻ trên một hình đen có kích thước giống hệt kích thước của hình gốc bằng cách sử dụng hàm `Cv2.line` trong thư viện OpenCv-Python
- Hàm `cv2.line` có cú pháp như sau:  
`cv2.line(image, start_point, end_point, color, thickness)`
  - + image: Là hình ảnh mà cần được vẽ lên
  - + start\_point: Là vị trí tọa độ của điểm đầu của đường kẻ (bao gồm giá trị của x và y)

- + `end_ppint`: Là vị trí toạ độ của điểm kết thúc của đường kẻ (bao gồm giá trị của x và y) `color` : Là màu sắc của đường kẻ mà chúng ta kẻ. Sẽ được hiểu theo biểu thị màu RGB. Ví dụ màu xanh (0,0,255)
- + `thickness` : Là độ dày của đường kẻ

### 3.5 Average\_slope\_intercept

- Sử dụng các hàm trong thư viện Numpy để tính toán các đường kẻ trung bình của mỗi khung hình trong video. Bằng cách sử dụng hàm : `numpy.polyfit, numpy.average`
- Hàm `numpy.polyfit` có cú pháp:
 

```
numpy.polyfit(x, y, deg, rcond=None, full=False, w=None, cov=False)
```


  - + `x,y` : Toạ độ x,y
  - + `deg`: Bậc của đa thức cần tính toán
  - + `rcond` : Giá trị tương đối để
  - + `Full` : Mang giá trị bool
  - + `w` : là một mảng giống như x và y. Nhưng ở đây chúng ta không áp dụng
  - + `cov` : Nếu được cung cấp và không phải là Sai , trả về không chỉ ước tính mà còn cả ma trận hiệp phương sai của nó. Tỷ lệ này được bỏ qua nếu **`cov='unscaled'`**, phù hợp với trường hợp trọng số là  $w = 1/\sigma$ , với  $\sigma$  được biết là ước tính đáng tin cậy của độ không đảm bảo
- Hàm `numpy.average` có cú pháp:
 

```
numpy.average(x, axis=None, weights=None, returned=False)
```


  - + `x` : là mảng mà chúng ta cần tính giá trị trung bình của các giá trị trong mảng đó
  - + `Axis`: Tham số này xác định trục dọc theo đó giá trị trung bình sẽ được tính toán. Theo mặc định, trục được đặt thành Không có, trục này sẽ tính giá trị trung bình của tất cả các phần tử của mảng nguồn. Đếm bắt đầu từ kết thúc đến trục bắt đầu khi giá trị của trục là âm.
  - + `Weights`: Tham số này xác định một mảng chứa các trọng số được liên kết với các giá trị mảng. Mỗi giá trị của các phần tử mảng cùng nhau tạo thành giá trị trung bình theo trọng số liên quan của nó. Mảng trọng số có thể là một chiều hoặc có cùng hình dạng với mảng đầu vào. Khi không có trọng số nào được liên kết với phần tử mảng, trọng số sẽ được coi là 1 cho tất cả các phần tử.
  - + `Returned`: Theo mặc định, tham số này được đặt thành Sai. Nếu chúng ta đặt nó là True, thì một bộ giá trị trung bình và `sum_of_weights` sẽ được trả về. Nếu nó là Sai, giá trị trung bình được trả về. Tổng trọng số tương đương với số phần tử nếu không có giá trị nào cho trọng số.

### CHƯƠNG III: NỘI DUNG LẬP TRÌNH

#### 1. Giao diện theo yêu cầu:




**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM**  
**KHOA CƠ KHÍ CHẾ TẠO MÁY**  
**NGÀNH CNKT CƠ ĐIỆN TỬ**



**MÔN HỌC: THỊ GIÁC MÁY**  
**ĐỀ TÀI BÁO CÁO CUỐI KỲ:**

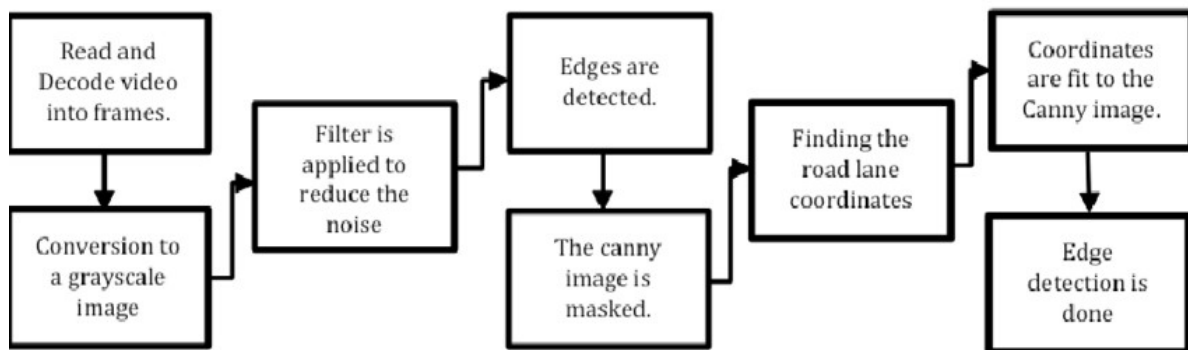
**NHẬN DIỆN ĐƯỜNG LANE TRONG NHIỀU  
ĐIỀU KIỆN THỜI TIẾT KHÁC NHAU:  
NẮNG, MƯA, ĐÊM, ...**

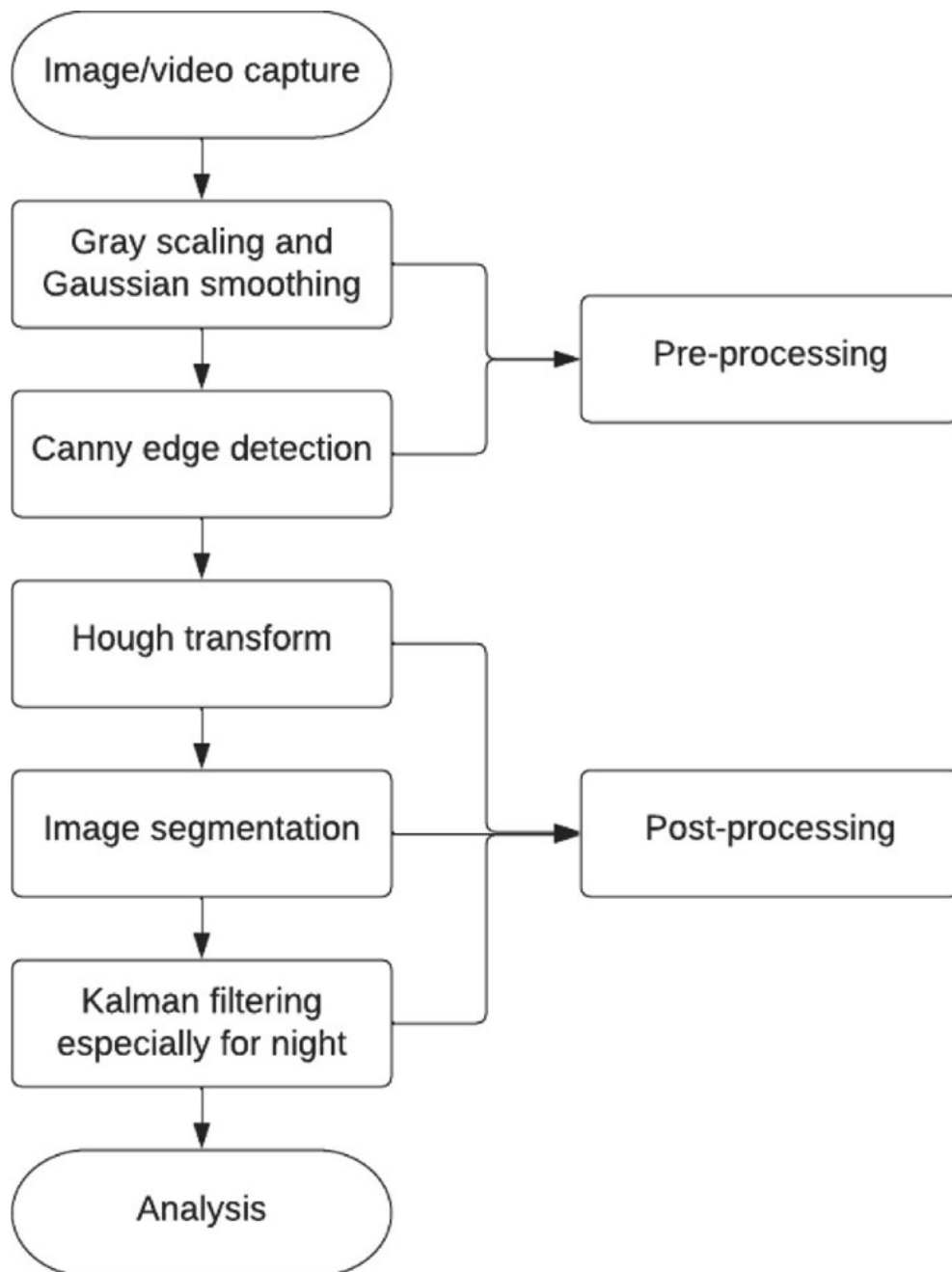


**GVHD: TS Nguyễn Văn Thái**

SVTH:  
Bùi Trung Kiên 21146115  
Võ Thành Đạt 21146450  
Lê Duy Huy 21146467

#### 2. Các bước thực hiện và lưu đồ giải thuật





### 3. Code

- Khai báo thư viện:

```
from tkinter import *
import tkinter.messagebox
import numpy as np
import cv2
```

Các thư viện sử dụng trong chương trình là: opencv , tkinter , numpy

- Tạo giao diện:

```
root = Tk()
root.title('NHOM_03')
root.geometry('960x593')
image = PhotoImage(file="bg.png")
label = Label(image=image)
label.pack()
```

Tạo giao diện người dùng sử dụng thư viện tkinter

Bước 1: Ta tạo một cửa sổ root của giao diện , đặt tên giao diện với câu lên

root.title, chọn kích thước của sổ với câu lên root.geometry

Bước 2: Tải ảnh bìa đã chuẩn bị lên và gán vào nhãn label để hiển thị hình đã tải lên.

Bước 3: Tạo một nút nhấn để vào chương trình chính (đặt tên nút nhấn, vị trí đặt nút nhấn )

- Hàm con Canny

```
def canny(img): # Dùng thuật toán Canny Edge Detection để phát hiện các góc cạnh trong bức ảnh .
    if img is None:
        cap.release()
        cv2.destroyAllWindows()
        exit()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # chuyển ảnh RGB sang Gray
    # Áp dụng bộ lọc Gauss để giảm nhiễu ảnh với mặt nạ kích thước 5x5
    blur = cv2.GaussianBlur(gray, ksize=(5,5), sigmaX=0, sigmaY=0)

    # Dùng thuật toán canny để xác định cạnh trong ảnh với 2 ngưỡng tự chọn
    canny = cv2.Canny(blur, 50, 150) # các ngưỡng so sánh với cường độ pixel
    return canny
```

Hàm trên dùng thuật toán Canny Edge Detection để phát hiện các góc cạnh trong

bức ảnh. Với đầu vào là một ảnh

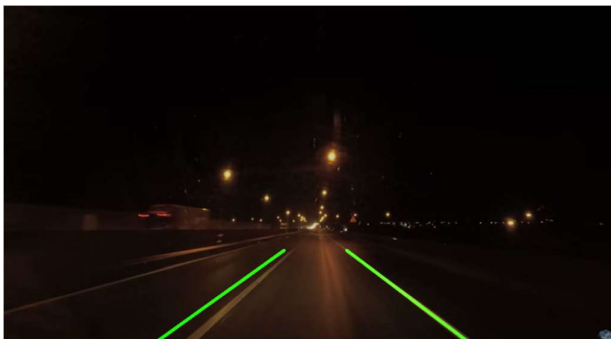
Bước 1: Chuyển ảnh đầu vào thành ảnh gray

Bước 2: Làm mờ ảnh gray bằng bộ lọc Gauss kích thước mặt nạ 5x5

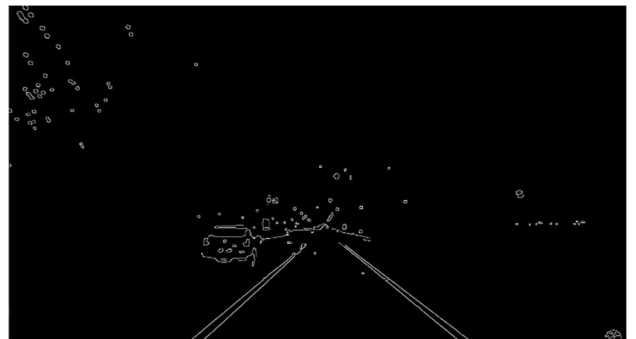
Bước 3: Dùng câu lệnh cv2.Canny với 2 ngưỡng tự chọn, ngưỡng trên là 150, ngưỡng dưới là 50. Ngưỡng này dùng để so sánh cường độ pixel.

Bước 4: Sau khi so sánh ta được hình canny với các góc cạnh được phát hiện

Kết quả sau khi sử dụng hàm con canny



Input



Output

- Hàm con make\_points

```
def make_points(image, line): #Tạo ra 2 điểm dựa vào phương trình đường thẳng
    # slope là hệ số góc , intercept là hệ số giao
    slope, intercept = line # tính hệ số góc và hệ số giao của đường thẳng
    y1 = int(image.shape[0])
    y2 = int(y1 * 3.5 / 5)
    x1 = int((y1 - intercept) / slope)
    x2 = int((y2 - intercept) / slope)
    return [[x1, y1, x2, y2]]
```

Đầu vào là hình ảnh và đường thẳng (đường thẳng đã được detect ở phía trên )

Slope: với hệ số góc

Intercept: hệ số cắt trục y

x1,y1 ; x2,y2 lần lượt là tọa độ 2 điểm đầu và cuối

Phương trình tổng quát bậc 1 của đường thẳng :

$$y=ax+b \text{ ( a là hệ số góc , b là hệ số giao trục y ) } \Rightarrow x= (y-b)/a$$

Ta chọn tung độ điểm đầu của đường thẳng y1= chiều cao của hình và tung độ điểm cuối của đường thẳng là y2= 3.5(y1)/5

Sau khi chọn xong ta tính x1, x2 dựa vào công thức trên.

Hàm trả về một mảng còn tọa độ 2 điểm của đường thẳng.

- Hàm Display\_line

```
def display_lines(img, lines): # Vẽ ra các đường thẳng đã được detect vào hình

    line_image = np.zeros_like(img)
    if lines is not None:
        for line in lines:
            for x1, y1, x2, y2 in line:
                cv2.line(line_image, (x1, y1), (x2, y2), (0, 255, 0), 5) # hàm để vẽ đường thẳng
    return line_image
```

Bước 1: Chúng ta sẽ tạo một ma trận có kích thước giống hình đầu vào. Và các giá trị của ma trận đó sẽ mang giá trị 0 để tạo thành một hình đen.

Bước 2: Chúng ta sẽ kiểm tra xem có đường kẻ nào có trong ma trận đầu vào không. Nếu có thì sẽ bắt đầu chạy vòng lặp

Bước 3: Chạy vòng lặp trong ma trận để vẽ từng đường kẻ một.

Bước 4: Chạy vòng lặp từng điểm tọa độ x,y của điểm đầu và điểm cuối của đường thẳng cần vẽ.

Bước 5: Sử dụng hàm cv2.line để vẽ đường thẳng cần vẽ lên ảnh.

Bước 6: Trả về ảnh mà đã vẽ đường kẻ lên.



## Hàm average slope intercept

```
def average_slope_intercept(image, lines):
    #Tạo ra một phương trình đường thẳng trung bình dựa vào các lines mà đã được detect ở phía trên

    left_fit = []
    right_fit = []
    if lines is None:
        return None
    for line in lines:
        for x1, y1, x2, y2 in line:
            fit = np.polyfit((x1, x2), (y1, y2), 1) # tính pt đường thẳng
            slope = fit[0]
            intercept = fit[1]
            if slope < 0:
                left_fit.append((slope, intercept))
            else:
                right_fit.append((slope, intercept))

    #Kiểm tra nếu left_fit hoặc right_fit rỗng
    if len(left_fit) == 0:
        left_fit_average = None
    else:
        left_fit_average = np.average(left_fit, axis=0)

    if len(right_fit) == 0:
        right_fit_average = None
    else:
        right_fit_average = np.average(right_fit, axis=0)

    # Xử lý trường hợp left_fit_average hoặc right_fit_average là None
    if left_fit_average is None or right_fit_average is None:
        # Trả về một đường thẳng mặc định hoặc None
        return None

    left_line = make_points(image, left_fit_average)
    right_line = make_points(image, right_fit_average)
    averaged_lines = (left_line, right_line)
    return averaged_lines
```

Bước 1: Tạo hai ma trận để chứa giá trị hệ số góc và hệ số giao của các đường thẳng trái và phải.

Bước 2: Kiểm tra có giá trị trong mảng đầu vào không

Bước 3: Chạy vòng lặp từng giá trị đường kẻ trong mảng đầu vào

Bước 4: Chạy vòng lặp giá trị tọa độ (x,y) của điểm đầu và điểm cuối của đường

thẳng đang xét.

Bước 5: Dùng hàm numpy.polyfit để xác định giá trị hệ số tọa độ gốc và hệ số giao

của đường thẳng dựa vào tọa độ (x,y) của 2 điểm đầu và cuối của đường thẳng

Bước 6: Lấy giá trị slope: hệ số góc, intercept: hệ số giao đã được tính toán

Bước 7: Thực hiện mệnh đề if

- + slope < 0 ( Hay có nghĩa đường thẳng nằm bên trái trục tung) thì sẽ coi là đường kẻ bên trái. Và thêm giá trị slope và intercept vào mảng left\_fit
- + slope > 0 (Hay có nghĩa đường thẳng nằm bên phải trục tung) thì sẽ coi là đường kẻ bên phải. Và thêm giá trị slope và intercept vào mảng right\_fit

Bước 8: Kiểm tra xem có đường thẳng nào thuộc về làn đường bên trái hoặc bên phải hay không, và tính toán trung bình cộng các hệ số của những đường thẳng đó để lấy ra đường thẳng trung bình đại diện cho mỗi làn đường.

- + Kiểm tra nếu left\_fit rỗng, gán left\_fit\_average là None. Ngược lại, tính trung bình cộng các cặp (slope, intercept) trong left\_fit để lấy left\_fit\_average.
- + Tương tự, kiểm tra nếu right\_fit rỗng, gán right\_fit\_average là None. Ngược lại, tính trung bình cộng các cặp (slope, intercept) trong right\_fit để lấy right\_fit\_average.
- + Nếu cả left\_fit\_average và right\_fit\_average đều là None (không có đường trung bình cho cả hai bên), trả về None hoặc một đường thẳng mặc định.

Bước 9: Tính giá trị trung bình của tất cả các giá trị đường kẻ được chứa trong mảng left\_fit, right\_fit

Bước 10: Vẽ hai đường kẻ trái phải trung bình sau khi được tính toán bằng hàm make\_points

Bước 11: Lưu trữ 2 đường kẻ trái phải trung bình vào mảng mới tạo averaged Lines

Bước 12: Trả về averaged\_lines chứa hai đường thẳng trung bình đại diện cho làn đường bên trái và bên phải

- Hàm houghline

```
def houghLine(cropped_canny): # trả về mảng chứa các đường thẳng có trong ảnh
    return cv2.HoughLinesP ( cropped_canny, 2, np.pi / 180, 100, np.array([]), minLineLength=10, maxLineGap=2 )
```

Chúng ta sẽ trả về mảng chứa các đường thẳng có trong ảnh.

Với:

- + Cropped\_canny: ảnh sau khi crop
- + 2 giá trị rho tính theo giá trị pixel
- + np.pi / 180: giá trị tính theo độ
- + 100: là ngưỡng
- + minLineLength=10: chiều dài tối thiểu của đường thẳng là 10

- + maxLineGap=2: độ rộng tối đa của đường thẳng là 22
- Hàm addWeighted

```
def addWeighted(frame, line_image): #Trộn 2 ảnh lại với nhau để ra ảnh hoàn chỉnh ( ảnh gốc và ảnh line)  
    return cv2.addWeighted(frame, 0.8, line_image, 1, 0)
```

Trả về ảnh trộn 2 ảnh lại với nhau.

Với:

- + frame: ảnh 1
- + 0.8: trọng số mức sáng của ảnh 1
- + line\_image: ảnh 2
- + 1: trọng số mức sáng của ảnh 2
- + 0: mức sáng sau khi pha trộn

## CHƯƠNG IV: KẾT QUẢ ĐẠT ĐƯỢC

### 1. Kết quả:

Xây dựng được chương trình nhận dạng lane đường thực tế bằng thuật toán Canny Edge Detection. Vận dụng được các kiến thức môn học xử lý ảnh, tìm hiểu và áp dụng được các thuật toán xử lý ảnh nâng cao hơn để xây dựng chương trình. Chương trình hoạt động khá ổn định, giao diện dễ dùng, cho kết quả rõ nét.

### 2. Hạn chế:

Chương trình vẫn còn một số hạn chế nhất định: Đôi khi nhận dạng còn chưa chính xác. Một số trường hợp không phân biệt được lane đường do có nhiều chi tiết đường thẳng và video có độ phức tạp cao.

### 3. Hướng phát triển:

Từ đề tài này có thể phát triển ra ý tưởng mô hình xe tự lái xử dụng thêm các cảm biến khác để làm tăng độ chính xác và an toàn cho người sử dụng.

## TÀI LIỆU THAM KHẢO

- [1] "Feature Detection," [Online]. Available: [https://docs.opencv.org/2.4/modules/imgproc/doc/feature\\_detection.html?highlight=canny#canny](https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=canny#canny).
- [2] "OpenCV used to draw a line on any image," [Online]. Available: <https://www.geeksforgeeks.org/python-opencv-cv2-line-method/>.
- [3] H. L. Transform. [Online]. Available: [https://docs.opencv.org/3.4/d9/db0/tutorial\\_hough\\_lines.html](https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html).
- [4] "Pha trộn ảnh trong OpenCV (blending)," [Online]. Available: <https://minhng.info/tutorials/blending-anh-opencv.html>.
- [5] "Canny Edge Detection," [Online]. Available: [https://docs.opencv.org/3.4/da/d22/tutorial\\_py\\_canny.html](https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html).
- [6] "Edge detection and processing using Canny edge detector and Hough transform," [Online]. Available: <https://wttech.blog/blog/2022/edge-detection-and-processing-using-canny-edge-detector-and-hough-transform/>.
- [7] "Canny – Phát hiện cạnh trong OpenCV: Hướng dẫn chi tiết từng bước," [Online]. Available: <https://vinbigdata.com/kham-pha/canny-phat-hien-can-h-trong-opencv-huong-dan-chi-tiet-tung-buoc.html>.