# ASSIGNMENT 2: LANE DETECTION

## Nội dung



## 1. Canny detection

- Là thuật toán phát hiện cạnh dựa trên sự thay đổi của gradient điểm ảnh theo hướng x và y.

>>> Đầu tiên cần làm mịn ảnh để bớt nhiễu và cho qua 2 bộ lọc Sobel theo x và theo y để lấy được ma trận độ lớn gradient theo hướng x và y. Sau đó tính ma trận gradient cạnh bằng cách tính khoảng cách L2 của 2 ma trận Gx và Gy, ma trận này thể hiện độ lớn biến đổi mức sáng của các pixel.

```
smooth_img = cv2.GaussianBlur(img, ksize=(5, 5), sigmaX=1, sigmaY=1)
Gx = cv2.Sobel(smooth_img, cv2.CV_64F, 1, 0, ksize=sobel_size)
Gy = cv2.Sobel(smooth_img, cv2.CV_64F, 0, 1, ksize=sobel_size)
edge_gradient = np.sqrt(Gx*Gx + Gy*Gy)
```

>>> Tính ma trận góc (Angle) thể hiện hướng của cạnh tương ứng với từng pixel. Chia các góc về 4 bin chính để dễ cho việc tính toán. Bin 0 là các cạnh có hướng thẳng đứng, bin 45 là các cạnh có hướng chéo bên phải, bin 90 là cạnh nằm ngang và bin 135 là các cạnh có hướng chéo bên trái.

```
angle_radian = np.arctan2(Gy, Gx)
angle = angle_radian * 180 / np.pi

# round angle to 4 directions
angle = np.abs(angle)
angle[angle <= 22.5] = 0
angle[angle >= 157.5] = 0
angle[(angle > 22.5) * (angle < 67.5)] = 45
angle[(angle >= 67.5) * (angle <= 112.5)] = 90
angle[(angle > 112.5) * (angle <= 157.5)] = 135
```

>>> Trượt 1 cửa sổ qua ma trận cạnh và lấy ra các giá trị độ lớn gradient pixel, góc pixel, độ lớn gradient của pixel nằm giữa và góc pixel nằm giữa của cửa sổ

```
for y in range(1, edge_gradient.shape[0]-1):
    for x in range(1, edge_gradient.shape[1]-1):
        area_grad_intensity = edge_gradient[y-1:y+2, x-1:x+2] # 3x3 area
        area_angle = angle[y-1:y+2, x-1:x+2] # 3x3 area
        current_angle = area_angle[1,1]
        current_grad_intensity = area_grad_intensity[1,1]
```

>>> Ứng với mỗi pixel giữa, ta sẽ xem xét nó thuộc bin số mấy để so sánh độ lớn gradient với 2 điểm nằm 2 bên đường thẳng mà pixel đó thuộc về. Nếu như nó lớn hơn thì giữ lại còn không thì sẽ bị bỏ đi.

```
if current_angle == 0:
    if current_grad_intensity > max(area_grad_intensity[1,0],
area_grad_intensity[1,2]):
        keep_mask[y,x] = 255
    else:
        edge_gradient[y,x] = 0
elif current_angle == 45:
    if current_grad_intensity > max(area_grad_intensity[2,0],
area_grad_intensity[0,2]):
        keep_mask[y,x] = 255
    else:
        edge_gradient[y,x] = 0
elif current_angle == 90:
    if current_grad_intensity > max(area_grad_intensity[0,1],
area_grad_intensity[2,1]):
        keep_mask[y,x] = 255
    else:
        edge_gradient[y,x] = 0
elif current_angle == 135:
    if current_grad_intensity > max(area_grad_intensity[0,0],
area_grad_intensity[2,2]):
        keep_mask[y,x] = 255
    else:
        edge_gradient[y,x] = 0
```

>>> Xét các pixel được giữ lại, nếu như độ lớn đạo hàm của nó lớn hơn giá trị ngưỡng thì sẽ được giữ lại. còn không thì sẽ bị bỏ đi.

```
canny_mask = np.zeros(smooth_img.shape, np.uint8)
canny_mask[(keep_mask>0) * (edge_gradient>min_val)] = 255
```

## 2. Hough Transform

- Là thuật toán tìm ra cạnh dựa trên các pixel cạnh tiềm năng bằng cách tìm thử các đường thẳng nào đi qua nhiều pixel tìm năng nhất.
- Đường thẳng qua 1 pixel sẽ được đặc trưng bởi 2 tham số đó là góc của đường thẳng và khoảng cách từ góc toạ độ tới đường thẳng đó.

>>> Lấy ra chiều dài và rộng của ảnh để tính độ dài đường chéo (khoảng cách tối đa có thể đạt được từ gốc toạ độ (góc trái trên) tới bất kì cạnh nào đi qua 1 pixel trên ảnh.

```
img_height, img_width = img.shape[:2]
diagonal_length = int(math.sqrt(img_height*img_height + img_width*img_width))
```

>>> Tính số lượng các khoảng cách và góc của cạnh cần phải thử để tìm ra đường thẳng. Từ đó xây dựng ma trận cạnh để tìm ra các đường thẳng đi qua nhiều điểm nhất.

```
num_rho = int(diagonal_length / rho)
num_theta = int(np.pi / theta)
edge_matrix = np.zeros([2*num_rho+1, num_theta])
```

>>> Tìm ra các pixel tiềm năng, tạo ma trận góc sau đó nhân ma trận pixel với ma trận góc để tìm được ma trận khoảng cách tương ứng với các đường thẳng có thể đi qua từng pixel tiềm năng $\rho = x\sin(\theta) + y\cos(\theta)$.

```
idx = np.squeeze(cv2.findNonZero(img))
range_theta = np.arange(0, np.pi, theta)
theta_matrix = np.stack((np.cos(np.copy(range_theta)),
np.sin(np.copy(range_theta))), axis=-1)
vote_matrix = np.dot(idx, np.transpose(theta_matrix))
```

>>> Duyệt qua từng phần tử trong ma trận mới thu được, đếm số lần xuất hiện của các giá trị khoảng cách và góc xuất hiện cùng nhau, cũng chính là số pixel tiềm năng mà 1 đường thẳng đã đi qua.

```
for vr in range(vote_matrix.shape[0]):
    for vc in range(vote_matrix.shape[1]):
        rho_pos = int(round(vote_matrix[vr, vc]))+num_rho
        edge_matrix[rho_pos, vc] += 1
```

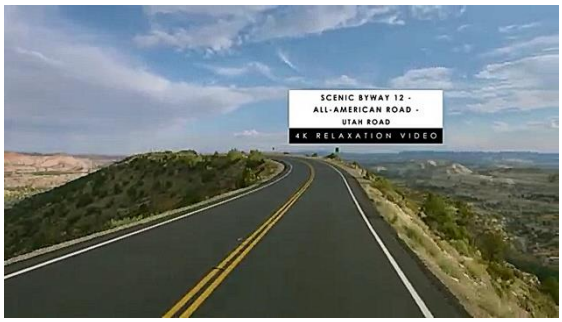
>>> Lấy ra các đường thẳng đi qua số điểm lớn hơn giá trị ngưỡng (các đường thẳng chỉ đi qua vài pixel thì coi như không phải là cạnh).

```
line_idx = np.where(edge_matrix > threshold)
```

## 3. Lane detection

- Ứng dụng Color space segmentation, Canny detection và Hough Transform để nhận diện các đường phần chia làn đường.

>>> Nhận thấy con đường chỉ xuất hiện trong 1 khoảng không gian nhất định trong từng khung ảnh trong video, nên em xây dựng một ảnh RoI để áp dụng lên từng frame và chỉ xử lý những vùng nằm trong RoI.




*Hình 1. Khung hình trong video*

*Hình 2 RoI*

>>> Đọc video, đọc RoI, xác định các khoảng màu vàng trong không gian màu HSV.

```
video = cv2.VideoCapture("images/line.mp4")
roi = cv2.imread('images/roi.jpg', 0)
low_yellow = np.array([20, 140, 125])
up_yellow = np.array([30, 255, 255])
```

>>> Xây dựng hàm để áp dụng vùng RoI lên một ảnh.

```
def apply_roi(img, roi):
    img = cv2.bitwise_and(img, img, mask=roi)
    return img
```

>>> Đọc từng khung hình, chuyển qua hệ màu GrayScale, và HSV. Dùng ảnh HSV để tìm được những pixel có màu vàng, dung ảnh Grayscale để tìm những pixel có màu trắng.

```
ret, frame = video.read()

gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

yellow_mask = cv2.inRange(hsv, low_yellow, up_yellow)
white_mask = cv2.inRange(gray, 150, 255)
```

>>> Cộng cả 2 lại rồi áp dụng RoI lên để chỉ giữ lại những pixel màu vàng và trắng nằm trên con đường. Sau đó dùng Canny detection để detect các pixel

cạnh (line) và dùng Hough transform để tìm các đường thẳng chứa các pixel cạnh (line).

```
mask = apply_roi(yellow_mask + white_mask, roi)
edges = cv2.Canny(mask, 50, 100)
lines = cv2.HoughLinesP(edges, 1, np.pi/180, 50, maxLineGap=30)
```

## 4. Nhận xét

- Việc xây dựng một vùng RoI là quan trọng vì có rất nhiều những pixel có màu trắng và vàng như mây, cỏ héo, đất cát, … Tuy có thể chọn một khoảng màu chính xác hơn nhưng sẽ rất tốn thời gian và có thể mất đi một số pixel.
- Sẽ tốt hơn nếu như dùng cả 2 kỹ thuật là giới hạn khoảng màu và dùng RoI. Cả 2 kỹ thuật đều có nhược điểm: RoI có thể không đúng trong một số trường hợp vật thể đột ngột thay đổi (đoạn đường đột nhiên to ra hay nhỏ lại). Còn xét khoảng màu cũng có trường hợp cả object muốn nhận diện và object không muốn nhận diện đều cùng 1 màu như line trắng và mây đều có chung 1 màu nên không thể xét một khoảng màu nào cố định để lấy 1 trong 2 được).


*Hình 3. White mask*

- Việc dùng Gaussian Blur có thể giúp ích trong việc loại bỏ một số nhiễu nhưng cũng đồng thời làm mất đi một số điểm ảnh nằm trên cạnh, khiến Canny detection nhận diện thiếu.