# Simple Lane Detection

https://medium.com/@mrhwick/simple-lane-detection-with-opencv-bfeb6ae54ec0

1. Take simple video as input data and process it to detect the lane
2. Find a representative line for both the left and right lane lines
3. Render those representations back out to the video as a red overlay

** OpenCV(process the input images to discover any lane line and rendering a representation)
** Numpy and Matplotlib(transformation and rendering of image data)

# Cropping to a Region of Interest

## 1. Loading an Image into Memory

load an image from a file into array of image data which can be manipulated in python

```
In [6]: print('This image is : ', type(image), 'with dimensions : ', image.shape)
        plt.imshow(image)
        plt.show()

        This image is :  <class 'numpy.ndarray'> with dimensions :  (540, 960, 3)
```


Dimension (540, 960, 3)
# height, width, channel

## 2. Defining the Region of Interest

We want a region of interest that fully contains the lane lines (** realize that the point (0, 0) is actually in the upper left corner of the image)

```
In [8]: height, width, channel = image.shape

In [9]: region_of_interest_vertices = [
            (0, height),
            (width / 2, height / 2),
            (width, height)
        ]
```

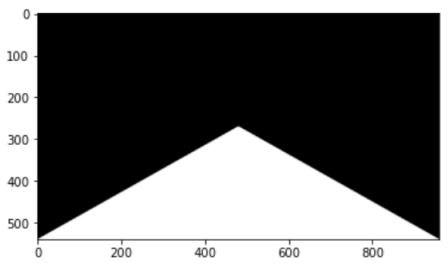
## 3. Cropping the Region of Interest

To actually do the cropping of the image, we'll define a utility function region_of_interest():

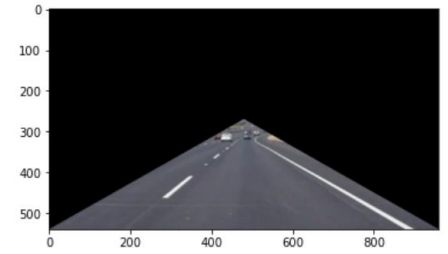
```
pip install opencv-python

Collecting opencv-python
  Downloading opencv_python-4.5.5.64-cp36-abi3-win_amd64.whl (35.4 MB)
Requirement already satisfied: numpy>=1.14.5 in c:₩users₩hyeonjin₩anaconda3₩lib₩site-packages (from opencv-python) (1.20.3)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.5.5.64
Note: you may need to restart the kernel to use updated packages.
```

- 사용자 정의함수 구현(region_of_interest)

```
import numpy as np
import cv2

def region_of_interest(img, vertices):
    mask = np.zeros_like(img) # define a blank mask that matches the image height / width.
    channel_count = img.shape[2] # retrieve the number of color channels
    match_mask_color = (255, ) * channel_count # create a match color with the same color channel counts
    cv2.fillPoly(mask, vertices, match_mask_color) # fill inside the polygon
    masked_image = cv2.bitwise_and(img, mask) # return the image only where mask pixels match
    return masked_image
```




fillPoly : 다각형 내부 색 채우기

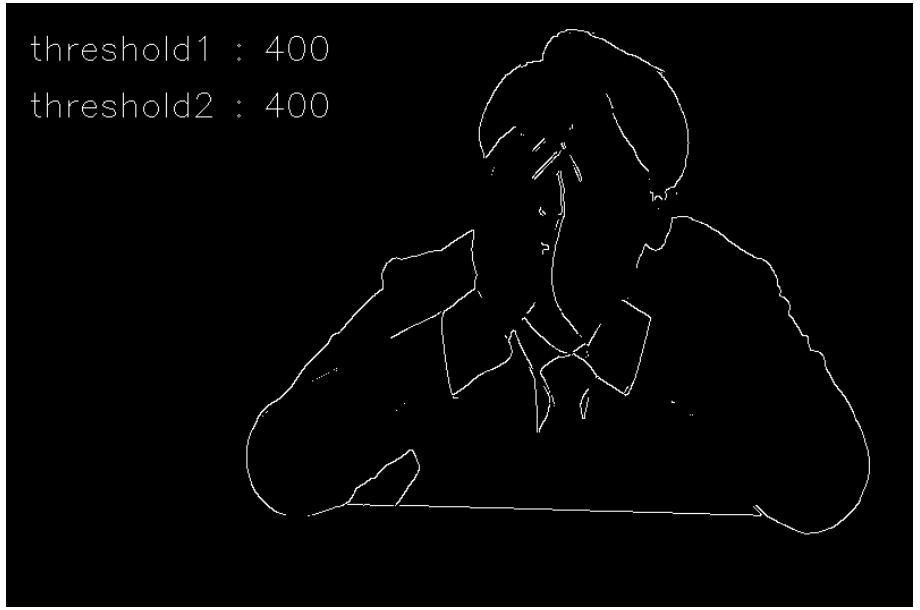



cv2_bitwise_and : mask와 image 모두 색 있는 부분만 표현

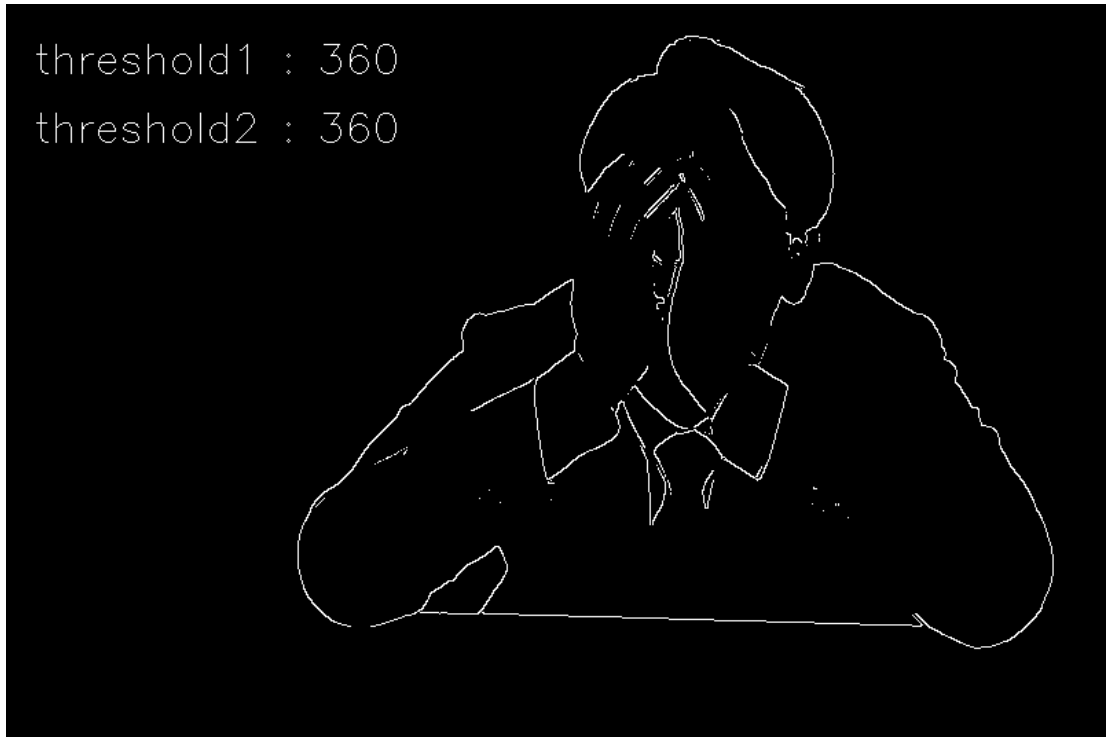
# Cv2.Canny()

OpenCV에 제공되어 있는 엣지 검출 함수로, 임계값 두 개가 클수록 엣지가 검출되기 어렵고, 작을수록 엣지가 검출되기 쉽다.

**Threshold 2**
직감적인 값, 엣지를 판단하는 임계값 그 자체



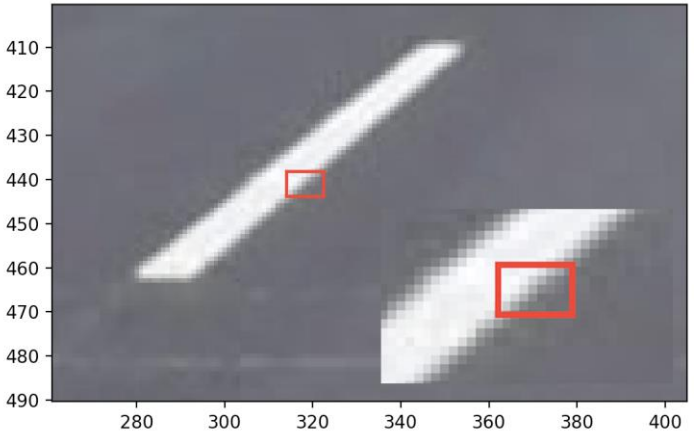

**Threshold 1**
다른 엣지와의 인접 부분의 엣지 여부를 판단하는 임계값
작은 값으로 설정하면, 원래 threshold2에 의해 검출된 엣지의 인접 부분이 엣지로 검출하기 쉬워짐 즉, 엣지의 선 길게 늘림



# Detecting Edges in the Crop Image

## 1. Mathematics of Edge Detection

Areas of an image where the color values change very quickly. (where a pixel is a mismatch in color to all of its neighbors.

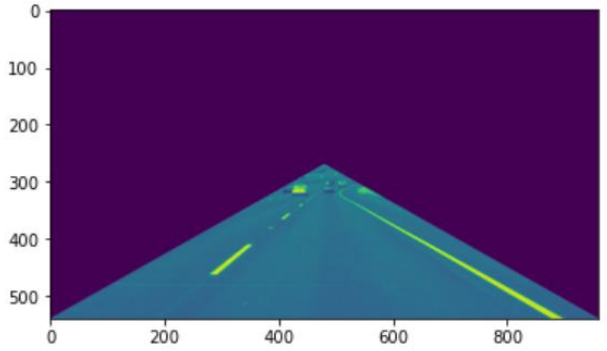



→ Strong gradient in the image's color function.

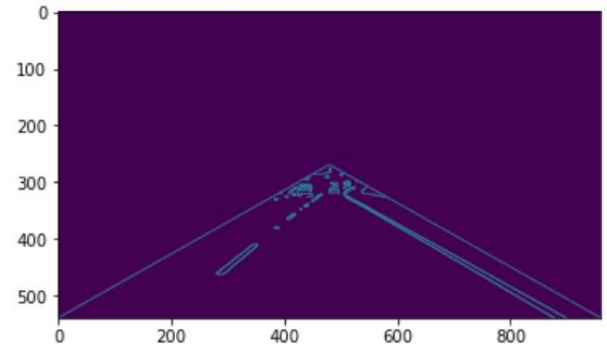
## 2. Grayscale Conversion and Canny Edge Detection

Remove color information and replace it with a single intensity value for each pixel and find areas of the image that rapidly change over the intensity value.

```
gray_image = cv2.cvtColor(cropped_image, cv2.COLOR_RGB2GRAY)
cannyed_image = cv2.Canny(gray_image, 100, 200)
plt.figure()
plt.imshow(cannyed_image)
plt.show()
```




gray_image

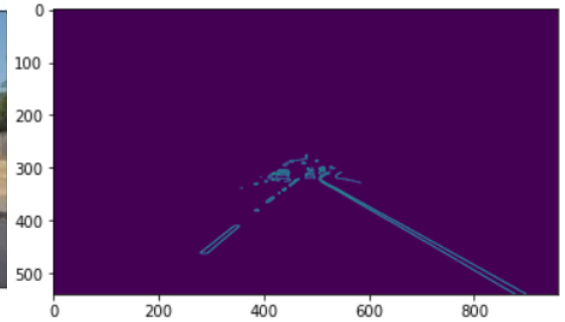



cannyed_image

# Detecting Edges in the Crop Image

**3. Place the region of interest cropping after the Canny process in our pipeline.**
Grayscale conversion → Canny edge detection
→ Cropped Image
because we must not detect the edge our cropped region of interest!

Original image

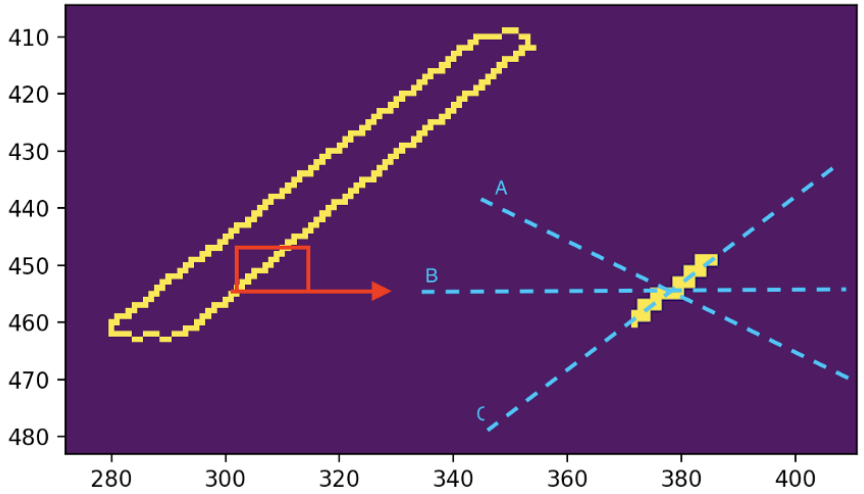


Canny edge detection

# Generating Lines from Edge Pixels

## 1. Mathematics of Line Detection

Discover lines which intersect multiple edge pixels at once.

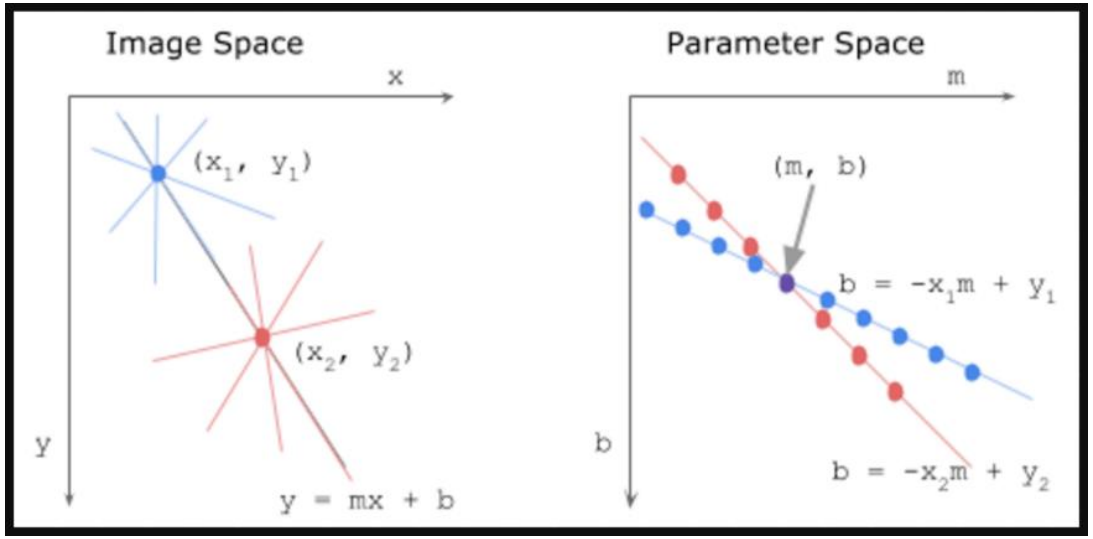



→ Hough Transform

```
lines = cv2.HoughLinesP(
    cropped_image,
    rho = 6,
    theta = np.pi / 60,
    threshold = 160,
    lines = np.array([]),
    minLineLength = 40,
    maxLineGap = 25
)
```

## 2. Hough Transform

한 점이 가질 수 있는 모든 직선은 m과 b에 대한 평면에서 하나의 직선으로 표현할 수 있다.
두 직선의 교점(xy평면에서 기울기와 절편)
→ 두 점을 지나는 직선을 의미




```
lines = cv2.HoughLinesP(
    cropped_image,
    rho = 6,
    theta = np.pi / 60,
    threshold = 160,
    lines = np.array([]),
    minLineLength = 40,
    maxLineGap = 25
)
print(lines) # [x1, y1, x2, y2]
```

```
[[[483 311 878 539]]

 [[647 399 716 438]]

 [[704 430 806 488]]
```

# Generating Lines from Edge Pixels

### 3. Rendering Detected Hough Lines as an Overlay

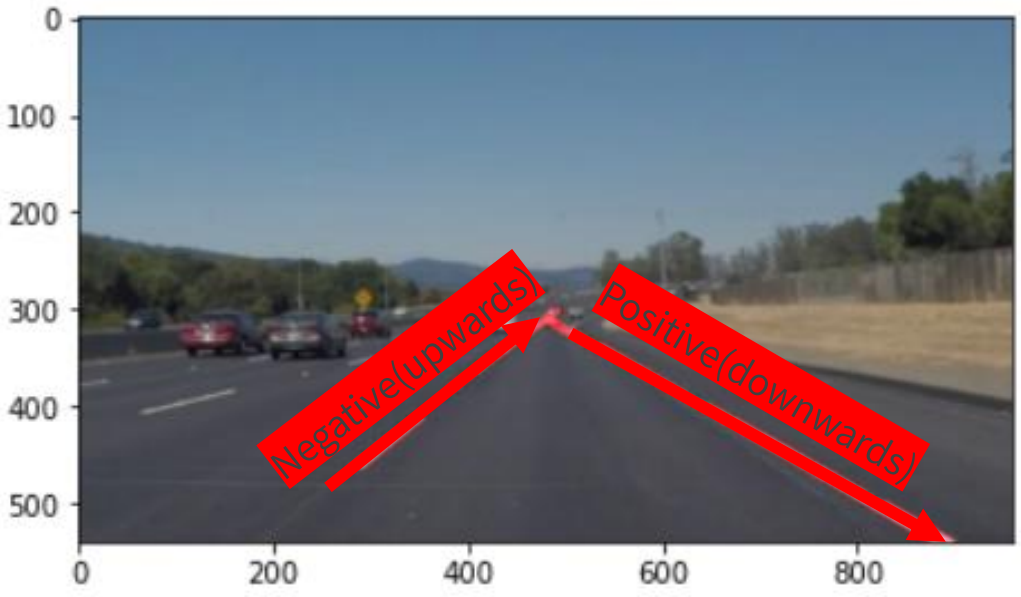
Have copy of the original with the detected lines rendered as an overlay.

```
def draw_lines(img, lines, color = [255, 0, 0], thickness = 3):
    if lines is None:
        return

    img = np.copy(img) # make a copy of the original image
    line_img = np.zeros((img.shape[0], img.shape[1], 3), dtype = np.uint8,) # create a blank image that matches the original

    for line in lines: # loop over all lines and draw them on the blank image
        for x1, y1, x2, y2 in line:
            cv2.line(line_img, (x1, y1), (x2, y2), color, thickness)

    img = cv2.addWeighted(img, 0.8, line_img, 1.0, 0.0) # merge the image with the lines onto the original(img * 0.8 + line_img * 1)
    return img
```

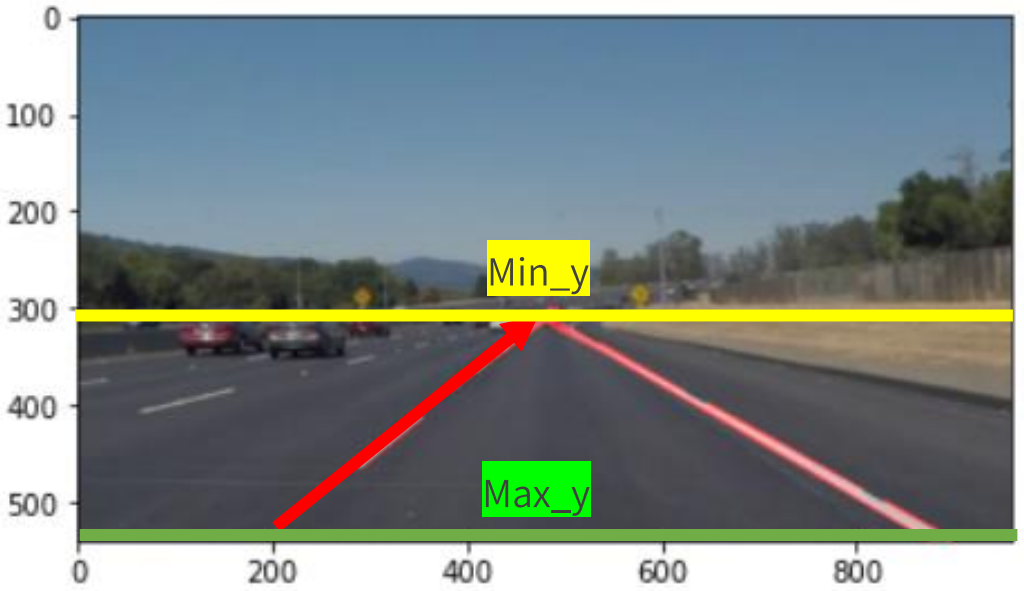
# Creating a Single Left and Right Line

**1. Grouping the Lines into Left and Right Groups**
- Negative slopes : traveling upwards
- Positive slopes : traveling downwards
- Lane markings appear extreme in slope

→ move quickly towards the horizon or bottom of the image




**2. Creating a Single Linear Representation of each Line Group**
- Begin at the bottom of the image
- End just below the horizon



# Creating a Single Left and Right Line

## 1. Grouping the Lines into Left and Right Groups

- Negative slopes : traveling upwards
- Positive slopes : traveling downwards
- Lane markings appear extreme in slope

→ move quickly towards the horizon or bottom of the image

```
import math
left_line_x = []
left_line_y = []
right_line_x = []
right_line_y = []

for line in lines:
    for x1, y1, x2, y2 in line:
        slope = (y2 - y1) / (x2 - x1)
        if math.fabs(slope) < 0.5: # 기울기의 절대값 비교
            continue
        if slope <= 0:# negative, left group
            left_line_x.extend([x1, x2])
            left_line_y.extend([y1, y2])
        else: # positive, right group
            right_line_x.extend([x1, x2])
            right_line_y.extend([y1, y2])
```

```
print(*left_line_x)
```

```
383 465 383 493 383 493 384 493 289 353 406 487
```

```
print(*right_line_x)
```

```
483 878 647 716 704 806 484 812 830 899 479 538 673 752 537 623
```

## 2. Creating a Single Linear Representation of each Line Group

- Begin at the bottom of the image
- End just below the horizon

→ Need to be a linear fit(feed y and find x values)

- polyfit() : 데이터의 기울기와 절편 추정
- Ploy1d() : 위의 계수를 이용하여 선형 방정식 생성

```
min_y = image.shape[0] * (3 / 5)
max_y = image.shape[0]
```

```
poly_left = np.poly1d(np.polyfit(left_line_y, left_line_x, deg = 1)) # 1차원 직선 선형 모델 생성
left_x_start = int(poly_left(max_y)) # x 시작점 추정
left_x_end = int(poly_left(min_y)) # x 끝점 추정

poly_right = np.poly1d(np.polyfit(right_line_y, right_line_x, deg = 1))
right_x_start = int(poly_right(max_y))
right_x_end = int(poly_right(min_y))
```

```
left_x_start, left_x_end, right_x_start, right_x_end
```

```
(183, 461, 895, 508)
```