로봇 공학 개론

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

자율주행

- OpenCV 라이브러리
  - 실시간 컴퓨터 비전을 목적으로 한 프로그래밍 오픈소스 라이브러리
  - 초기 인텔이 개발
  - 객체, 얼굴, 모션(행동) 인식 등의 응용 프로그램에서 사용
  - 자율주행의 영상처리 기술

공식 홈페이지 https://opencv.org/

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

이미지 전처리

- 차선 인식을 위한 기본적인 전처리

  - GrayScale 변환

  - Blur (Smothing)

  - Edge 검출

전처리 – GrayScale

- 이미지 불러오기

```
import cv2  # opencv 사용
import numpy as np

image = cv2.imread('Line_image.jpg') # 이미지 읽기
cv2.imshow('result', image) # 이미지 출력
cv2.waitKey(0)
```

Cv2.imread (‘파일 이름’)

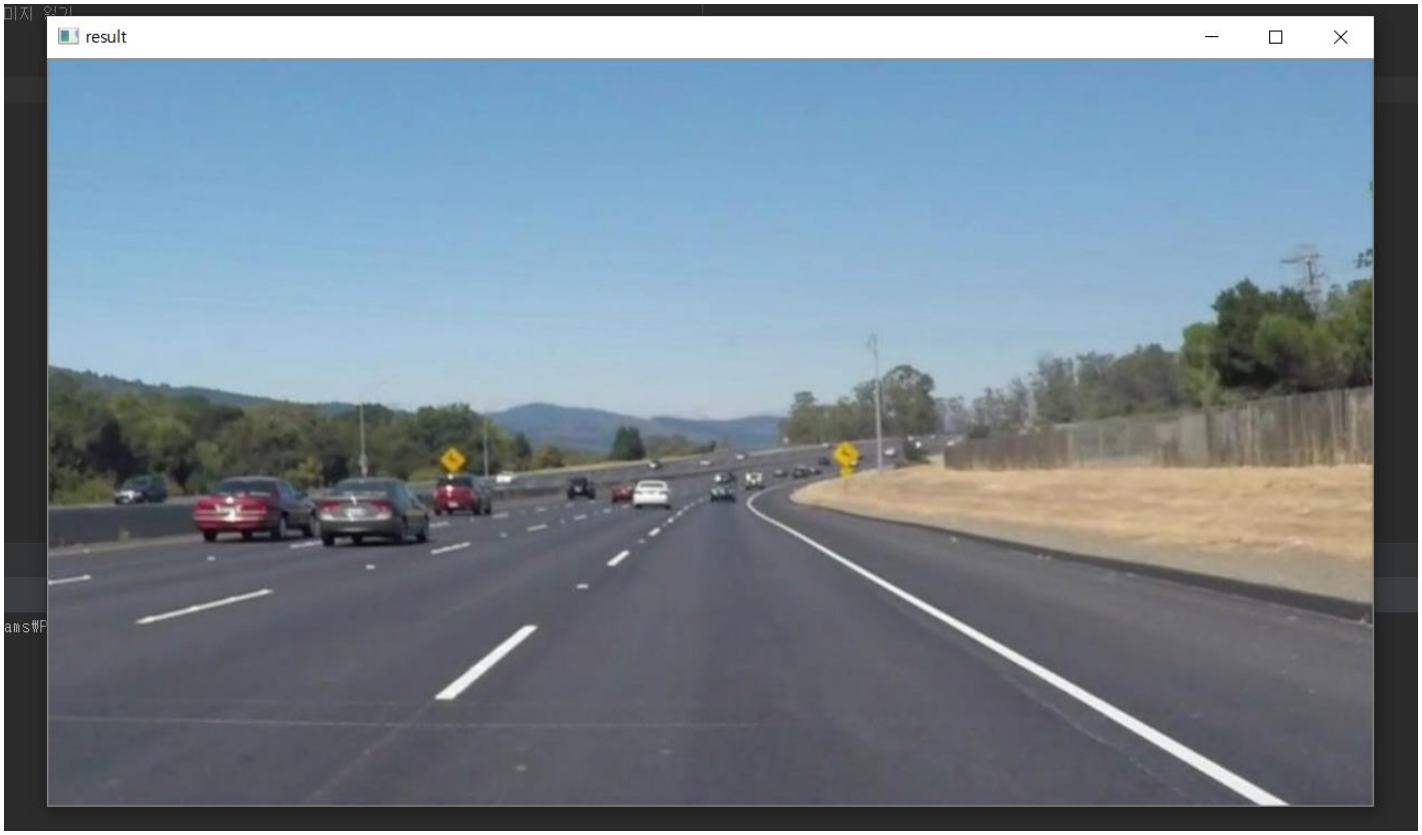
Cv2.imshow (‘팝업 창 이름’ , 이미지 변수)

Cv2.waitKey(양수)

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

전처리 – GrayScale

- 이미지 불러오기



전처리 - GrayScale

```python
import cv2
import numpy as np

image = cv2.imread('Line_image.jpg')

gray_img = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

cv2.imshow('result', image)
cv2.imshow('gray', gray_img)

cv2.waitKey(0)
```

- 컬러 변환 함수

Cv2.cvtColor(“변환할 이미지” , 변환 색상)

cv2.COLOR_RGB2GRAY :

RGB 를 Gray 이미지로 변환

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

전처리 - GrayScale

전처리 – Blur

```python
import cv2
import numpy as np

image = cv2.imread('Line_image.jpg')

gray_img = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

blur_img = cv2.GaussianBlur(gray_img, (3, 3), 0)

cv2.imshow('result', image)
cv2.imshow('gray', gray_img)
cv2.imshow('blur', blur_img)

cv2.waitKey(0)
```

cv2.GaussianBlur(‘0이미지 변수’ , (커널 크기), SD)

커널 크기는 양수인 홀수 쌍

SD : standard deviation X

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

전처리 – Blur

img_input

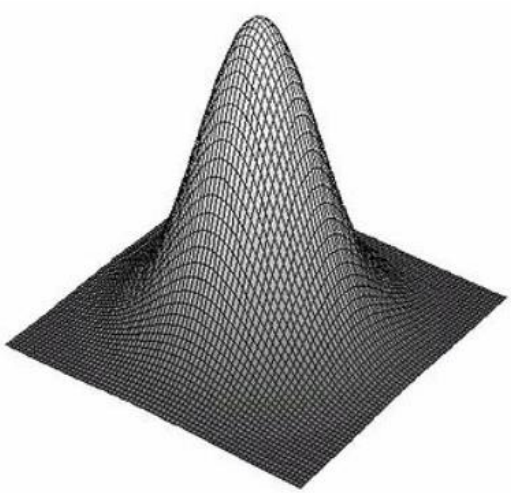
(0,0) (0,0)	(0,1) (0,1)	(0,2) (0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
(1,0) (1,0)	(1,1) (1,1)	(1,2) (1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
(2,0) (2,0)	(2,1) (2,1)	(2,2) (2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
(6,0)	(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
(7,0)	(7,1)	(7,2)	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)
(8,0)	(8,1)	(8,2)	(8,3)	(8,4)	(8,5)	(8,6)	(8,7)

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

전처리 – Blur

$$\frac{1}{273}$$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1



img_input

(0,0) (0,0)	(0,1) (0,1)	(0,2) (0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
(1,0) (1,0)	(1,1) (1,1)	(1,2) (1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
(2,0) (2,0)	(2,1) (2,1)	(2,2) (2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
(6,0)	(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
(7,0)	(7,1)	(7,2)	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)
(8,0)	(8,1)	(8,2)	(8,3)	(8,4)	(8,5)	(8,6)	(8,7)

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

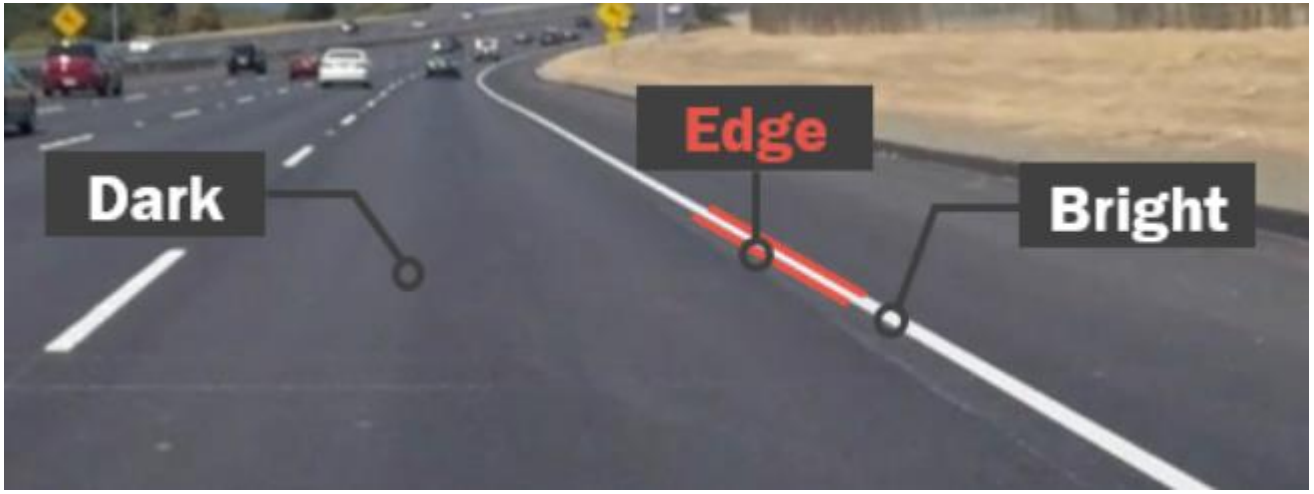
백색노이즈가 있는 이미지

가우시안 필터링을 적용한 이미지

전처리 – Blur

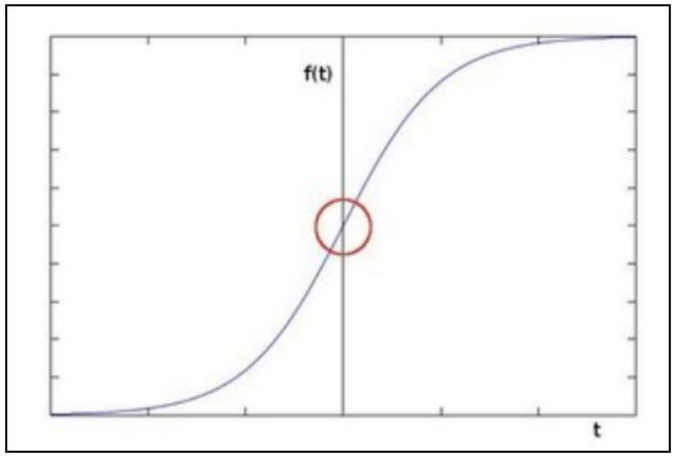
Canny Edge

- **Edge**

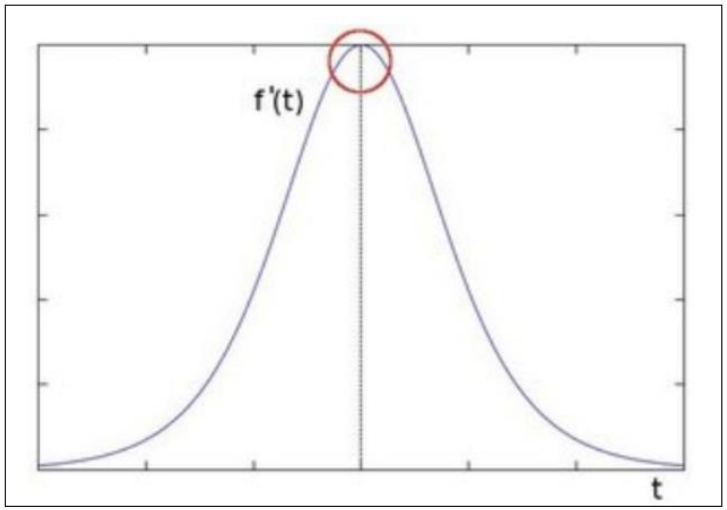


- 픽셀값이 급격하게 변하는 지점
- 물체에 경계선을 추출하기 위한 기술
- 알고자 하는 물체의 모양, 크기, 위치 등 정보를 확인하고자 할 때 사용되는 기술

Canny Edge

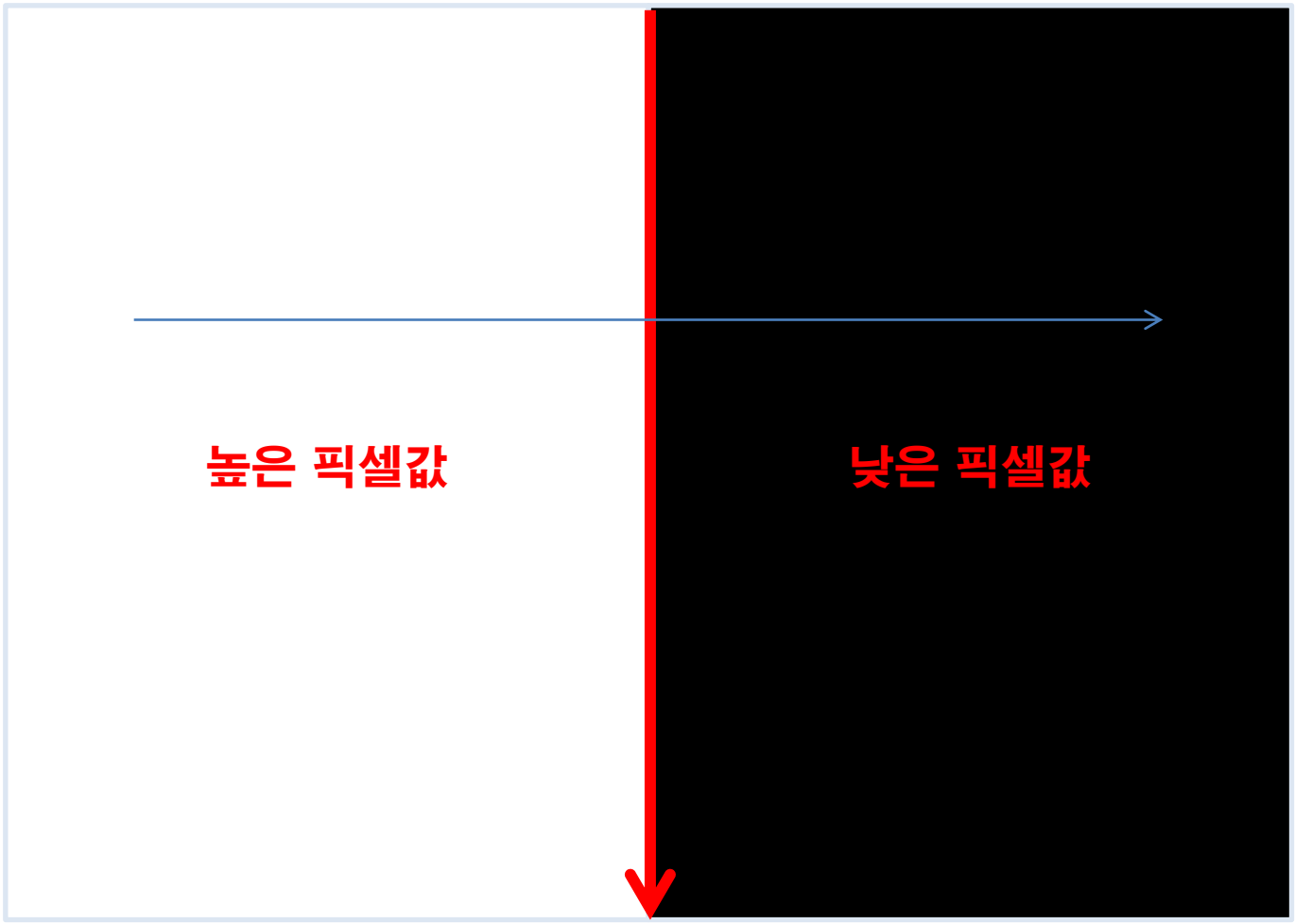


1차 미분



CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

Canny Edge



Canny Edge

```python
import cv2
import numpy as np

image = cv2.imread('Line_image.jpg')

gray_img = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

blur_img = cv2.GaussianBlur(gray_img, (3, 3), 0)

canny_img = cv2.Canny(blur_img, 70, 210)




cv2.imshow('result',image)
cv2.imshow('gray',gray_img)
cv2.imshow('blur',blur_img)
cv2.imshow('canny', canny_img)

cv2.waitKey(0)
```

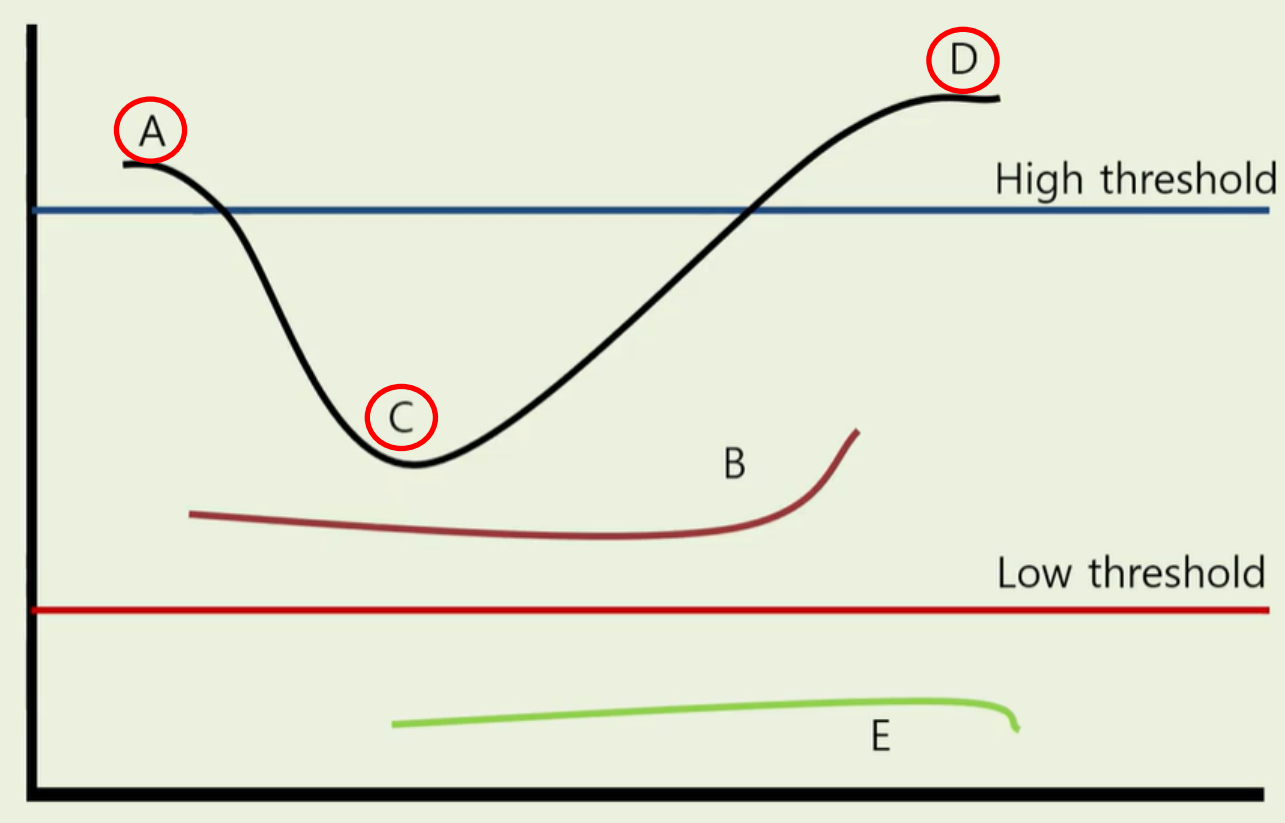
존 F 캐니에 의해 고안됨

cv2.Canny(‘이미지 변수’ ,Low_thres, High_thres)

보통 Low 와 High 는 1:2, 1:3

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

Canny Edge



-High threshold 값 이상의 픽셀만 Edge로 검출

-Low 와 High 사이 값이면 주변에
Edge로 판정된 이웃이 있을때 Edge 로 검출

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

Canny Edge

ROI (Region of Interest)

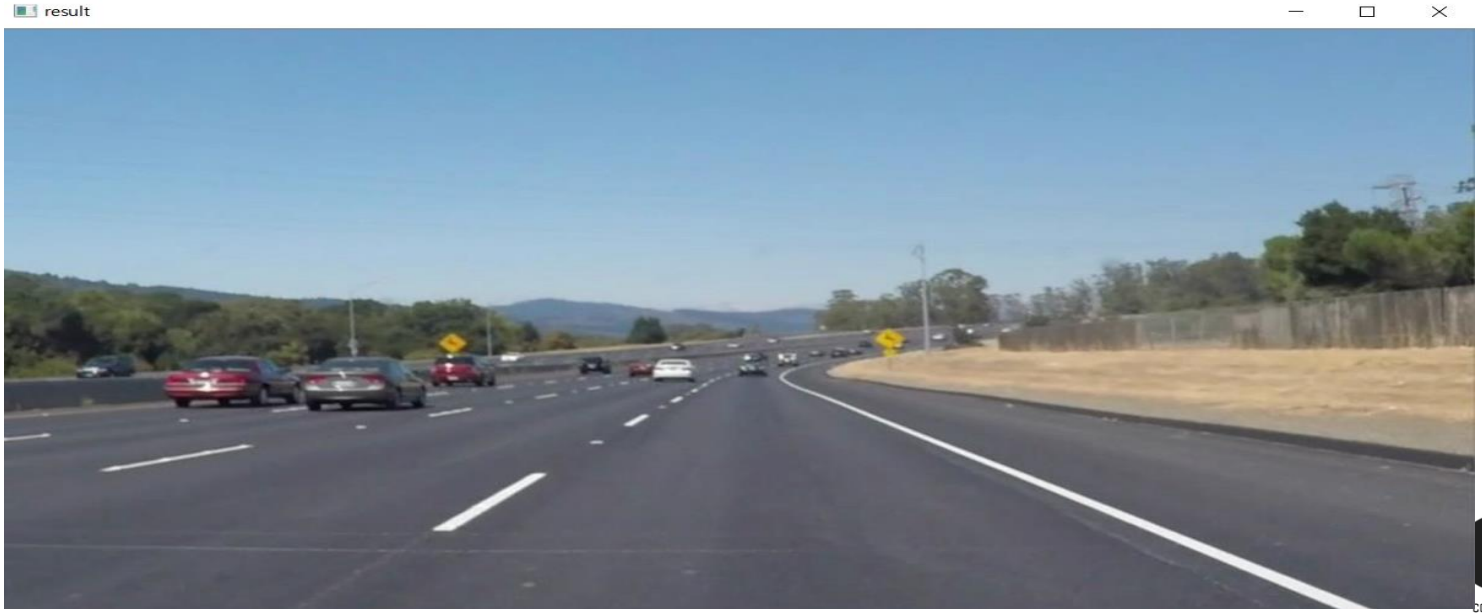
이미지 내에서 관심 지역만 추출

1) 관심 지역 선택 (다각형 모양)

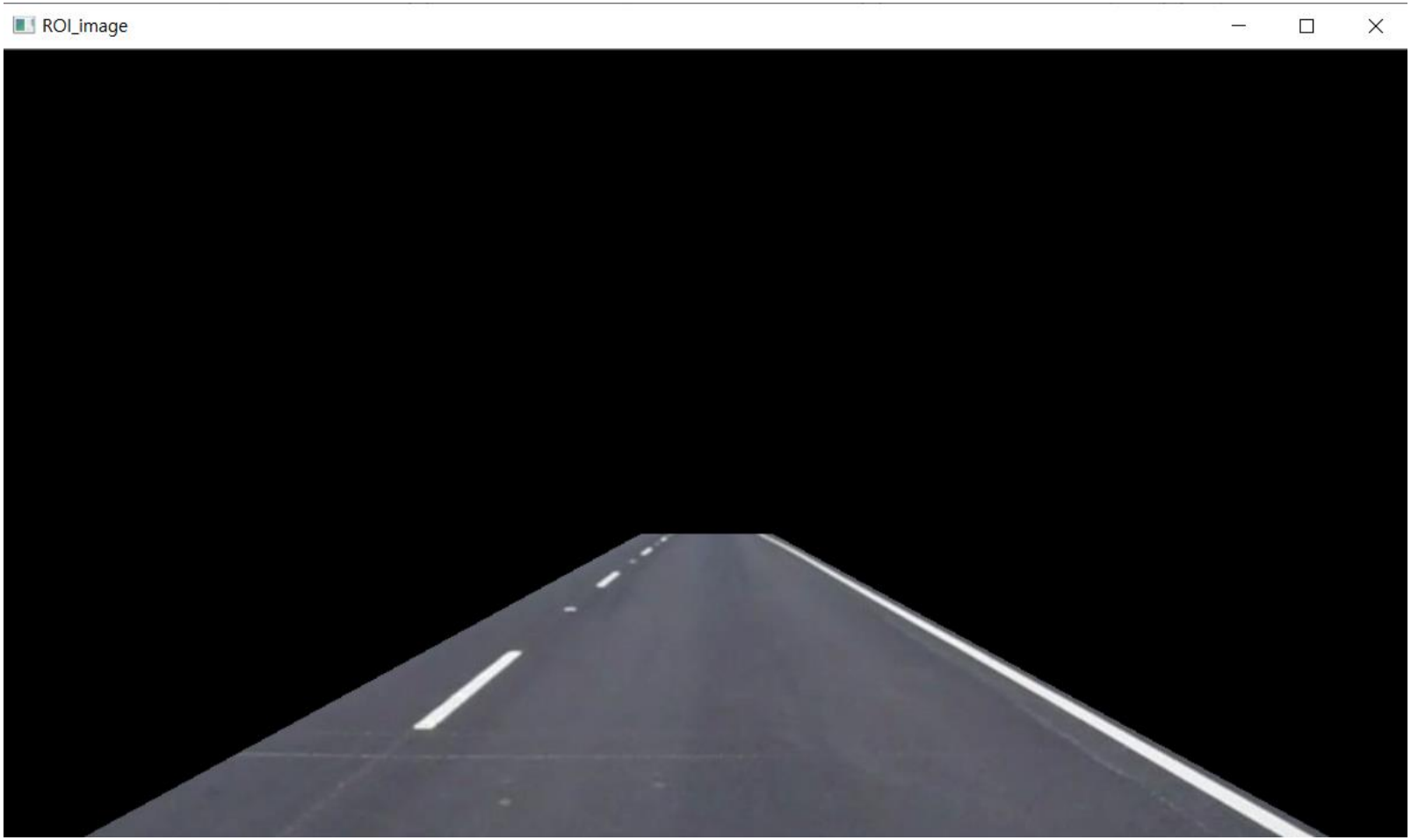
2) 관심 지역 mask 로 만듦

3) 추출한 mask 와 원본 이미지 Bit연산

ROI (Region of Interest)



ROI (Region of Interest)



```
height, width = image.shape[:2]
vertices = np.array([[(50,height),(width/2-45, height/2+60),
                      (width/2+45, height/2+60), (width-50,height)]], dtype=np.int32)

ROI_img = region_of_interest(canny_img, vertices)
cv2.imshow('ROI_image', ROI_img)
```

image.shape => (540, 960, 3)

이미지 행렬 사이즈

Vertices => 관심 영역 다각형 꼭지점 좌표

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

ROI (Region of Interest)

```python
def region_of_interest(img, vertices, color3=(255, 255, 255), color1=255):

    mask = np.zeros_like(img)

    if len(img.shape) > 2:  # Color 이미지(3채널)라면 :
        color = color3
    else:  # 흑백 이미지(1채널)라면 :
        color = color1

    # vertices에 정한 점들로 이뤄진 다각형부분(ROI 설정부분)을 color로 채움
    cv2.fillPoly(mask, vertices, color)
    cv2.imshow('mask', mask)

    # 이미지와 color로 채워진 ROI를 합침
    ROI_image = cv2.bitwise_and(img, mask)
    return ROI_image
```

cv2.fillPoly(“이미지” , 꼭지점 좌표, 색상)

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

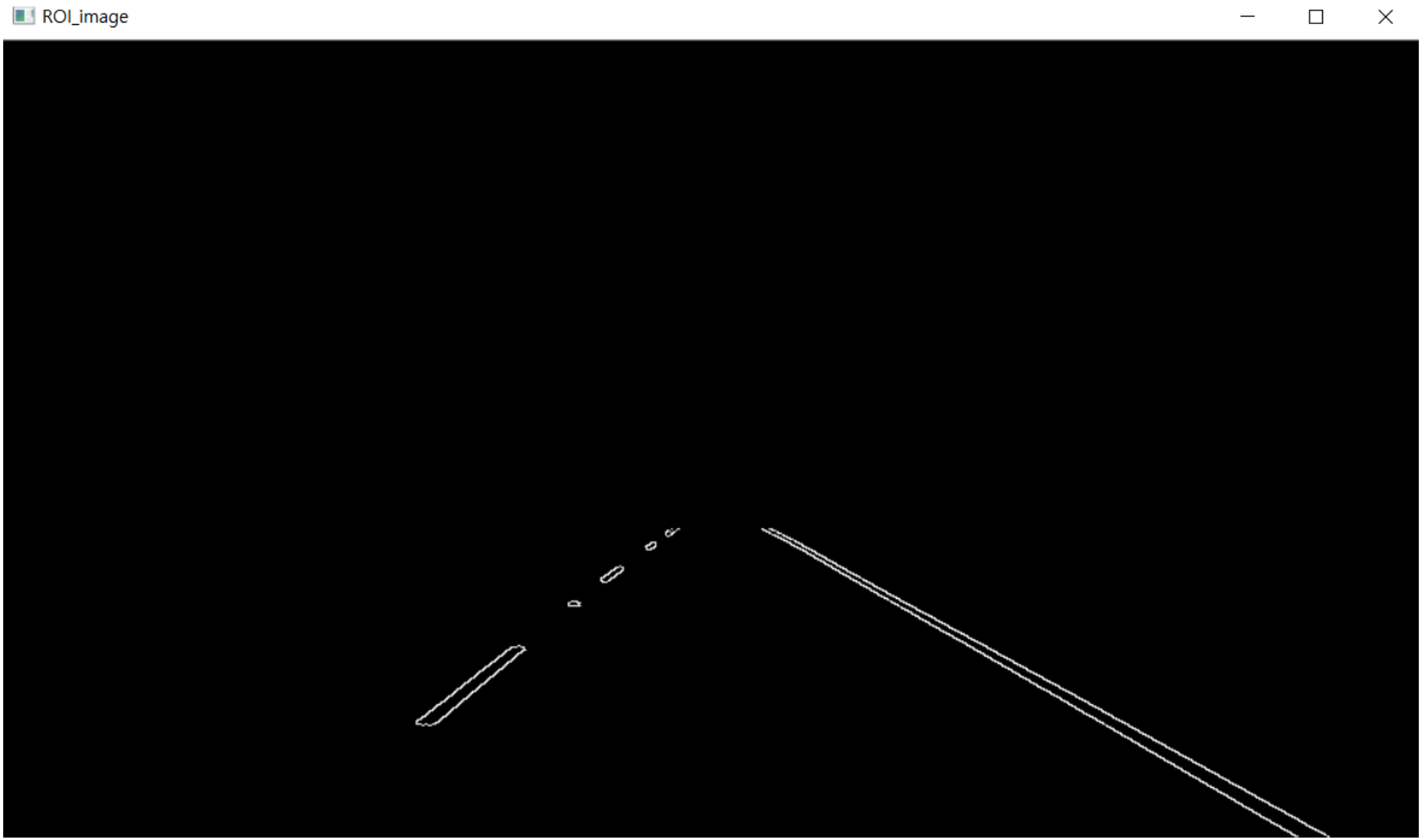
ROI (Region of Interest)

```python
def region_of_interest(img, vertices, color3=(255, 255, 255), color1=255):

    mask = np.zeros_like(img)

    if len(img.shape) > 2:  # Color 이미지(3채널)라면 :
        color = color3
    else:  # 흑백 이미지(1채널)라면 :
        color = color1

    # vertices에 정한 점들로 이뤄진 다각형부분(ROI 설정부분)을 color로 채움
    cv2.fillPoly(mask, vertices, color)
    cv2.imshow('mask', mask)

    # 이미지와 color로 채워진 ROI를 합침
    ROI_image = cv2.bitwise_and(img, mask)
    return ROI_image
```

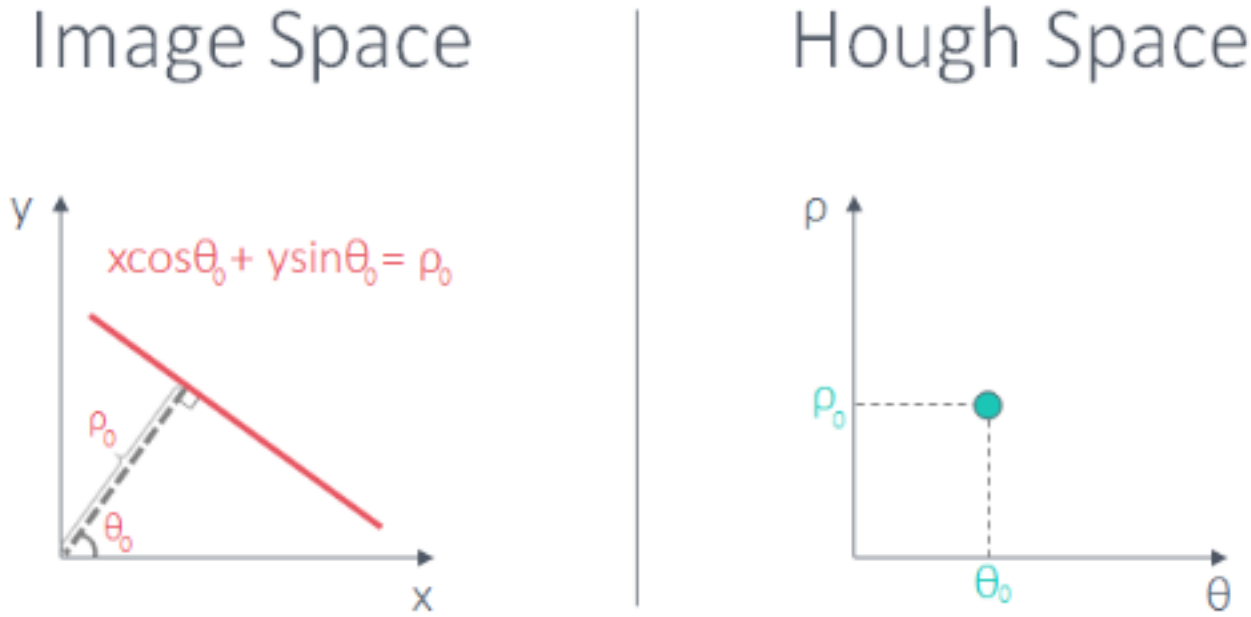
cv2.bitwise_and(“이미지” , “mask”)
: 이미지와 mask 의 bit 연산
이미지와 mask 모두 값이 있는 픽셀 and 연산

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

ROI (Region of Interest)



Hough Tranform



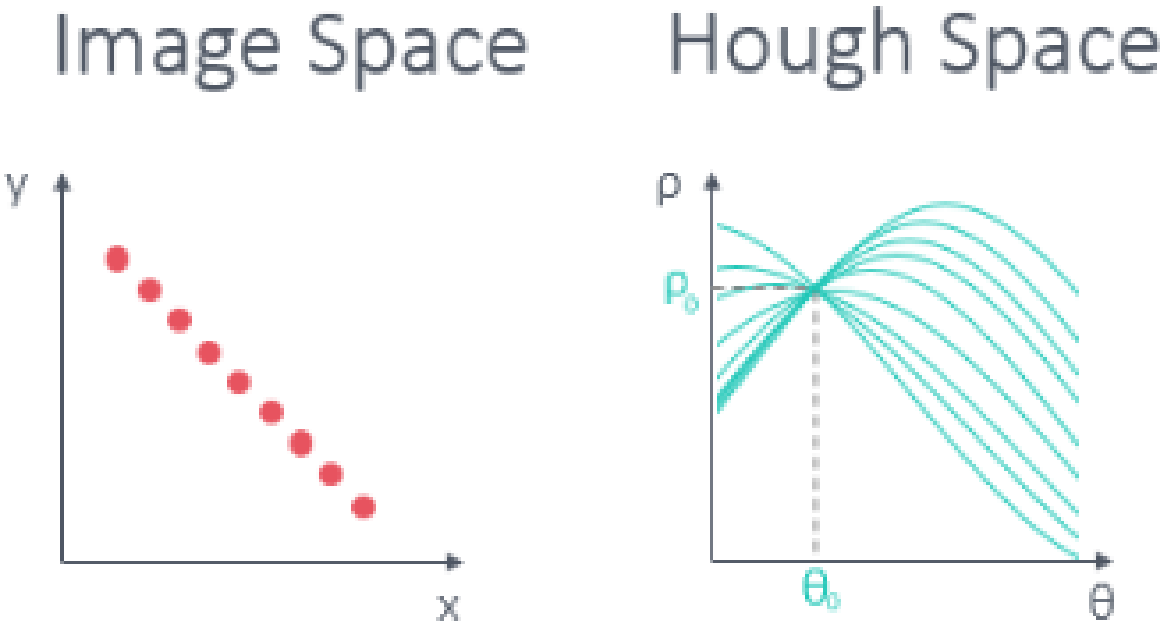
ρ = 원점에서 직선까지의 거리
θ = x 축으로부터의 각도

X,Y -> ρ, θ

Hough Transform : 직선 검출을 위한 허프 공간으로 좌표계 변환

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

Hough Tranform



OpenCV 선분 검출 함수

cv2.HoughLinesP(edges_img, rho, theta, threshold,
minLineLength, maxLineGap)

cv2.HoughLinesP(edges_img, rho, theta, threshold,
minLineLength, maxLineGap)

- egdes_img : 8bit single-channel binary image,
(canny edge를 선 적용)
- rho : hough space에서 ρ 값을 한번에 얼만큼 증가시키면서 조사할 것인가 , 10이면 ρ 값을 1 씩 증가하면서 검출 , 보통 1
(0 ~ 1 실수)

- theta : hough space에서 θ 값을 한번에 얼만큼 증가시키면서 조사할 것인가 , 10이면 θ 값을 1 라디안씩 증가하면서 검출 ,
보통 1*pi/180

- Threshold : Hough space 에서 교차점의 기준 개수,
교차점 개수가 Threshold 를 넘는 선만 검출

Image Space에서 바꿔 말하면 서로 일직선 위에 있는 점의 수가
threshold 이상인지 아닌지를 판단하는 척도

cv2.HoughLinesP(edges_img, rho, theta, threshold,
minLineLength, maxLineGap)

- **minLineLength : 검출하고자 하는 선의 최소 길이 , 단위는 픽셀**

- **maxLineGap : 선 위의 점들 사이 최대 거리,**
즉, 점 사이의 거리가 이 값보다 크면 다른 선으로 간주

- **Return 값 : 선분의 시작점, 끝점의 좌표**

```
[[[520 330 898 539]]

 [[517 331 877 538]]

 [[629 389 898 538]]

 [[280 461 320 430]]

 [[293 462 353 412]]

 [[288 454 345 410]]

 [[531 338 834 513]]

 [[295 461 354 412]]

 [[290 463 328 432]]

 [[388 382 457 330]]]
```

<< HoughLinesP 결과
[x1 y1 x2 y2]

Hough Tranform

```python
def draw_lines(img, lines, color=[0, 0, 255], thickness=2): # 선 그리기
    for line in lines:
        for x1,y1,x2,y2 in line:
            cv2.line(img, (x1, y1), (x2, y2), color, thickness)

def hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap): # 허프 변환
    lines = cv2.HoughLinesP(img, rho, theta, threshold, minLineLength=min_line_len, maxLineGap=max_line_gap)
    line_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
    draw_lines(line_img, lines)

    return line_img
```

- **draw_line (이미지, 선분의 양 끝점을 가지고 있는 행렬, 선분 색상, 선분 굵기)**
  선분을 그리는 함수
  cv2.line(이미지, 선분 첫 꼭지점, 선분 끝 꼭지점, 색상, 굵기)

- **hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap)**
  허프 변환을 수행하고 이를 검은 화면에 그리는 함수

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

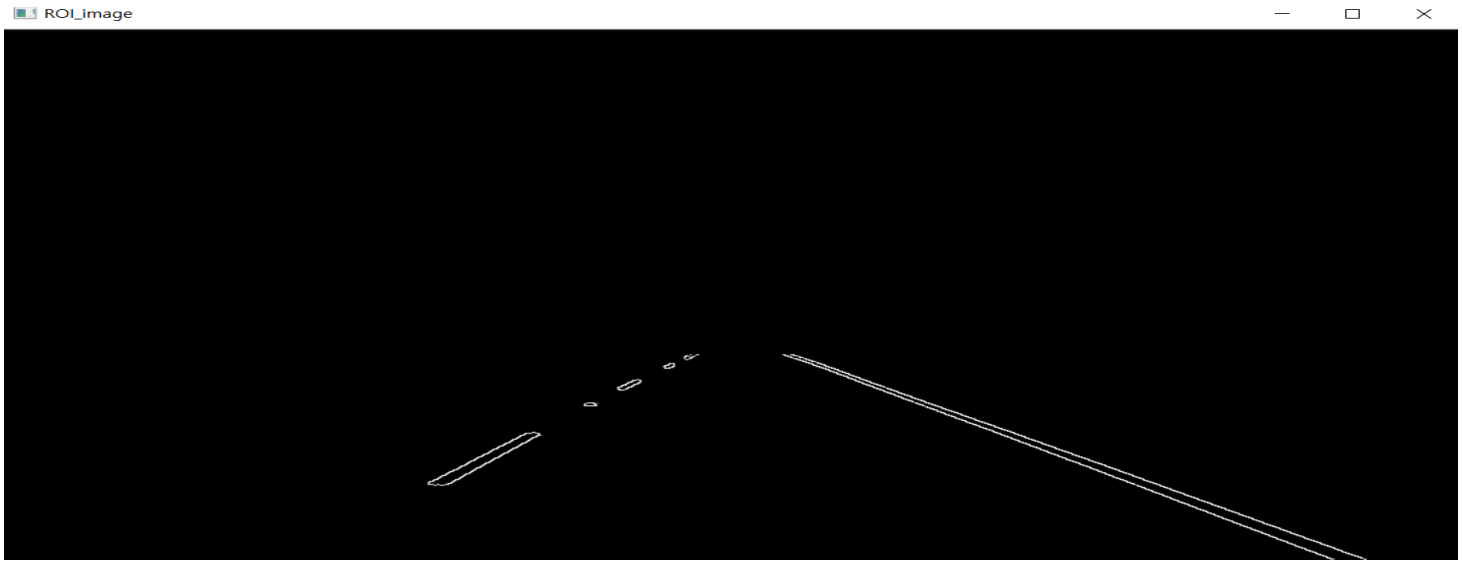
Hough Tranform

```
def draw_lines(img, lines, color=[0, 0, 255], thickness=2): # 선 그리기
    for line in lines:
        for x1,y1,x2,y2 in line:
            cv2.line(img, (x1, y1), (x2, y2), color, thickness)

def hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap): # 허프 변환
    lines = cv2.HoughLinesP(img, rho, theta, threshold, minLineLength=min_line_len, maxLineGap=max_line_gap)
    line_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
    draw_lines(line_img, lines)

    return line_img
```

```
ROI_img = region_of_interest(canny_img, vertices)

hough_img = hough_lines(ROI_img, 1, 1 * np.pi/180, 30, 10, 20)# 허프 변환

cv2.imshow('result',image)
cv2.imshow('gray',gray_img)
cv2.imshow('blur',blur_img)
cv2.imshow('canny', canny_img)
cv2.imshow('ROI_image', ROI_img)
cv2.imshow('hough_image', hough_img)

cv2.waitKey(0)
```

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

Hough Tranform



Hough Tranform

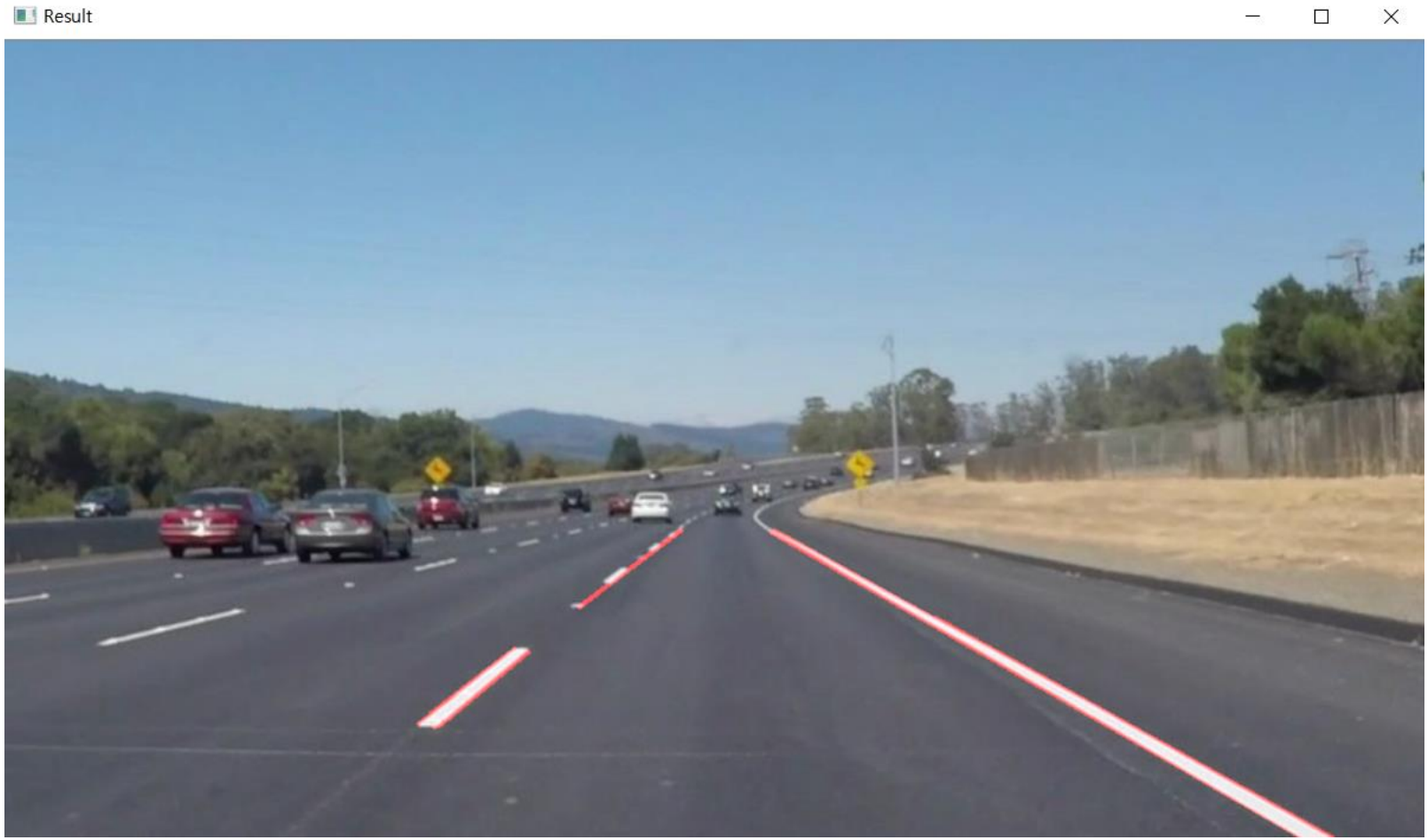
```
def weighted_img(img, img2, a=1, b=1., c=0.): # 두 이미지 operlap 하기
    return cv2.addWeighted(img2, a, img, b, c)
```

- **Cv2.addWeighted (img, a, img2, b, c) 이미지 Overlap 함수**
  new_img = img X a + img2 X b + c

```
hough_img = hough_lines(ROI_img, 1, 1 * np.pi/180, 30, 10, 20) # 허프 변환

result = weighted_img(hough_img, image) # 원본 이미지에 검출된 선 overlap

cv2.imshow('Result', result)
```

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

Hough Tranform



1. grayscale

2. blur

3. canny edge

4. ROI

5. hough transform

6. Result

그림자가 있을 때 차선으로 인식

Filtering (후처리)

- 선분의 기울기
- 각 차선의 대표 라인 추출

Filtering (후처리)

```
def hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap): # 허프 변환
    lines = cv2.HoughLinesP(img, rho, theta, threshold, minLineLength=min_line_len, maxLineGap=max_line_gap)
    #line_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
    #draw_lines(line_img, lines)

    return lines
```

```
line_arr = hough_lines(ROI_img, 1, 1 * np.pi / 180, 30, 10, 20)  # 허프 변환

line_arr = np.squeeze(line_arr)

# 기울기 구하기
slope_degree = (np.arctan2(line_arr[:, 1] - line_arr[:, 3], line_arr[:, 0] - line_arr[:, 2]) * 180) / np.pi
```

- line_arr 의 shape : (75, 1, 4)
- squeeze 함수 : 차원 중 사이즈가 1인 것을 찾아 해당 차원을 제거
- Squeeze 후 line_arr 의 shape : (75, 4)

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

Filtering (후처리)

```
def hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap): # 허프 변환
    lines = cv2.HoughLinesP(img, rho, theta, threshold, minLineLength=min_line_len, maxLineGap=max_line_gap)
    #line_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
    #draw_lines(line_img, lines)

    return lines
```

```
line_arr = hough_lines(ROI_img, 1, 1 * np.pi / 180, 30, 10, 20)  # 허프 변환

line_arr = np.squeeze(line_arr)

# 기울기 구하기
slope_degree = (np.arctan2(line_arr[:, 1] - line_arr[:, 3], line_arr[:, 0] - line_arr[:, 2]) * 180) / np.pi
```

Arctan2 함수를 이용해서 선분의 각도 구함
Arctan2 함수의 결과는 라디안 이기 때문에 180/pi 를 곱해줘서 degree 변환

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

Filtering (후처리)

Line_arr

[[10,20,54,55]
[74,20,168,328]
[123,34,42,34]

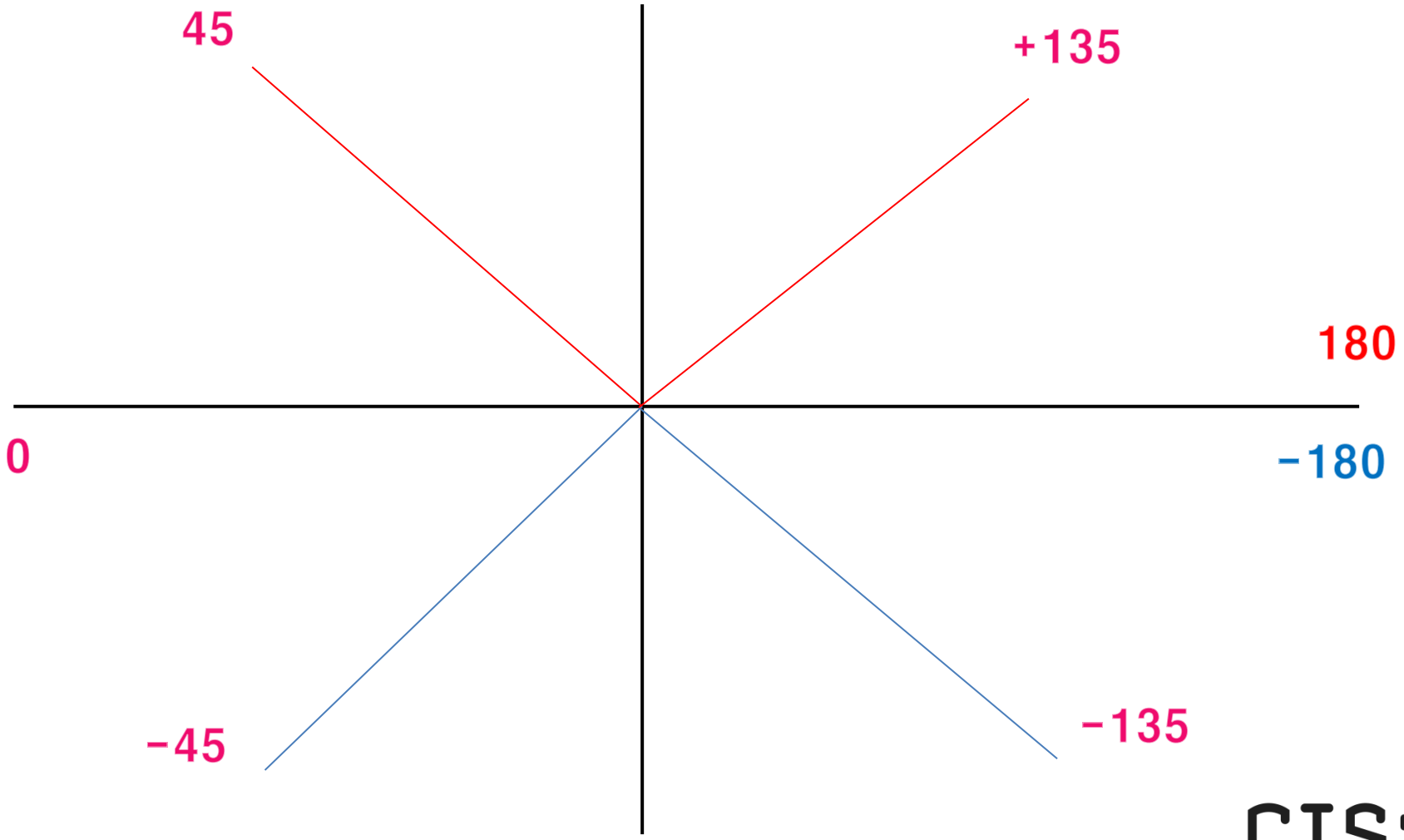
...

Slope_degree

[178
102
-124

...

Filtering (후처리)



```
[ 172.01067323  174.55966797  173.10757688  149.2259639  -172.02323121
  172.63194893  150.00492038  180.          177.51044708  175.97173633
 -171.21883727  176.30861401 -172.17092349  149.93141718  169.9920202
  178.47247456  175.38935068  180.          175.03025927 -173.48019825
  174.92039214  148.706961    150.46121774  175.15599962  176.18592517
 -177.51044708 -172.2781742   169.9920202   174.05996275  178.31531568
  180.         -168.11134196 -176.9872125   149.83270439  180.
  176.02750406  151.18920626 -171.02737339 -171.86989765 -173.15722659
  173.08877288  149.09561731  173.21102543  177.04922089  174.80557109
 -176.82016988 -175.36453657  159.04422327 -177.95459151  174.55966797
  180.          172.87498365  174.28940686  180.          172.69424047
 -147.99461679  177.27368901  176.18592517  173.0656511   172.97160376
 -174.98688624  177.13759477  148.85141901  172.87498365 -146.82148834
  172.40535663  178.60281897 -175.71084667  178.31531568 -174.05313695
 -175.7636052   168.99645915  174.28940686  175.91438322 -141.95295747]
```

Arctan2 함수를 이용해서 구한 선분의 각도 값들 (slope_degree)

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

Filtering (후처리)

```
# 수평 기울기 제한
line_arr = line_arr[np.abs(slope_degree) < 160]
slope_degree = slope_degree[np.abs(slope_degree) < 160]

# 수직 기울기 제한
line_arr = line_arr[np.abs(slope_degree) > 95]
slope_degree = slope_degree[np.abs(slope_degree) > 95]
```

line_arr = line_arr[np.abs(slope_degree) < 160]

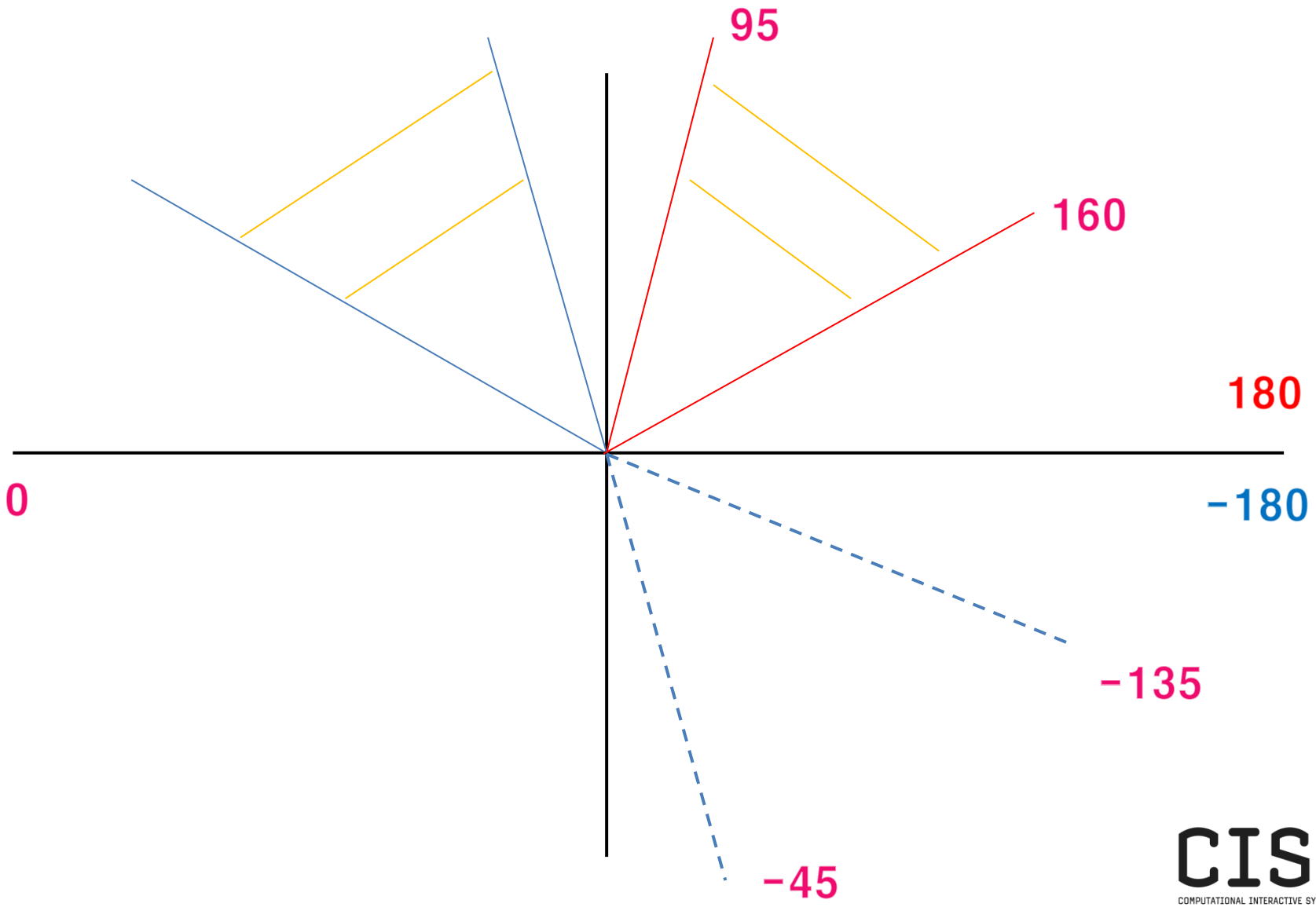
선분 기울기의 절대값이 160 도 보다 작은 선분만 필터링

line_arr = line_arr[np.abs(slope_degree) > 95]

선분 기울기의 절대값이 95도 보다 큰 선분만 필터링

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

Filtering (후처리)



Filtering (후처리)

```
# 필터링된 직선 버리기
L_lines, R_lines = line_arr[(slope_degree > 0), :], line_arr[(slope_degree < 0), :]
temp = np.zeros((image.shape[0], image.shape[1], 3), dtype=np.uint8)
L_lines, R_lines = L_lines[:, None], R_lines[:, None]

# 직선 그리기
draw_lines(temp, L_lines)
draw_lines(temp, R_lines)

result = weighted_img(temp, image)  # 원본 이미지에 검출된 선 overlap

cv2.imshow('Result', result)
```

선분의 기울기가 0보다 큰 선분들은 왼쪽 차선,
선분의 기울기가 0보다 작은 선분들은 오른쪽 차선으로 분류

Filtering (후처리)

```
# 필터링된 직선 버리기
L_lines, R_lines = line_arr[(slope_degree > 0), :], line_arr[(slope_degree < 0), :]
temp = np.zeros((image.shape[0], image.shape[1], 3), dtype=np.uint8)
L_lines, R_lines = L_lines[:, None], R_lines[:, None]

# 직선 그리기
draw_lines(temp, L_lines)
draw_lines(temp, R_lines)

result = weighted_img(temp, image)  # 원본 이미지에 검출된 선 overlap

cv2.imshow('Result', result)
```

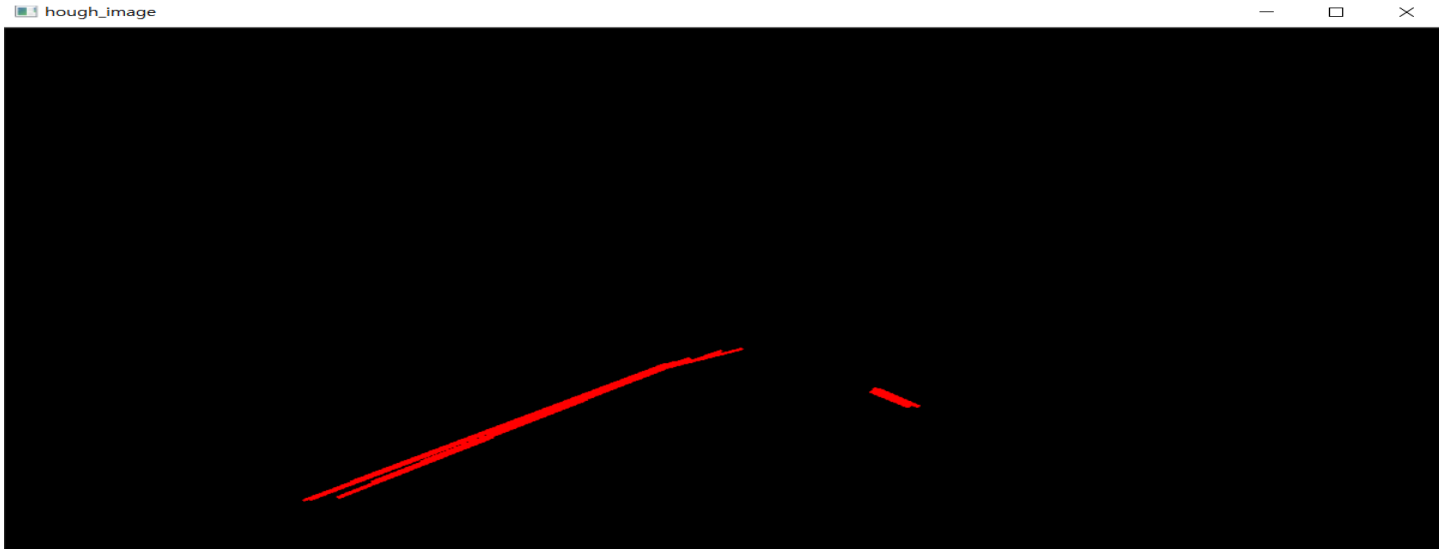
temp = np.zeros((image.shape[0], image[1], 3), dtype= np.uint9)

L_lines, R_lines = L_lines[:, None] , R_lines[:, None]

(행, 1, 4) (행, 4)

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS

Filtering (후처리)



Camera

```python
import cv2

cap = cv2.VideoCapture(1)

cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)  #cv2.CAP_PROP_FRAME_WIDTH
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)  #cv2.AP_PROP_FRAME_HEIGHT

while (cap.isOpened()):
    ret, image = cap.read()

    cv2.imshow('image', image)


    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

- cap = cv2.VideoCapture(1)
  : 비디오(camera) 캡쳐 객체 생성 (안에 숫자는 카메라 인덱스)
- cap.set (cv2.CAP_PROP_FRAME_WIDTH , 320)
  cap.set (cv2.CAP_PROP_FRAME_HEIGHT , 240)
  : 비디오(camera) 객체 설정. 추가 파라미터 설정 가능

CIS:
COMPUTATIONAL INTERACTIVE SYSTEMS