



OPENCV

김규현

```
import cv2
import numpy as np
```

```
trap_top_width_p1 = 0.45
trap_top_width_p2 = 0.58
trap_bottom_width_p1 = 0.0
trap_bottom_width_p2 = 1.0
trap_height_p1 = 0.6
trap_height_p2 = 1.0
```

자동차의 진행 방향 바닥에 존재하는 차선
만을 검출 하기 위한 관심영역 값 지정

```
rho = 10
theta = 1 * np.pi / 180
threshold = 50
min_line_length = 10
max_line_gap = 30
```

차선의 외곽선을 기준으로 선을 추출하
기 위한 값 지정

```
def draw_lines(img, lines, color=[0,0,255], thickness = 7):
    # 예외처리
    if (lines is None) or (len(lines) == 0):
        return

    # 왼쪽, 오른쪽 라인을 그릴것인지 체크
    left_draw_check = True
    right_draw_check = True

    # 모든선(lines)의 기울기를 체크해서 불필요한 선 제거
    # 기울기 임계값 보다 기울기가 작은 선은 제거
    slope_threshold = 0.5 # 기울기 30도
    slopes=[]
    new_lines=[]

    # 기울기가 작은 라인 제거
    # lines 변수는 (N, 1, 4)차원 형태를 가짐.
    for line in lines:
        x1,y1,x2,y2 = line[0] # line = [[x1,y1,x2,y2]]
```

기울기 계산

```
if x2-x1 == 0.:
    slope = 999    # 기울기가 999이면 선이 거의 수직에
                  # 가깝고 경사도가 매우 높다는 의미
else:
    slope = (y2-y1)/(x2-x1)
if abs(slope) > slope_threshold:
    slopes.append(slope)
    new_lines.append(line)
```

lines = new_lines # 조건을 만족하는 line만 걸러냄

오른쪽 / 왼쪽 라인 분리

기울기 및 선을 구성하는 두점이 영상의 가운데를 기준으로 좌우에 분포하는지 체크

left_lines = []

right_lines = []

```
for i, line in enumerate(lines):
```

```
    x1, y1, x2, y2 = line[0] # line = [[x1,y1,x2,y2]]
```

```
    img_center = img.shape[1]/2 # width
```

```
    # 기울기 방향이 바뀜 : y의 좌표가 위에서 아래로 내려옴
```

```
if slopes[i] > 0 and x1 > img_center and x2 > img_center: # right
    right_lines.append(line)
elif slopes[i] < 0 and x1 < img_center and x2 < img_center: # left
    left_lines.append(line)
```

LEFT / RIGHT 라인을 구성하는 점들을 사용해서 np.polyfit을 적용
np.polyfit에 사용될 점으로 추가

LEFT 찾기

left_lines_x = []

left_lines_y = []

```
for line in left_lines:
```

```
    x1,y1,x2,y2 = line[0] # line = [[x1,y1,x2,y2]]
```

```
    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) #  $y = mx + b$ 
else:
    left_m, left_b = 1, 1 # np. Polyfit : 주어진 데이터에 대해 최소 제곱을 갖는 다항식 값을 반환
    left_draw_check = False

# RIGHT 찾기
right_lines_x = []
right_lines_y = []

for line in right_lines:
    x1, y1, x2, y2 = line[0] # line = [[x1,y1,x2,y2]]
    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) #  $y = mx + b$ 
else:
```

```
right_m, right_b = 1, 1
right_draw_check = False
```

```
# 왼쪽과 오른쪽의 각각 2개의 점 찾기
# y값은 알고 있음, x값만 찾으려면 됨
y1 = int(img.shape[0]) # height
y2 = int(img.shape[0]*trap_height_p1)
```

```
right_x1 = int((y1-right_b) / right_m)
right_x2 = int((y2-right_b) / right_m)
left_x1 = int((y1-left_b) / left_m)
left_x2 = int((y2-left_b) / left_m)
```

```
# 차선그리기
```

```
if right_draw_check:
    cv2.line(img, (right_x1, y1), (right_x2, y2), color, thickness)
if left_draw_check:
    cv2.line(img, (left_x1, y1), (left_x2, y2), color, thickness)
```

```
trap_top_width_p1 = 0.45
trap_top_width_p2 = 0.58
trap_bottom_width_p1 = 0.0
trap_bottom_width_p2 = 1.0
trap_height_p1 = 0.6
trap_height_p2 = 1.0
```

```
capture = cv2.VideoCapture('mp4/driving_speed.mp4')
### 비디오 저장을 위한 코덱 지정
codec = cv2.VideoWriter_fourcc('m','p','4','v') # .mp4
# codec = cv2.VideoWriter_fourcc(*'XVID') # .avi

##### 동영상에 대한 정보를 획득
width = int(capture.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(capture.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(capture.get(cv2.CAP_PROP_FPS))

pts = np.array([[int(width*trap_bottom_width_p1),int(height*trap_height_p2)],
                [int(width*trap_top_width_p1),int(height*trap_height_p1)],
                [int(width*trap_top_width_p2),int(height*trap_height_p1)],
                [int(width*trap_bottom_width_p2),int(height*trap_height_p2)]],dtype=np.int32)
```

비디오 저장을 위한 파라미터 적용

```
video = cv2.VideoWriter("mp4/result.mp4", codec, fps, (width,height))
```

```
lower_yellow = (20, 120, 120)
```

```
upper_yellow = (40, 255, 255)
```

```
lower_white = (200, 200, 200)
```

```
upper_white = (255, 255, 255)
```

```
# img_mask = np.zeros((height,width,3), dtype=np.uint8)
```

```
# img_mask = cv2.fillPoly(img_mask,[pts], (255,255,255))
```

```
#cv2.imshow("mask",img_mask)
```

```
img_mask = np.zeros((height,width), dtype=np.uint8)
```

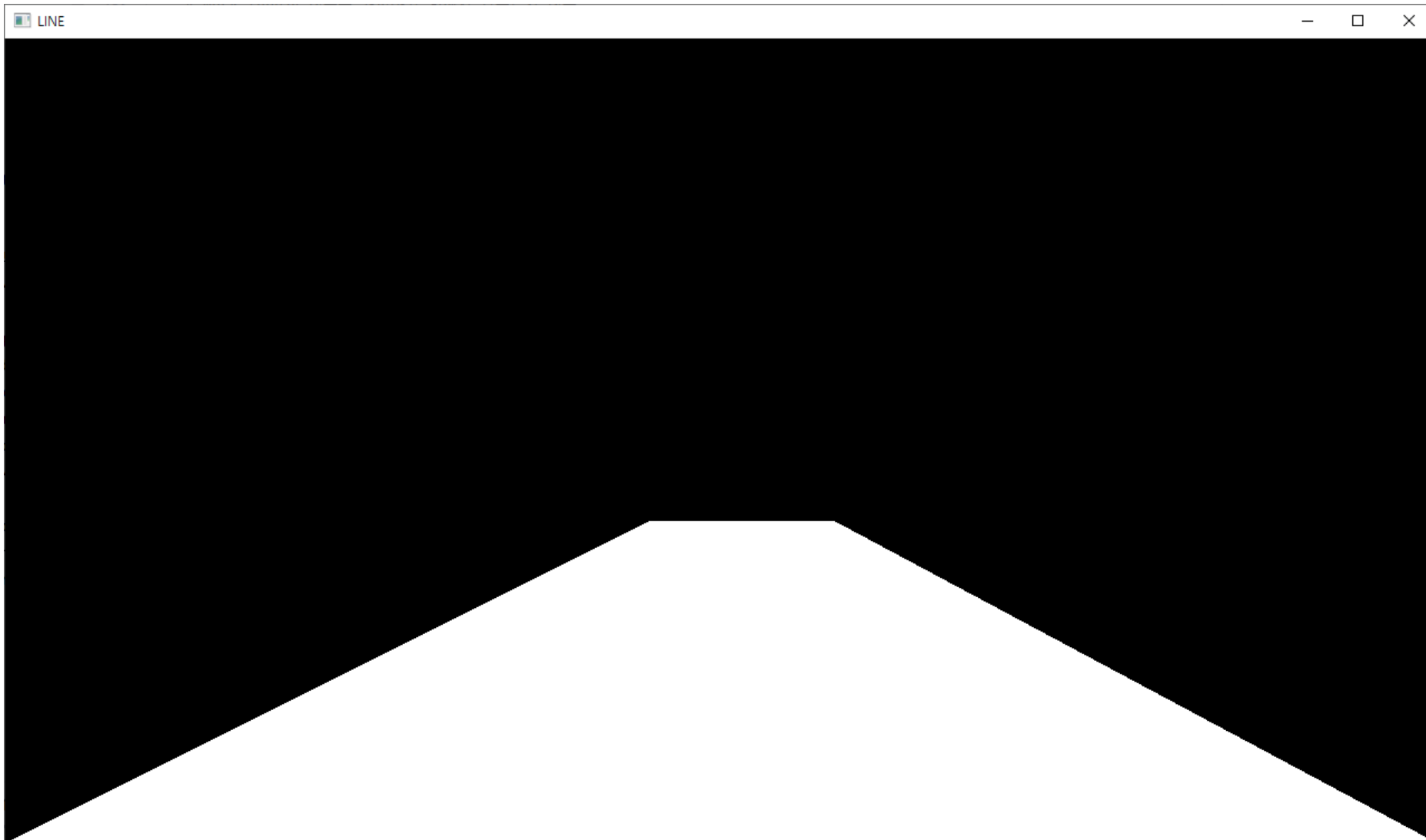
```
img_mask = cv2.fillPoly(img_mask,[pts], 255)
```

```
#cv2.imshow("mask",img_mask)
```

cv2.fillpoly : 좌표들의 list 인 pts의 좌표들을 순서대로 연결하여 내부가 채워진 다각형을 그림

OPENCV

Def draw_lines()



```
while True:
    ret, frame = capture.read()
    if ret == False:
        print('동영상 종료')
        break

    frame_bgr = frame.copy()
    # frame_bgr = cv2.bitwise_and(frame, img_mask)
    ##### 1. HSV로 변환 #####
    frame_hsv = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2HSV)

    mask_yellow = cv2.inRange(frame_hsv, lower_yellow, upper_yellow) # yellow는 hsv에서 적용
    mask_white = cv2.inRange(frame_bgr, lower_white, upper_white) # white는 bgr에서 적용

    mask_lane = cv2.addWeighted(mask_yellow, 1.0, mask_white, 1.0, 0.0)
    # mask_lane의 성능을 개선하기 위해서 모폴로지 적용
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
    mask_lane = cv2.morphologyEx(mask_lane, cv2.MORPH_CLOSE, kernel, iterations=2) # mask_lane → (720,1280,1)
```

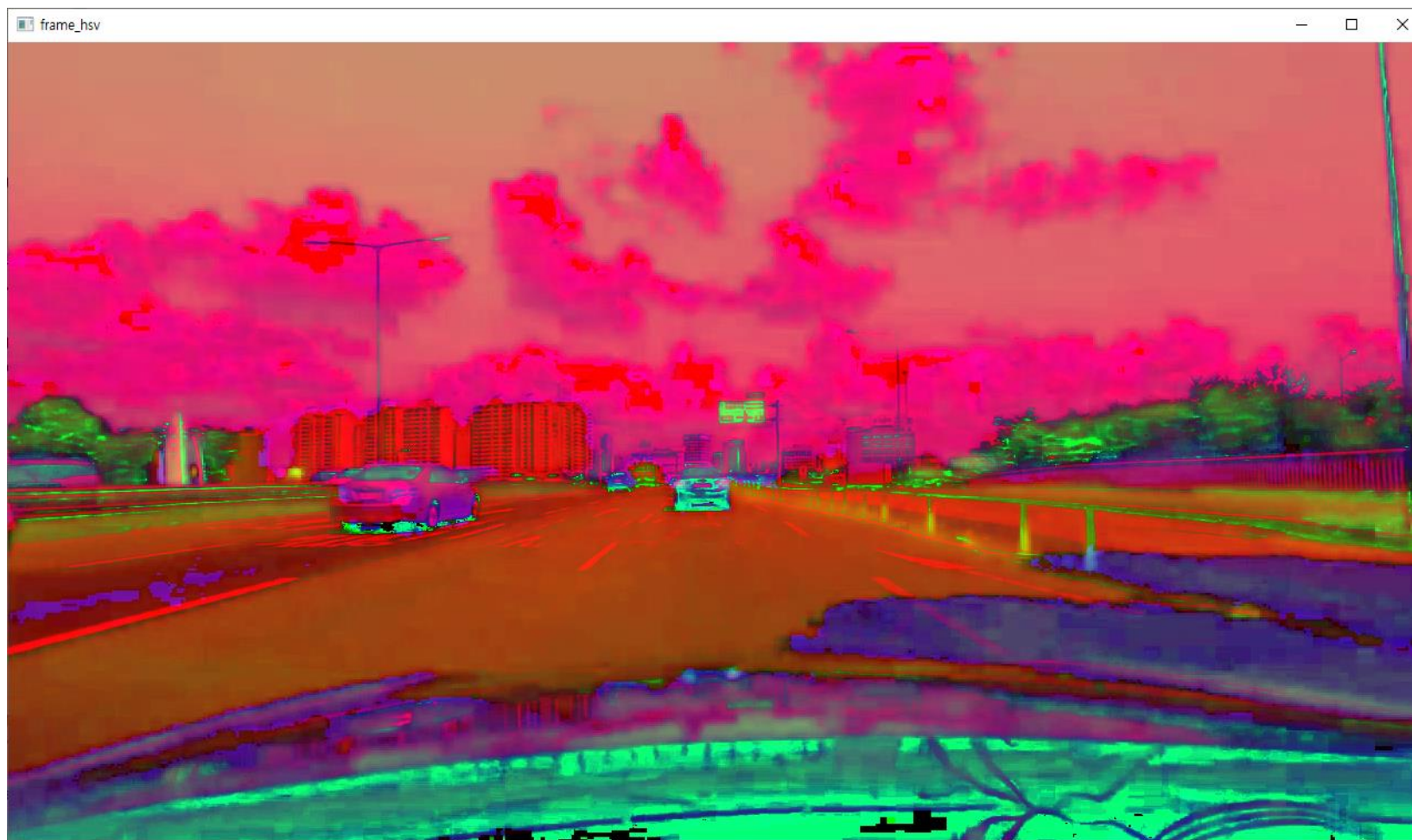
ret : 프레임을 사용 유무에 따라 True 또는 False를 반환하는 Bool 변수

색상(Hue), 채도(Saturation), 명도(Value)

cv2.addWeighted(src1, alpha, src2, beta, gamma, :
두개의 이미지에 각각의 가중치를 조절하여 덧셈

cv2.getStructuringElement(cv2.MORPH_RECT,(5,5))

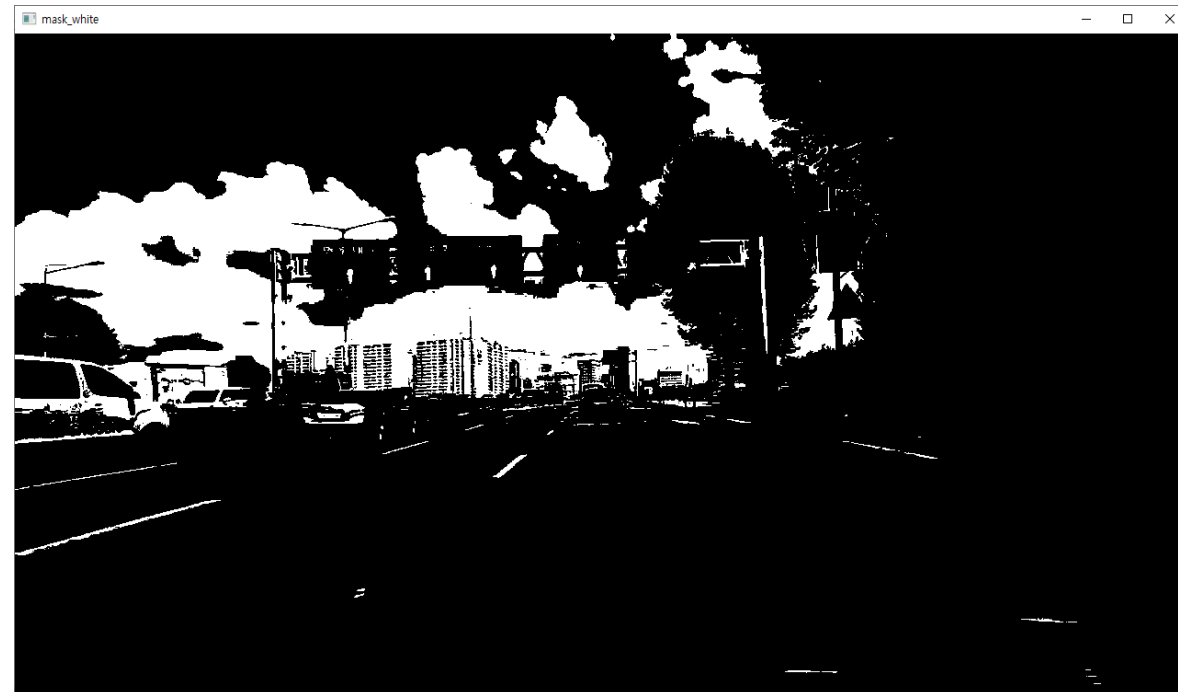
```
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]], dtype=uint8)
```



HSV 로 변환



yellow는 HSV 에서 적용



white는 BGR에서 적용

외곽선 구하기 : Canny엣지사용

```
frame_canny = cv2.Canny(mask_lane, 50,150)
```

```
frame_canny = cv2.bitwise_and(frame_canny, frame_canny, mask=img_mask)
```

외곽선을 기준으로 선을 추출

```
lines = cv2.HoughLinesP(frame_canny, rho, theta, threshold, minLineLength=min_line_length, maxLineGap=max_line_gap)
```

```
draw_lines(frame_bgr, lines)
```

```
cv2.polylines(frame_bgr, [pts], True, (255, 0,0), 3)
```

```
#####
```

```
video.write(frame_bgr)
```

```
cv2.imshow('frame_bgr', frame_bgr)
```

```
#####
```

```
key = cv2.waitKey(25)
```

```
if key == 27: # Esc키
```

```
    break;
```

```
capture.release()
```

```
video.release()
```

```
cv2.destroyAllWindows()
```

```
cv2.Canny(img, threshold1, threshold2)
```

cv2.bitwise_and(img1, img2, mask=None): 각 픽셀에 대해 AND 연산

cv2.HoughLinesP : 확률적 허프 변환에 의한 선 검출, 직선의 시작과 끝 정보를 제공

```
rho = 10
```

```
theta = 1 * np.pi / 180
```

```
threshold = 50
```

```
min_line_length = 10
```

```
max_line_gap = 30
```



mask_lane의 성능을 개선하기 위해서 모폴로지 적용

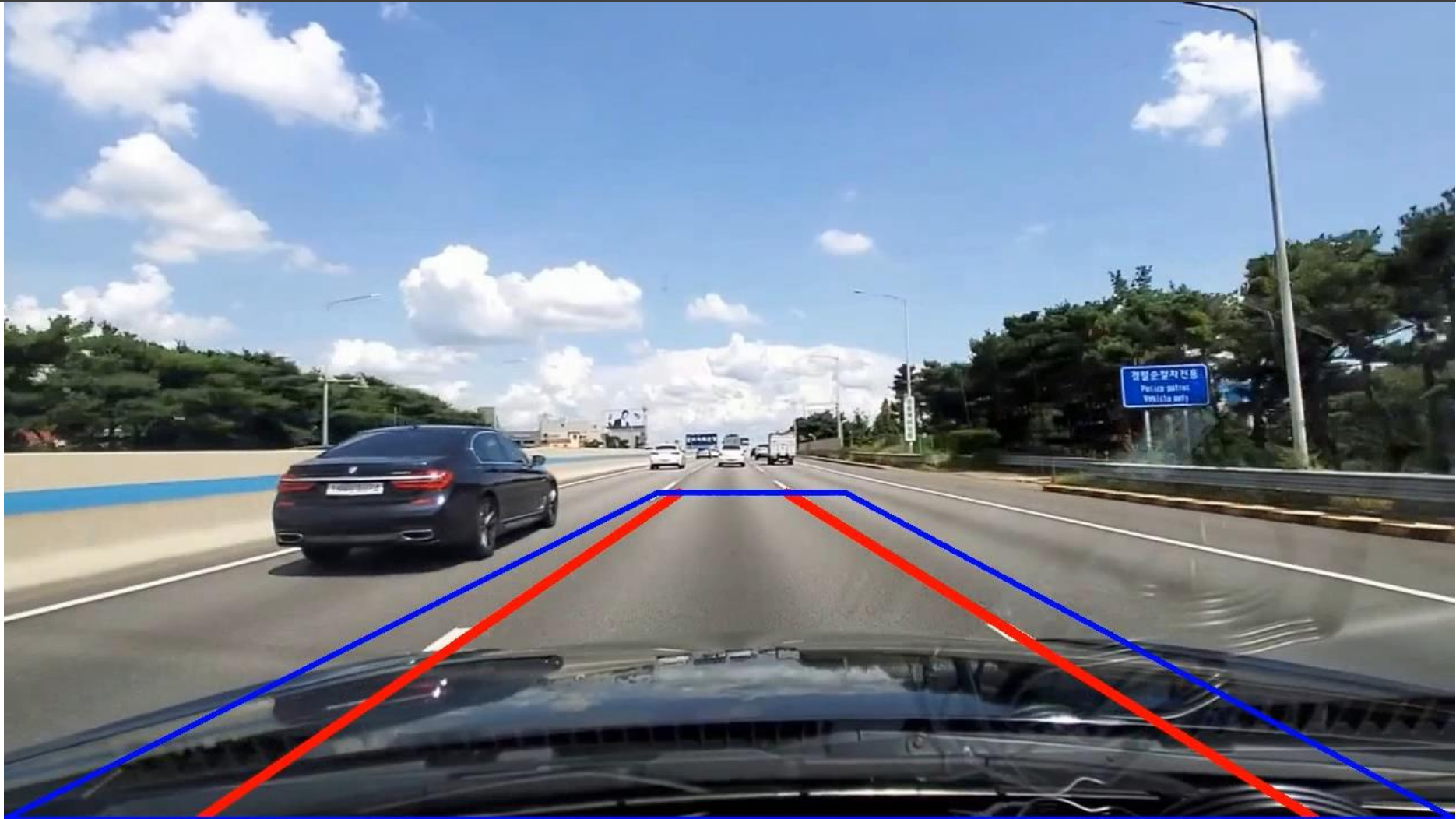


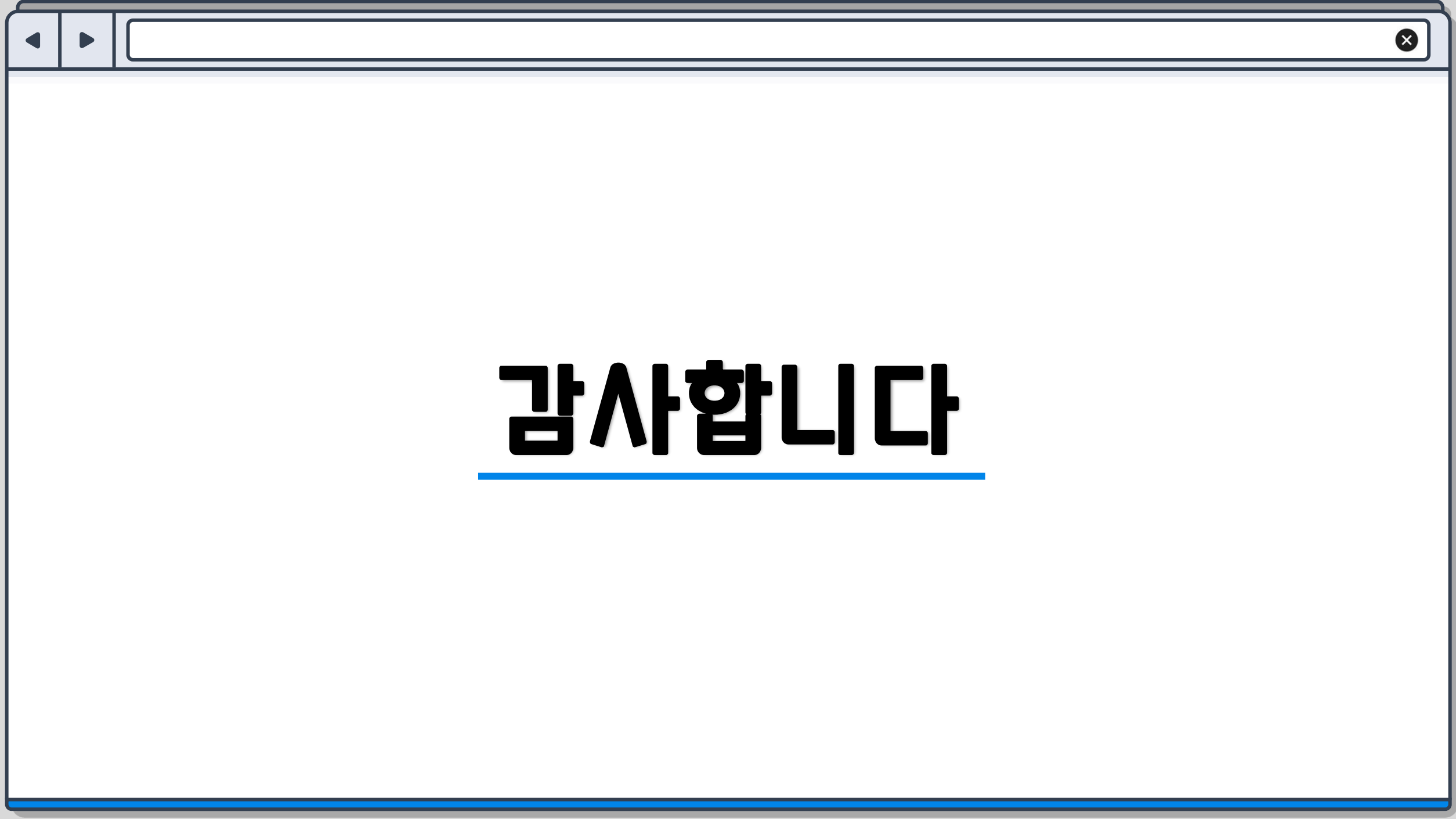
외곽선 구하기 : Canny 엣지 사용



OPENCV_BEFORE







감사합니다