차량지능기초 과제 1

20183404 정문규 자동차it융합학과

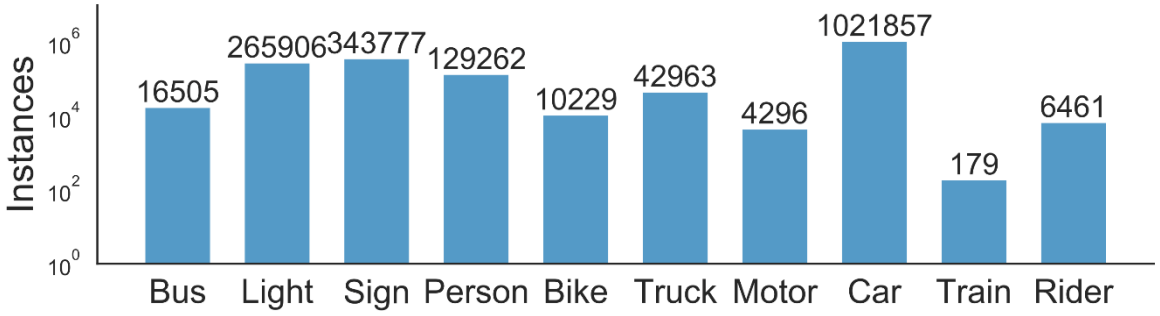
https://github.com/newsm5403/opencv_basic

자율주행 인지에 관련된 Data Set 조사

1. BDD100K :

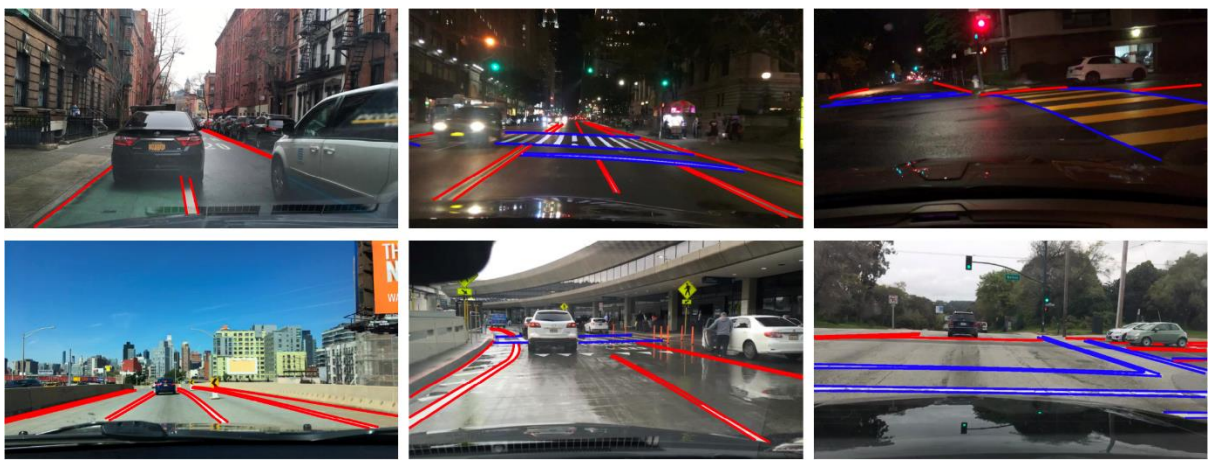
자동차 응용을 위한 컴퓨터 비전과 머신러닝의 최첨단 기술을 조사하는 버클리 딥드라이브 산업 컨소시엄이 주관하고 후원하는 프로젝트이다. 다양한 도시에서 촬영, 1억2천개의 이미지, 10만개의 시퀀스, 다양한 계절, 하루의 다양한 시간. 비디오에서 10초에 키프레임을 샘플링하고 그 키프레임에 대한 주석을 제공한다. 이미지 태그 지정, 도로 객체 경계 상자, 주행 가능 영역, 차선 표시 및 풀프레임 인스턴스 분할의 여러 레벨에서 레이블이 지정된다.

● 도로 객체 감지 :



● 차선표시 : 수직 차선 표시(빨간색)은 차선의 주행방향을 따라 표시된 차선이다.

평행 차선표시(파란색)은 차선에 있는 차량이 정지할 것을 나타낸다.



● 주행 가능 영역 : 우리가 주행할 수 있는 곳은 차선표시와 신호등에만 의존하지 않고 다른 물체들과의 복잡한 상호작용에 달려있다. 이를 위해 주행 가능 영역에 대한 분할 주석을 제공한다.

빨간색으로 표시된 곳은 현재 차량이 도로 우선권을 가지며 그곳에서 계속 주행할 수 있음을 의미한다.

반면 파란색으로 표시된 곳은 현재 차량이 주행은 가능하지만 다른 차량이 우선순위에 속할 가능성이 있다.

● 풀프레임 분할 : full-pixel-level 분할을 얻는 것은 비싸고 힘들지만 독자적인 라벨링 도구로 비용을 50% 절감했다. Data set 간의 도메인 이동을 더 쉽게 하기 위해 우리의 label 세트는 training 주석과 호환된다.

2. Mapillary Street-level Sequences Dataset (MSLS):

장소 인식을 위한 가장 크고 가장 다양한 데이터 셋이며, 많은 수의 짧은 시퀀스에 160만 개의 이미지를 포함하고 있다. 6개 대륙의 30개 도시에 걸쳐 있는 이 데이터 셋은 다양한 계절, 날씨 및 일광 조건, 다양한 카메라 유형과 뷰 포인트, 다양한 아키텍처 및 구조 설정(도로 공사 등), 장면에 존재하는 다양한 수준의 동적 물체(예: 이동 보행자 또는 자동차)를 포함한다.

동일한 위치에서 하루 중 다른 시간 또는 다른 년도에 찍은 사진들

각 이미지에는 원시GPS 좌표, 캡쳐 시간 및 나침반 각도, 주간/야간 특징 및 뷰 방향(전면,후면, 측면)과 같은 추가 연구에 관련된 메타 데이터와 속성이 함께 제공된다.

장소 인식을 위한 이전의 최첨단 방법을 사용하여 데이터 셋에 대한 광범위한 벤치마크를 실행했다. 지리적 분포, 계절적 및 시간적 변화, 특히 주간/야간 변화에서 데이터 셋의 다양성으로 인해 성능을 향상시킨다는 것을 보여준다.

각 열에 같은 장소이지만 하루의 다른 시간이나 다른 계절에 찍혀서 서로 매우 다른 모습을 하고 있다. 외관이 너무 다를 때 자동으로 장소를 탐지하는 것은 어렵지만 MSLS에서는 그러한 변화에 강력한 알고리즘을 훈련하는데 사용될 수 있다.

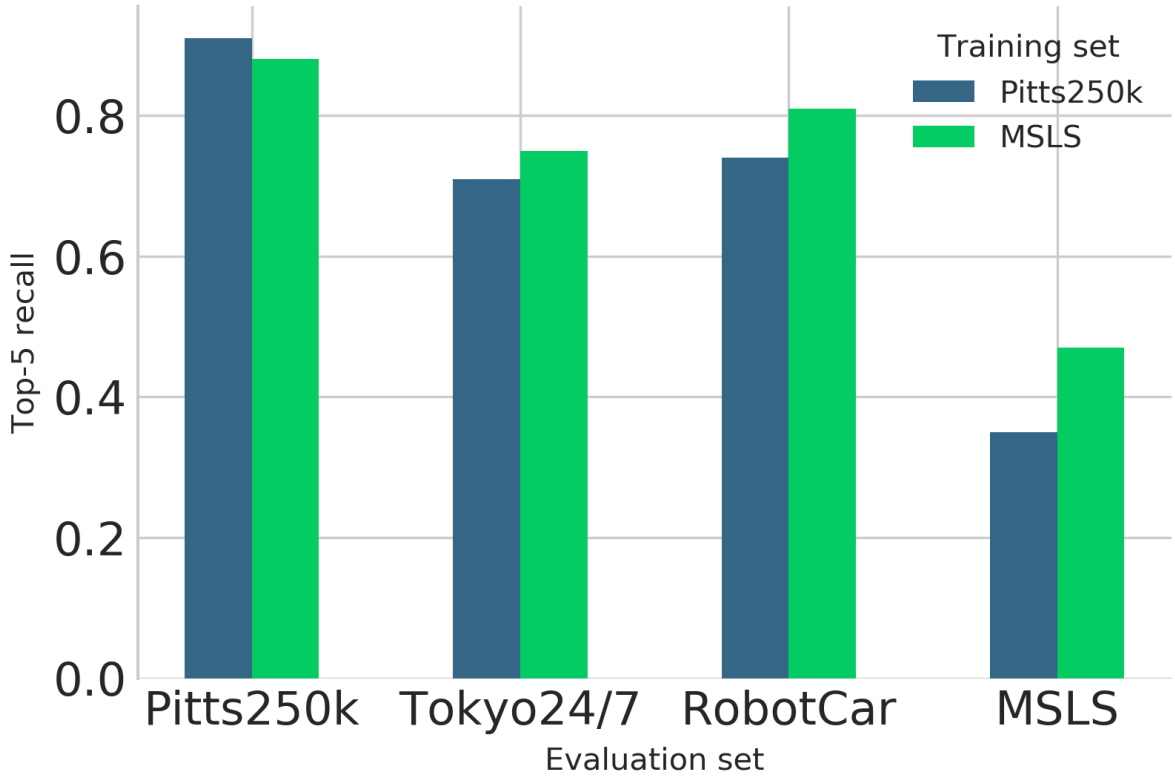
넓은 지리적 범위, 장면 특성의 다양성, 대용량 신경망을 훈련하기에 충분한 크기 때문에 MSLS은 시각적 장소 인식에서 최첨단 기술을 추진하고 전 세계의 실제 환경에서 응용하기 위한 최고의 데이터 세트이다.

● 장소 인식의 역할과 중요성

장소인식은 많은 대규모 컴퓨터 비전과 로봇 시스템에서 중요한 요소이다. 장소인식을 개발하고 벤치마킹하기 위해서는 다양한 시나리오를 대표하는 데이터세트가 필수적이다. 기존 데이터 셋은 장소인식 방법 개발에 크게 기여했지만, 특히 딥러닝을 기반으로 하는 방법을 개발할 때는 스케일이 제한적이고 장면 특성과 외관 변화가 다양해 여전히 한계가 있다. 이를 통해, MSLS 데이터셋은 연구실 밖에서 동작하는 딥러닝 기반 장소인식 알고리즘을 훈련하는데 사용될 수 있을 정도로 충분히 크고 다양하도록 설계했다.

● 벤치마킹 및 새로운 기준선 설정

이전 데이터 셋에는 일반적으로 단일 도시의 이미지가 포함 되어있다. 따라서 이런 데이터셋이 세계 다른 곳에서 장소 인식작업에서 잘 수행하기는 어렵다. 즉, 이전 데이터셋에서 훈련된 방법은 제대로 일반화되지 않는다.



Pitts250k(피츠버그의 이미지만 포함한 데이터 셋)은 피츠버그에서는 MSLS보다 약간 더 높은 결과를 갖지만 다른 세계 도시에서는 MSLS가 더 높은 결과를 보여준다.

3. Waymo Dataset:

이 자료는 Waymo 자율주행차가 수집한 고해상도 센서 데이터로 구성 되어있다. 데이터 셋은 밀집된 도시 중심에서 교외 풍경까지, 낮과 밤, 새벽과 황혼, 햇빛과 비 속에서 수집된 데이터까지 다양한 환경을 포함한다.

● 크기 : 1000개의 운전 세그먼트가 수록 되어있다. 각 세그먼트는 센서당 10Hz에서 20만 프레임에 해당하는 연속 주행 20초를 캡처했다.

● 다양한 주행 환경 : Phoenix, AZ, Kirkland, WA, Mountain View, CA, San Francisco 에서 광범위한 주행 조건(낮과 밤, 새벽과 황혼, 햇빛과 비)을 캡처한 밀집한 도시 및 교외 환경을 포함하고 있다.

● 고해상도 360도 뷰 : 각 세그먼트는 5개의 고해상도 Waymo LiDAR와 5개의 전면 및 측면 Camera의 센서 데이터를 포함하고 있다.

● 밀로 라벨링 : 데이터 셋에는 차량, 보행자, 자전거를 탄 사람 및 주의 깊게 라벨을 부착한 LiDAR 프레임과 이미지가 포함되어 있으며, 총 1200만 개의 3D 라벨과 120만 개의 2D 라벨을 캡처했다.

● Camera-LiDAR 동기화 : Waymo는 센서 배치와 고품질 시간 동기화를 포함하여 하드웨어와 소프트웨어를 포함한 전체 자율주행 시스템을 원활하게 연동되도록 설계했다.

Dataset에는 LiDAR 및 Camera 데이터의 고품질 시간 동기화가 포함되어 있다.

자율주행 인지에 관련된 Open Source 조사

1. 차량 번호판 영역 추출 및 인식하기

차량 사진을 받으면 차량 번호판을 인식해서 번호를 return 하는 코드이다.

Opencv를 활용해 코드가 구성되어 있다.

canny알고리즘과 같이 많이 쓰이는 가우시안 필터를 사용해 윤곽선을 추출했다.

원본 사진


```
#!/usr/bin/etc python

import cv2
import numpy as np
import pytesseract
from PIL import Image
pytesseract.pytesseract.tesseract_cmd = r'C:\Program
Files\Tesseract-OCR\tesseract.exe'

class Recognition:
    def ExtractNumber(self):
        global delta_x
        Number = 'unnamed.jpg'
        # 번호판의 글자에 윤곽선을 잡아주기 위해 OpenCV 라이브러리를
이용해 원본이미지에 전처리 과정을 해줘야 한다.
        img = cv2.imread(Number, cv2.IMREAD_COLOR)
```

```
copy_img = img.copy()
img2 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# 이미지 컬러를 Gray 로 바꿔준다


cv2.imwrite('gray.jpg', img2)
```

컬러를 gray 로 바꾼 사진


```
blur = cv2.GaussianBlur(img2, (3, 3), 0)
# Gray 이미지를 필터를 적용 시켜 윤곽선을 더 잘 잡을 수 있도록 한다.(엣지검출, 영상처리과정이 수월하도록)
cv2.imwrite('blur.jpg', blur)
```

블러 처리를 한 사진


```
canny = cv2.Canny(blur, 180, 200)
cv2.imwrite('canny.jpg', canny)
# 전처리 사진에서 엣지를 검출한다 (가우시안 필터 이미지에서 Canny Detection 을 사용해서 이미지를 추출한다.)
# 여기서 엣지란 흑백 영상에서 명암의 밝기 차이에 대한 변화율이다.
# cv2.Canny 와 cv2.GaussianBlur 의 Threshold1,2 의 값은 특정 사진에 맞춰 설정해준 값이다.
```

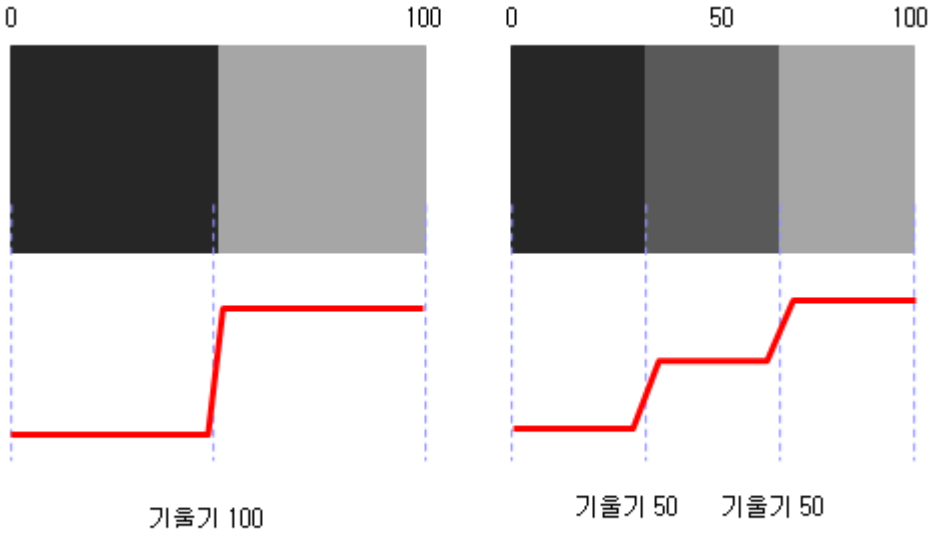
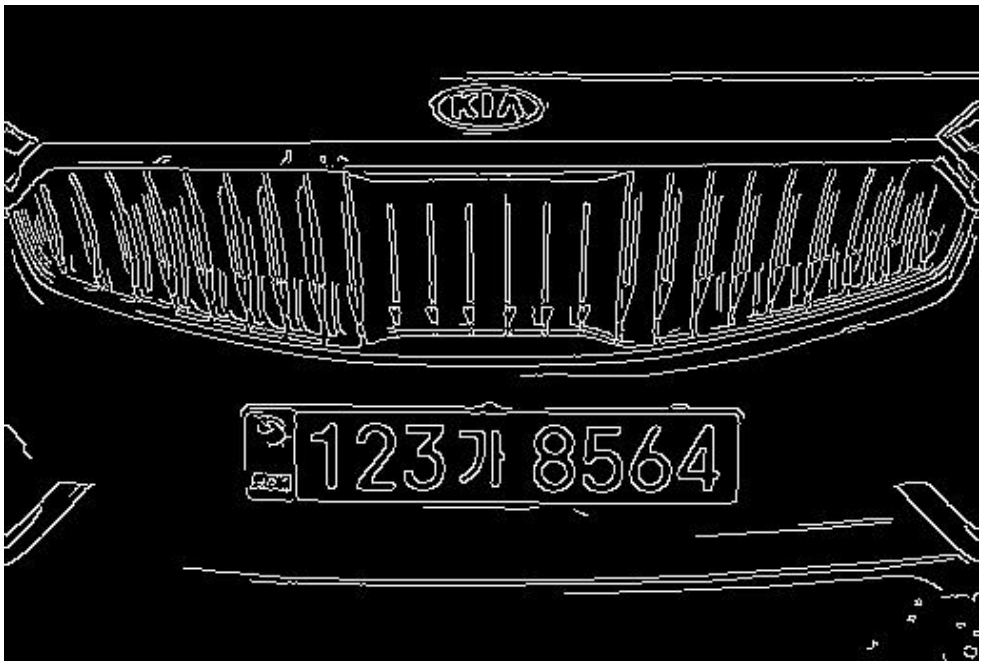


사진 출처:(http://iskim3068.tistory.com/49)

위 그림과 같이 Edge 란 흑백 이미지에 대한 미분 값이다.

이렇게 추출한 Edge 값으로 윤곽선을 얻는다.

윤곽선 처리를 한 사진




```
contours, hierarchy = cv2.findContours(canny, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
# 엣지를 추출한 이미지에서 cv.findContours 사용해 canny 이미지에 대해
Contours(윤곽선)을 찾는다.
# 여기서 Contours(윤곽선)은 같은 에너지를 가지는 점들을 연결한 선이다.
# OpenCV는 Contour(윤곽선)을 찾을 때 검은 바탕에 찾는 물체는 흰색으로
설정해야한다.

box1 = []
f_count = 0
select = 0
plate_width = 0

for i in range(len(contours)):  # findContours 함수로 찾은
contours(윤곽선)들의 bounding 처리해준다.
    cnt = contours[i]
    area = cv2.contourArea(cnt)  # 폐곡선 형태의 윤곽선으로 둘러싸인
면적
    x, y, w, h = cv2.boundingRect(cnt)  # 윤곽선 cnt 에 외접하는
직사각형의 좌상단 꼭지점 좌표, 가로, 세로 리턴
    rect_area = w * h  # 영역 크기
```

```
    aspect_ratio = float(w) / h  # ratio = width/height

    if (aspect_ratio >= 0.2) and (aspect_ratio <= 1.0) and
(rect_area >= 100) and (rect_area <= 900):
        #  전체 이미지에서 Contour 의 가로 세로 비율 값과 면적을 통해,
번호판 영역에 벗어난 걸로 추정되면 제외해준다.
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0),
1)  # 외접하는 직사각형 그리기

        box1.append(cv2.boundingRect(cnt))  # 외접하는 직사각형
리스트에 추가


for i in range(len(box1)):  # 버블 정렬
    for j in range(len(box1) - (i + 1)):
        if box1[j][0] > box1[j + 1][0]:
            temp = box1[j]
            box1[j] = box1[j + 1]
            box1[j + 1] = temp

# rectangles 사이의 길이를 재서 번호판을 찾는다. for 문을 마친 후
count 값이 가장 큰 직사각형이 번호판의 시작점이다.
for m in range(len(box1)):
    count = 0
    for n in range(m + 1, (len(box1) - 1)):
        delta_x = abs(box1[n + 1][0] - box1[m][0])
        if delta_x > 150:  # 일정 값 이상의 차이가 나면 종료
            break
        delta_y = abs(box1[n + 1][1] - box1[m][1])
        if delta_x == 0:  # 기울기를 구하려면 0 으로 나눌 순 없기 때문에
1 로 대체
            delta_x = 1
        if delta_y == 0:
            delta_y = 1
        gradient = float(delta_y) / float(delta_x)  # 직사각형
사이의 tan 값
        if gradient < 0.25:  # 일정 값 미만의 tan 값이면 count 값
증가
            count = count + 1
    # 번호판 사이즈를 잰다.
    if count > f_count:
        select = m
        f_count = count
```

```
        plate_width = delta_x
cv2.imwrite('snake.jpg', img)
```

직사각형 바운딩한 사진


```
number_plate = copy_img[box1[select][1] - 10:box1[select][3] +
box1[select][1] + 20,
              box1[select][0] - 50:200 + box1[select][0]]
# 번호판 사이즈 부분은 상수값으로 offset 줘서 추출한다.

resize_plate = cv2.resize(number_plate, None, fx=1.8, fy=1.8,
interpolation=cv2.INTER_CUBIC + cv2.INTER_LINEAR)
plate_gray = cv2.cvtColor(resize_plate, cv2.COLOR_BGR2GRAY)  #
번호판 영역 이미지를 Gray 바꾼다.
ret, th_plate = cv2.threshold(plate_gray, 150, 255,
cv2.THRESH_BINARY)  # cv2.threshold 흑백 값만 나오도록 이진화 처리를
한다.

cv2.imwrite('plate_th.jpg', th_plate)
```

번호판 추출 사진


```
kernel = np.ones((3, 3), np.uint8)
er_plate = cv2.erode(th_plate, kernel, iterations=1)  # erode
함수로 검은색 글자 강조
er_invplate = er_plate
cv2.imwrite('er_plate.jpg', er_invplate)
```

검정색 강조한 사진


```
result =
pytesseract.image_to_string(Image.open('er_plate.jpg'),
lang='kor')  # protester 번호판 인식
       return result.replace(" ", "")


recogtest = Recognition()  # 객체 생성
result = recogtest.ExtractNumber()  # 결과값 생성
print(result)  # 출력
```

2. 차선 인식

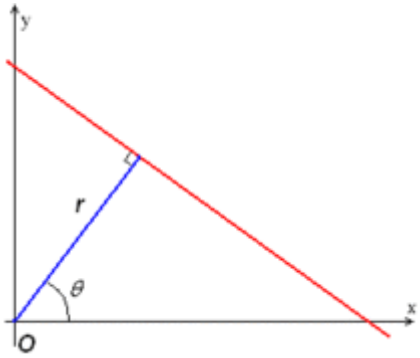
도로 사진을 입력하면 차선을 인식하는 코드이다.

이 코드도 마찬가지로 opencv 를 활용한 코드이며 차선을 인식하기 위해 허프 변환이라는 개념을 사용했다. 허프 변환의 개념은 코드 중간에 설명했다.

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


def grayscale(img):  # 흑백이미지로 변환
    return cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)


def canny(img, low_threshold, high_threshold):  # Canny 알고리즘 경계선을
검출한다.
    return cv2.Canny(img, low_threshold, high_threshold)


def gaussian_blur(img, kernel_size):  # 가우시안 필터 이미지의 잡음을 제거한다
    return cv2.GaussianBlur(img, (kernel_size, kernel_size), 0)


def region_of_interest(img, vertices, color3=(255, 255, 255), color1=255):
# ROI 셋팅 원하는 대상을 따로 분리해서 처리 하기 위해 설정한다.

    mask = np.zeros_like(img)  # mask = img 와 같은 크기의 빈 이미지


    if len(img.shape) > 2:  # Color 이미지(3 채널)라면 : 채널이 3 일 경우, 다색
이미지이다. 채널이 1 일 경우 단색 이미지이다.
        color = color3
    else:  # 흑백 이미지(1 채널)라면 :
        color = color1

    # vertices 에 정한 점들로 이뤄진 다각형부분(ROI 설정부분)을 color 로 채움
    cv2.fillPoly(mask, vertices, color)

    # 이미지와 color 로 채워진 ROI 를 합침
    ROI_image = cv2.bitwise_and(img, mask)
    return ROI_image


def draw_lines(img, lines, color=[255, 0, 0], 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 draw_fit_line(img, lines, color=[255, 0, 0], thickness=10):  # 대표선
그리기
```

```
    cv2.line(img, (lines[0], lines[1]), (lines[2], lines[3]), color,
thickness)


def hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap):  #
허프 변환
```

허프 변환이란???

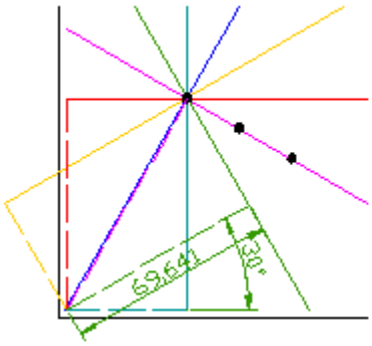


x-y 평면을 이미지라고 생각하자. 이미지 안에 있는 빨간 직선을 찾는 알고리즘을 허프 변환을 통해 찾을 수 있는데 세타와 rho 라는 두개의 파라미터를 가지고 구할 수 있다. 빨간 직선에 수직인 선분 r(파란색)을 식으로 나타내면,

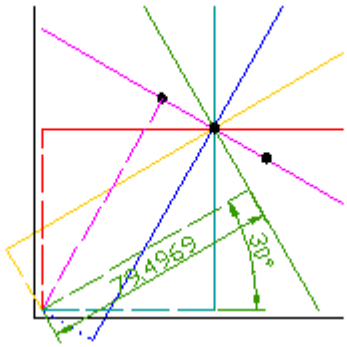
$r = x\cos\theta + y\sin\theta$ 이다.

실제 이미지에 허프 변환을 바로 적용하기에는 계산량이 많기 때문에 엣지 검출 알고리즘인 canny edge 알고리즘과 같이 쓰이며, 이렇게 검출된 엣지들에 있는 점들에 허프 변환을 적용해 직선을 찾을 수 있다.

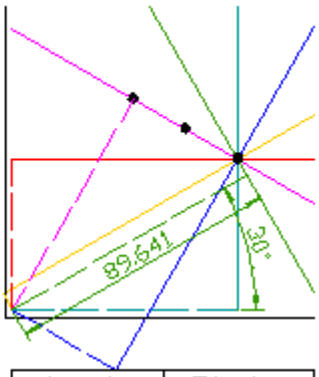
예를 들어 3 개의 점이 있다. 우리가 구하고 싶은 직선은 3 개의 점을 지나는 핑크색 직선이다. angle 은 세타이고 dist.는 rho 를 의미한다.



Angle	Dist.
0	40
30	69.6
60	81.2
90	70
120	40.6
150	0.4

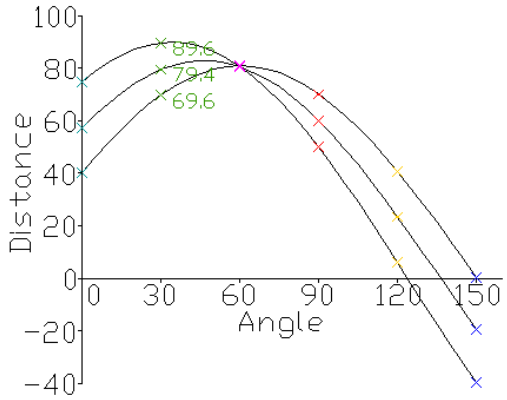


Angle	Dist.
0	57.1
30	79.5
60	80.5
90	60
120	23.4
150	-19.5



Angle	Dist.
0	74.6
30	89.6
60	80.6
90	50
120	6.0
150	-39.6

위에서 얻은 각도와 거리를 그래프로 표현하면 사인파의 그래프로 표현된다.



그래프를 보면 약 60 도에서 교점이 생기는데 이 곳이 핑크 직선이다. 만약 이미지에서 수많은 점들이 한 곳에서 만난다면 그 점은 세타와 rho 로 표현될 가능성이 높다.

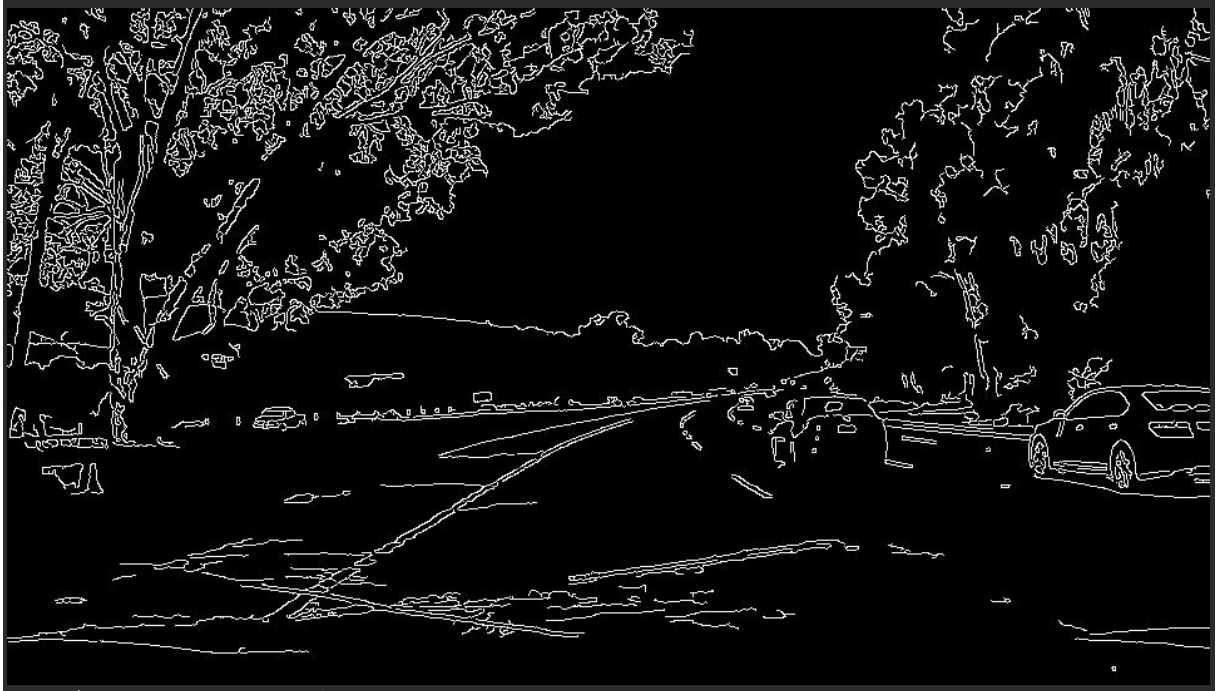
```
lines = cv2.HoughLinesP(img, rho, theta, threshold, np.array([]),
minLineLength=min_line_len,
                        maxLineGap=max_line_gap)
    # line_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
    # draw_lines(line_img, lines)

    return lines


def weighted_img(img, initial_img, α=1, β=1., λ=0.):  # 두 이미지 operlap 하기
    return cv2.addWeighted(initial_img, α, img, β, λ)


def get_fitline(img, f_lines):  # 대표선 구하기
    lines = np.squeeze(f_lines)
    lines = lines.reshape(lines.shape[0] * 2, 2)
    rows, cols = img.shape[:2]
    output = cv2.fitLine(lines, cv2.DIST_L2, 0, 0.01, 0.01)
    vx, vy, x, y = output[0], output[1], output[2], output[3]
    x1, y1 = int(((img.shape[0] - 1) - y) / vy * vx + x), img.shape[0] - 1
    x2, y2 = int(((img.shape[0] / 2 + 100) - y) / vy * vx + x),
int(img.shape[0] / 2 + 100)

    result = [x1, y1, x2, y2]
    return result
```

```
image = cv2.imread('slope_test.jpg')  # 이미지 읽기
```

```
height, width = image.shape[:2]  # 이미지 높이, 너비 저장
gray_img = grayscale(image)  # 흑백이미지로 변환
```

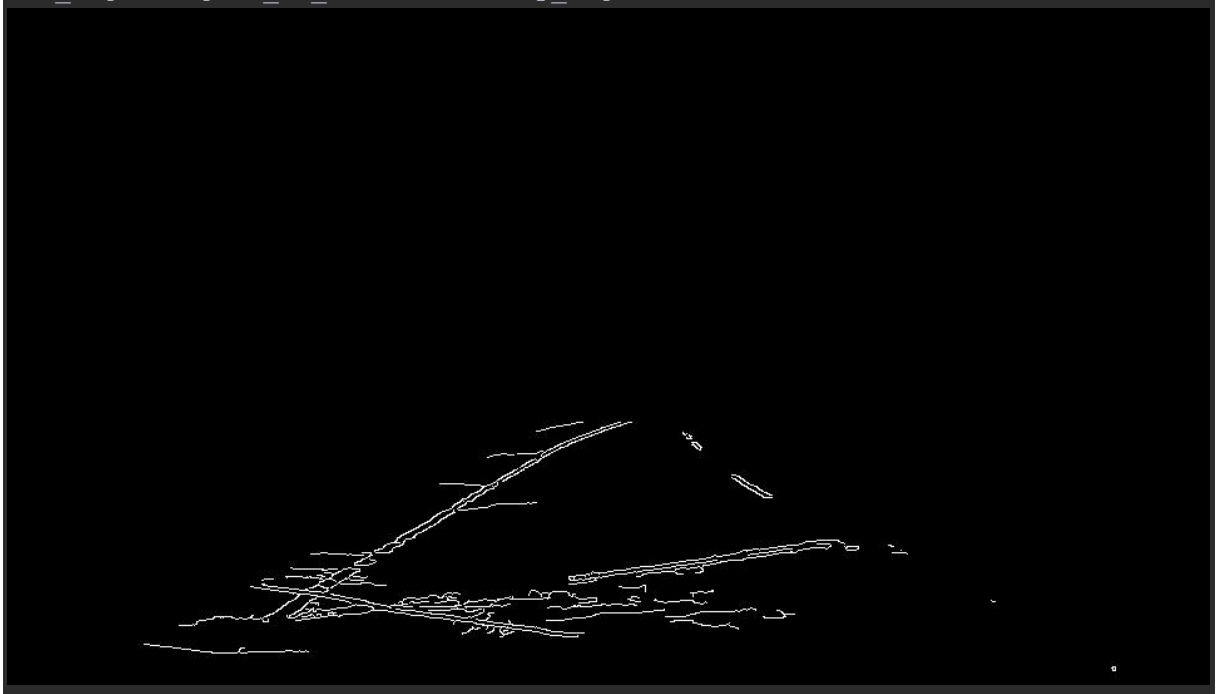
```
blur_img = gaussian_blur(gray_img, 3)  # Blur 효과
```

```
canny_img = canny(blur_img, 70, 210)  # Canny edge 알고리즘을 통해 윤곽선 이미지 리턴
```



```
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)  # ROI 설정
```



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

# 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]
# 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]
# 왼쪽, 오른쪽 각각 대표선 구하기
left_fit_line = get_fitline(image, L_lines)
right_fit_line = get_fitline(image, R_lines)
# 대표선 그리기
draw_fit_line(temp, left_fit_line)
draw_fit_line(temp, right_fit_line)

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

```
cv2.imshow('result', result)  # 결과 이미지 출력
```

```
cv2.waitKey(0)
```

Open source 중 하나 실행해서 결과 확인(2번)

구현환경

Python 3.5 와 Opencv 3.1 기준으로 코드가 작성됐다.

1. opencv란 오픈 소스 컴퓨터 비전 라이브러리 중 하나로 크로스플랫폼과 실시간 이미지 프로세싱에 중점을 둔 라이브러리이다. Windows, Linux, macOS, IOS, Android 등 다양한 플랫폼을 지원한다.

본래 C언어만 지원했지만 현재는 c++과 python을 공식적으로 지원한다.

픽셀 단위의 접근이 빈번하게 이루어진다면 당연히 C++을 써야겠지만, 단순한 매트릭스 연산이라면 python의 numpy와 cv2의 궁합을 이용하면 상당히 편리하다.

Opencv가 제공하는 주요 알고리즘은 이진화(binarization), 노이즈 제거(주로 가우시안 필터를 사용한다), 외곽선 검출(여기선 canny 알고리즘을 사용했다), 기계학습, ROI(Region Of Interest) 설정, 이미지 변환(image wraping) 등이 있다.

사용 방법

운영체제는 Windows이다.

OpenCV는 pip을 통해 설치할 수 있다.

명령 프롬프트나 터미널에서

```
python -m pip install opencv-python
```

명령어로 설치할 수 있다.

설치가 됐으면 아래코드처럼 import 해주면 사용 가능하다.

```
import cv2
```

2. Numpy란 고성능 수치계산을 위해 만들어진 파이썬 패키지이다.

Numpy는 외부 라이브러리이기 때문에 install을 해야한다.

다운로드 사이트

https://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy

자신의 버전에 맞게 확인하여 다운받는다.

파이썬 버전이 3.6이면 cp36

윈도우 64비트면 adm64이다.

설치 명령어 : python -m pip install {다운로드 파일 경로}

```
>python -m pip install D:₩numpy-1.13.3+mkl-cp36-cp36m-win64.whl
...
...
Successfully installed numpy-1.13.3+mkl
```

np라는 이름으로 import 해준다.

```
import numpy as np
```

코드(2번) 실행결과

이 코드는 9개의 함수와 구현부로 이루어져 있다.

```
def grayscale(img):  # 흑백이미지로 변환
    return cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
```

cv2.cvtColor(img, cv2.COLOR_RGB2GRAY) 는 입력받은 이미지를 gray로 바꿔주는 함수이다.

```
def canny(img, low_threshold, high_threshold):  # Canny 알고리즘 경계선을
검출한다.
    return cv2.Canny(img, low_threshold, high_threshold)
```

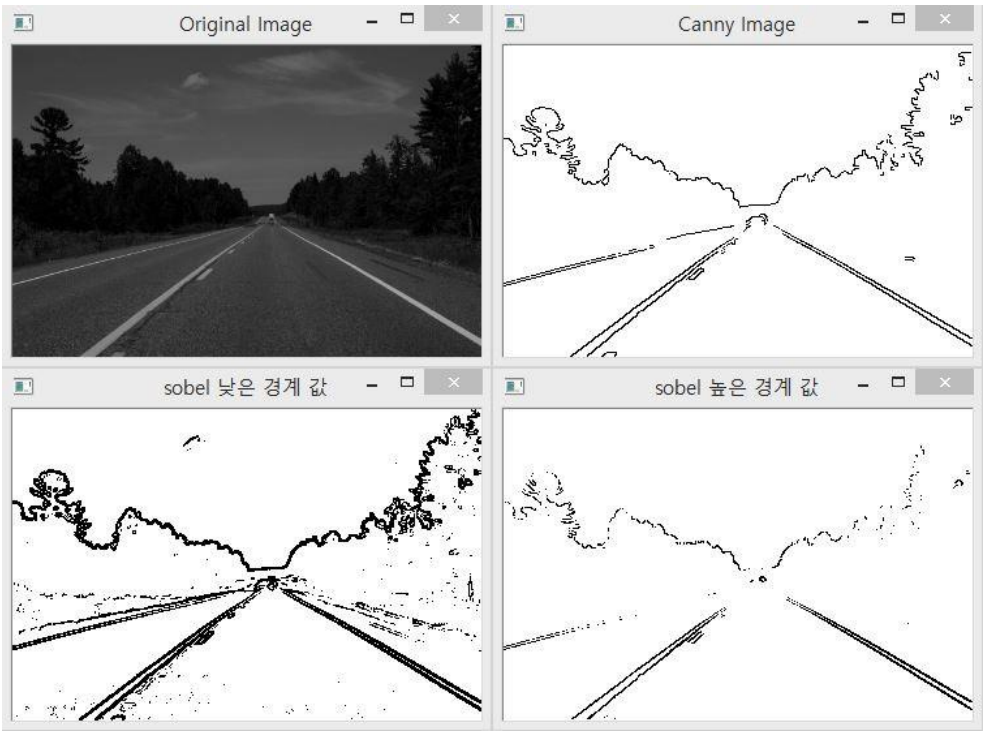
cv2.Canny(img, low_threshold, hight_threshold) 는 경계선을 찾는 방식 중 가장 좋은 방식으로 알려져 있다.

단계 1 : 가우시안 필터링을 하여 영상을 부드럽게 한다.

단계 2 : Sobel 연산자를 이용해 기울기(gradient) 벡터의 크기(magnitude)를 계산

단계 3 : 가느다란 Edge를 얻기 위해 3*3 창을 사용하여 gradient 벡터 방향에서 gradient 크기가 최대값인 화소만 남기고 나머지는 0으로 억제

단계 4 : 연결된 Edge를 얻기 위해 두 개의 임계값을 사용. 높은 값의 임계값을 사용하여 gradient 방향에서 낮은 값의 임계값이 나올 때까지 추적하며 Edge를 연결하는 히스테리시스 임계값(hysteresis thresholding) 방식을 사용



Sobel 낮은 경계 값 이미지는 Edge를 검출했지만 진하다.

반대로 Sobel 높은 경계 값 이미지는 Edge를 정확히 찾질 못했다.

반면 Canny 이미지는 진하지도 않고 흐리지도 않다. (참고로 canny는 흑색 배경에 흰색 Edge 이미지지만 비교를 위해 반전을 시켜줬다.)

의미있는 Edge에 속해 보이는 모든 Edge 화소들을 포함하기 위해서는 낮은 경계(low_threshold)를 선택해야한다. 하지만 필요한 Edge보다 더 많이 검출했다.

반대로 높은 경계 값(high_threshold)의 경우 흐릿하지만 중요한 외곽선에는 확실히 Edge를 표시해준다.

Canny 알고리즘은 외곽선의 optimal map을 만들기 위해 두 Edge를 조합한다.

여기서 hysteresis thresholding 방식을 사용한다.

High threshold와 low threshold의 2개의 값이 존재하며 high threshold 이상의 edginess를 갖는 픽셀들은 무조건 edge 픽셀로 분류한다.

Low 와 high 사이에 있으면서 이미 edge로 분류된 픽셀과 연결되어 있으면 edge로 분류한다.

나머지(low threshold 이하 또는 high threshold 이하 면서 edge 픽셀과 연결되어 있지 않은 경우)는 모두 non edge로 분류한다.

즉 hysteresis thresholding 은 확실한 판단이 될 경우 그대로 이진화를 하되, 불확실한 경우엔 주변 상황을 보고 결정을 하는 것이다.

```
cv2.Canny(img, low_threshold, high_threshold)
```

그렇기 때문에 함수 매개변수에 low_threshold와 high_threshold가 들어있다.

```
def gaussian_blur(img, kernel_size):  # 가우시안 필터 이미지의 잡음을 제거한다
    return cv2.GaussianBlur(img, (kernel_size, kernel_size), 0)
```

canny 알고리즘을 사용하기 위해 가우시안 필터로 전처리를 해준다.

```
def region_of_interest(img, vertices, color3=(255, 255, 255), color1=255):
# ROI 셋팅 원하는 대상을 따로 분리해서 처리 하기 위해 설정한다.

    mask = np.zeros_like(img)  # mask = img와 같은 크기의 빈 이미지


    if len(img.shape) > 2:  # Color 이미지(3채널)라면 : 채널이 3일 경우, 다색
이미지이다. 채널이 1일 경우 단색 이미지이다.
        color = color3
    else:  # 흑백 이미지(1채널)라면 :
        color = color1

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

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

관심 영역을 설정하기 위한 함수이다.

관심영역을 설정하면 정확도를 높이고, 연산속도를 증가시키기 때문에 필수적이다.

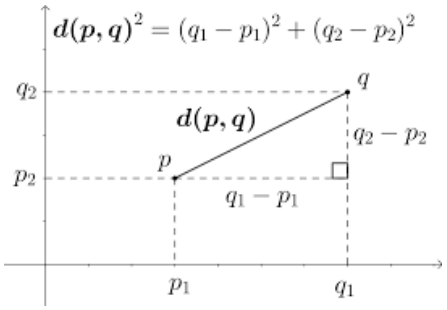
```
mask = np.zeros_like(img)  # mask = img와 같은 크기의 빈 이미지
```

빈 이미지를 사용하는 이유는 ROI를 사용할 때 원본 이미지를 참조할 수 있기 때문이다.

```
def get_fitline(img, f_lines):  # 대표선 구하기
    lines = np.squeeze(f_lines)
    lines = lines.reshape(lines.shape[0] * 2, 2)
    rows, cols = img.shape[:2]
    output = cv2.fitLine(lines, cv2.DIST_L2, 0, 0.01, 0.01)
    vx, vy, x, y = output[0], output[1], output[2], output[3]
    x1, y1 = int(((img.shape[0] - 1) - y) / vy * vx + x), img.shape[0] - 1
    x2, y2 = int(((img.shape[0] / 2 + 100) - y) / vy * vx + x),
int(img.shape[0] / 2 + 100)

    result = [x1, y1, x2, y2]
    return result
```

```
output = cv2.fitLine(lines, cv2.DIST_L2, 0, 0.01, 0.01)
```

cv2.fitLine()에서 매개변수로 받는 cv2.DIST_L2는 유클리드 거리이다.

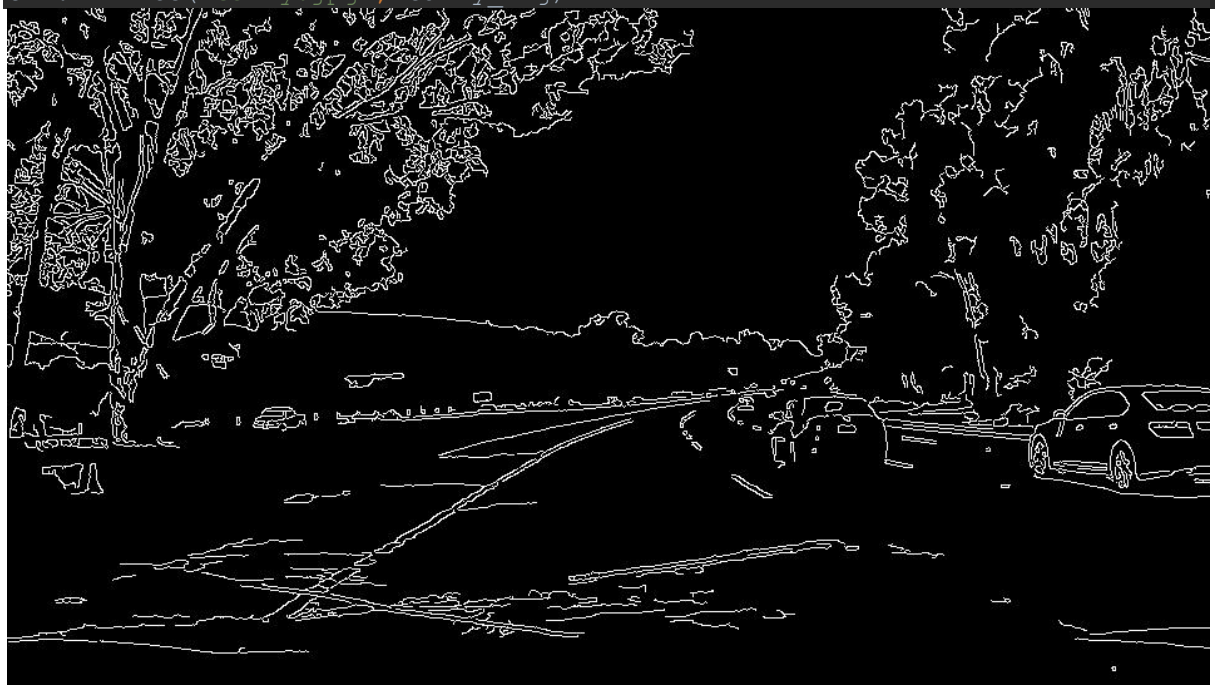


유클리드 거리란 두 점 사이의 거리이다.

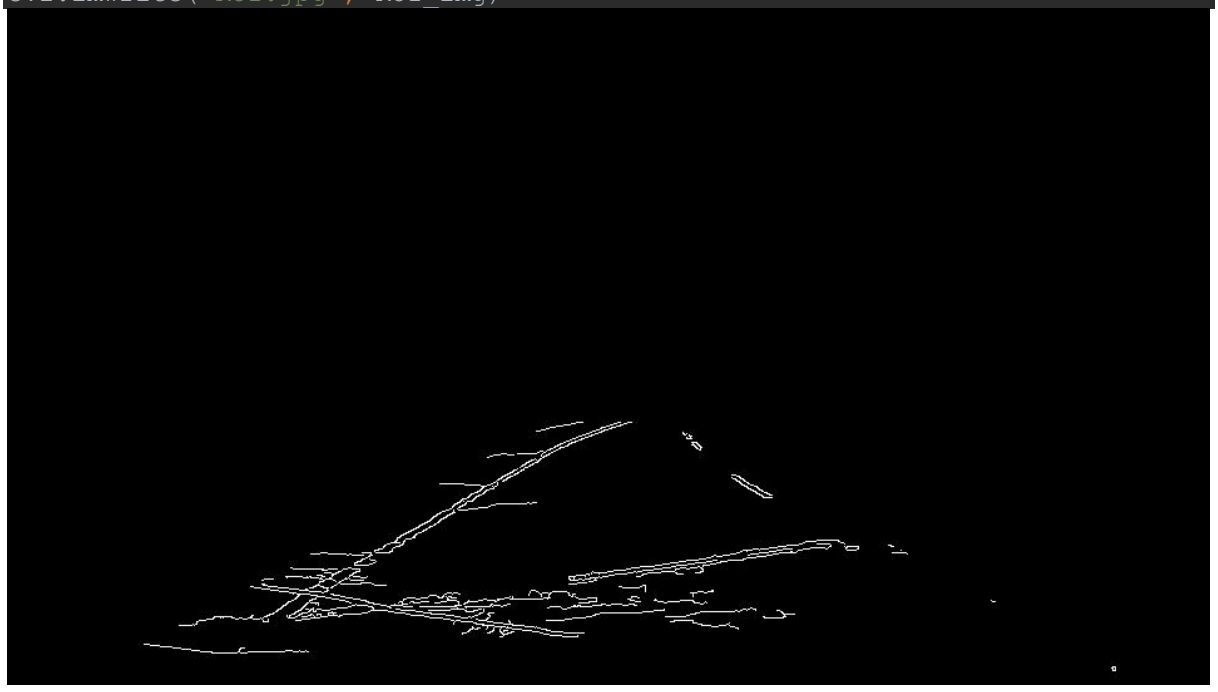
```
image = cv2.imread('slope_test.jpg')  # 이미지 읽기
height, width = image.shape[:2]  # 이미지 높이, 너비 저장
gray_img = grayscale(image)  # 흑백이미지로 변환
cv2.imshow('gray', gray_img)
cv2.imwrite('gray.jpg', gray_img)
```

```
blur_img = gaussian_blur(gray_img, 3)  # Blur 효과
cv2.imshow('blur', blur_img)
cv2.imwrite('blur.jpg', blur_img)
```

```
canny_img = canny(blur_img, 70, 210)  # Canny edge 알고리즘을 통해 윤곽선 이미지 리턴
cv2.imshow('canny', canny_img)
cv2.imwrite('canny.jpg', canny_img)
```



```
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)  # ROI 설정
cv2.imshow('ROI', ROI_img)
cv2.imwrite('ROI.jpg', ROI_img)
```



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

# 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]
# 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]
# 왼쪽, 오른쪽 각각 대표선 구하기
left_fit_line = get_fitline(image, L_lines)
right_fit_line = get_fitline(image, R_lines)
# 대표선 그리기
draw_fit_line(temp, left_fit_line)
draw_fit_line(temp, right_fit_line)

result = weighted_img(temp, image)  # 원본 이미지에 검출된 선 overlap
cv2.imshow('result', result)  # 결과 이미지 출력
cv2.imwrite('result.jpg', result)
cv2.waitKey(0)
```