차량지능기초 과제

20203086 소프트웨어학부 송수인

I. 자율주행 인지 관련 Open Dataset

1. Waymo Open Dataset for Perception

1-1. Dataset 설명

구글의 자율주행 관련 자회사인 Waymo에서 20??년 8월 21일 공개한 자율주행용 고품질 다양한 센서 Dataset으로 밀집된 도시 중심, 교외 풍경, 낮과 밤, 날씨 등 다양한 환경에서 Waymo의 자율주행차가 수집한 고해상도 센서 데이터로 구성되어 있다.

1-2. 크기와 범위(Size and Coverage)

1000개의 Driving Segment가 포함되어 있으며 각 Segment는 센서당 10Hz에서 20만 프레임에 해당하는 연속 주행 20초를 캡처한 영상이다. (그림 1)

그림 1. Waymo Dataset Segments

1-3. 구성

해당 Dateset은 Phoenix, AZ, Kirkland, WA, Mountain View, CA 와 San Francisco, CA에서 광범위한 주행 조건(낮과 밤, 시간대, 날씨) 상의 도로를 캡처한 밀집한 도시 및

교외 환경을 포함하고 있다.

각 Segment는 5개의 고해상도 Waymo LiDAR와 5개의 전면 및 측면 Camera의 센서 데이터를 포함하고 있으며 차량, 보행자, 자전거 탄 사람 및 민감하게 라벨링된 LiDAR 프레임과 이미지가 포함되어 있고 총 1200만 개의 3D 라벨과 120만 개의 2D 라벨이 캡처되어 있다. (그림 2)

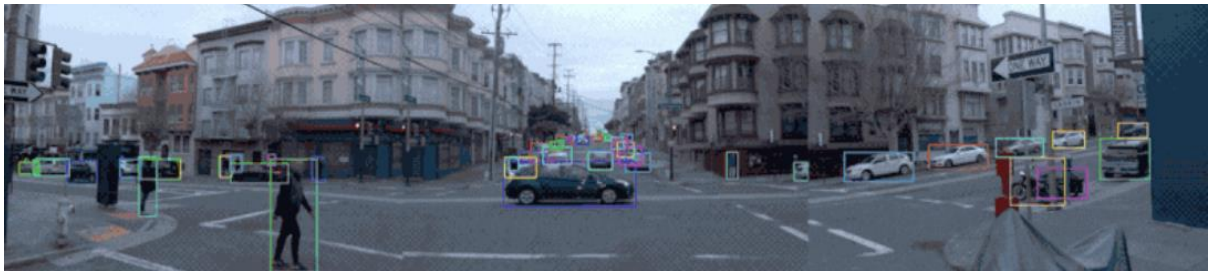


그림 2. Waymo Dense Labeling

2. 도로주행영상 AI 데이터

2-1. Dataset 설명

㈜티큐에스코리아가 주관하고 복수의 기업이 참여한 AIHub에서 공개한 자율주행 모사차량을 활용하여 사전에 계획된 유즈케이스(UseCase) 및 시나리오에 따라서 주행영상을 취득하고, 정제, 가공, 검수 과정을 거쳐서 자율주행 인지 AI학습 및 다양한 분야에서 활용 가능한 학습용 데이터셋이다. (그림 3)

(그림 3)

2-2. 크기와 범위(Size and Coverage)

실도로 주행 DB로 분류되는 Use-case(DGPS, LiDAR, IVN)가 70개 이상 포함되어 있고 차량번호판 및 사람 얼굴 마스킹이 28만 frame 이상, 학습용 데이터인 바운딩 박스, 스플라인, 폴리곤이 20만 frame 이상, 자유주행공간(폴리곤)이 5천 frame 이상 포함되어 있다.

2-3. 구성

XML 형식의 자동차, 보행자, 차선, 신호등, 표지판, 노면표시, 노면 화살표 도면이 포함되어 있고 실도로 주행 DB 포함 메타데이터가 있다.

3. BDD100K

3-1. Dataset 설명

BDD100K는 Berkeley Deep Drive의 약자로, 날씨와 조도에 따른 동적인 주행환경 구현, GPS 정보, IMU 데이터 및 스탬프가 포함되어 있다. 녹화된 비

디오는 우천, 흐린 날씨, 맑은 날씨, 안개와 같은 다양한 날씨 조건이 기록되어 있다. Dataset는 낮과 밤이 적절한 비율로 기록되어 있다.

3-2. 크기와 범위(Size and Coverage)

BDD100K는 40초의 비디오 sequence, 720픽셀 해상도, 초당 30 프레임 고화질로 취득된 100,000개 비디오 sequence로 구성된다. 또한 해당 Dataset은 버스, 신호등, 교통 표지, 사람, 자전거, 트럭, 모터, 자동차, 기차 및 라이더를 위해 100,000개 이미지에 주석이 달린 2D Bounding Box가 포함되어 있다. 또한 차량용 컴퓨터 비전 및 기계 학습을 연구하기 위해, 현재 백만 대의 자동차, 300,000rork 넘는 도로 표지판, 120,000명의 보행자 등으로 구성된 Berkeley Deep Drive (BDD) Industry Consortium의 지원을 받고 있다.

II. 자율주행 인지 관련 Open Source 분석

II – 1. Line recognition using OpenCV

Line 1 ~ 4. 모듈 Import

```
# -*- coding: cp949 -*-
# -*- coding: utf-8 -*-
import cv2 # import OpenCV
import numpy as np # import Numpy
```

Line 1 ~ 2는 한글 주석을 오류나 깨짐 현상없이 달기 위한 코드이다.

Line 3 ~ 4는 OpenCV와 Numpy 라이브러리를 사용하기 위해 Import하는 코드이다.

Line 7 ~ 8. Convert RGB scale to gray scale

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

본 함수는 Canny edge detection algorithm을 적용하기 위하여 RGB iamge를 cv2.cvtColor 메소드와 cv2.COLOR_RGB2GRAY 옵션을 사용하여 Gray image로 변환하는 기능을 한다.

line 11 ~ 12. Canny edge detection algorithm

```
def canny(img, low_threshold, high_threshold):
     return cv2.Canny(img, low_threshold,
high_threshold)
```

위에서 변환된 Gray image에 Canny algorithm을 적용한다.

- **Canny edge detection algorithm이란?**

  영상 내의 edge를 검출하는 알고리즘 중 상당히 인기있는 알고리즘으로, 다음 단계를 따라 edge를 검출한다.

1. 노이즈 제거

   영상에 노이즈가 있으면 edge 검출의 정확도가 감소하므로 5X5 Gaussian filter를 적용하여 이미지의 노이즈를 줄인다.

2. Gradient 값이 높은 부분 찾기

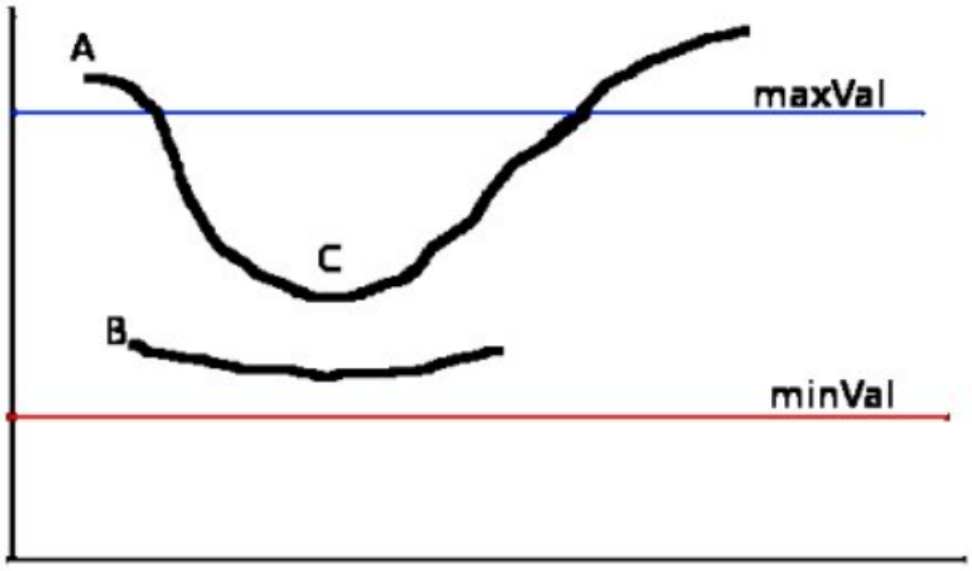
   노이즈가 제거된 영상을 Sobel 커널을 수평방향, 수직방향으로 적용하여 각 방향의 gradient를 획득한다.

3. 최대값이 아닌 픽셀의 값을 0으로 만들기

   2단계를 거친 후, edge에 기여하지 않은 픽셀을 제거하기 위해 영상 전체를 스캔한다. 영상을 스캔하는 동안, gradient 방향으로 스캔 구역에서 gradient의 최대값을 가진 픽셀을 찾는다.

4. Hyteresis Thresholding

   3단계를 거친 후보들 중 실제 edge가 무엇인지 판단하는 단계이다. 먼저 임계값 minVal과 maxVal을 임의로 설정한 뒤, maxVal보다 높은 부분은 확실하게 edge로 판단하고 minVal보다 낮은 부분은 확실하게 edge가 아니라고 판단한다. minVal과 maxVal 사이에 있는 값들은 이 픽셀들의 연결구조를 보고 edge인지 edge가 아닌지 판단한다.



line 15 ~ 16. Gaussian filter

```
def gaussian_blur(img, kernel_size):
    return cv2.GaussianBlur(img, (kernel_size, kernel_size), 0)
```

임의로 지정한 Kernel size대로 Gaussian filter를 적용한다.

- **Gaussian filter란?**

  Gaussian distribution(가우시안 분포, 정규분포)을 영상처리에 적용한 것으로 정규분포, 확률분포에 의해 생성된 잡음을 제거하기 위한 필터이다.

line 19 ~ 36. Set region of interest(ROI)

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

    mask = np.zeros_like(img)

    if len(img.shape) > 2:
        color = color3
    else:
        color = color1
```

```
    cv2.fillPoly(mask, vertices, color)

    ROI_image = cv2.bitwise_and(img, mask)
        return ROI_image
```

- **Region of interest(ROI)란?**

  Region of interest(ROI)는 영상처리 내에서 영상처리 연산량을 줄이기 위해 목표 객체가 있을 가능성이 높은 부분을 임의로 지정하여 주는 것이다.

Line 22는 ROI 설정을 위하여 읽은 image와 크기가 같은 빈화면인 mask를 생성해준다.

Line 24 ~ 27은 읽은 image가 RGB scale인지 Gray scale인지를 판단하여 mask에서 ROI가 될 부분을 표시할 색(color 변수)을 흰색으로 설정한다.

Line 29는 cv2.fillPoly() 메소드를 이용하여 mask 위에 parameter로 설정한 ROI 다각형의 점 정보인 vertices에 따라 흰색으로 다각형을 그린다. (mask)

Line 31 ~ 32는 cv2.bitwise_and() 메소드를 이용하여 읽은 image와 mask를 더하여 return 한다. (ROI)

line 39 ~ 42. Draw line

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

Canny edge detection algorithm을 이용하여 찾은 edge(차선)를 표시하기 위하여 파란색으로 선을 그린다.

line 45 ~ 51. Hough transformation

```
def hough_lines(img, rho, theta, threshold,
min_line_len, max_line_gap):
    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 line_img
```

차선 찾기의 정확도를 높이기 위하여 Hough Transformation을 적용한다.

line 54 ~ 55. Synthesize image and Hough transformation image

```
def weighted_img(img, initial_img, α=1, β=1.,
λ=0.):
    return cv2.addWeighted(initial_img, α, img, β,
λ)
```

읽은 image와 Hough transformation이 적용된 이미지를 cv2.addweighted() 메소드를 이용하여 합성한다.

III. Line recognition 코드 main() 설명 및 결과

III – 1. 실행환경

Window 10

Pycharm

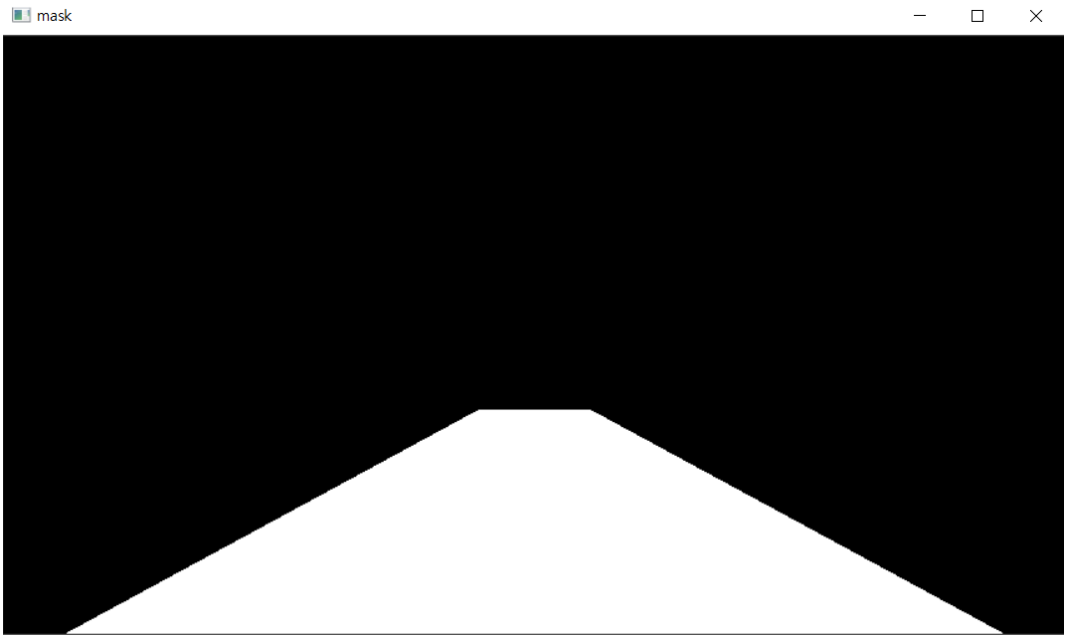
OpenCV ver. 4.5.1

Numpy ver. 1.19.3

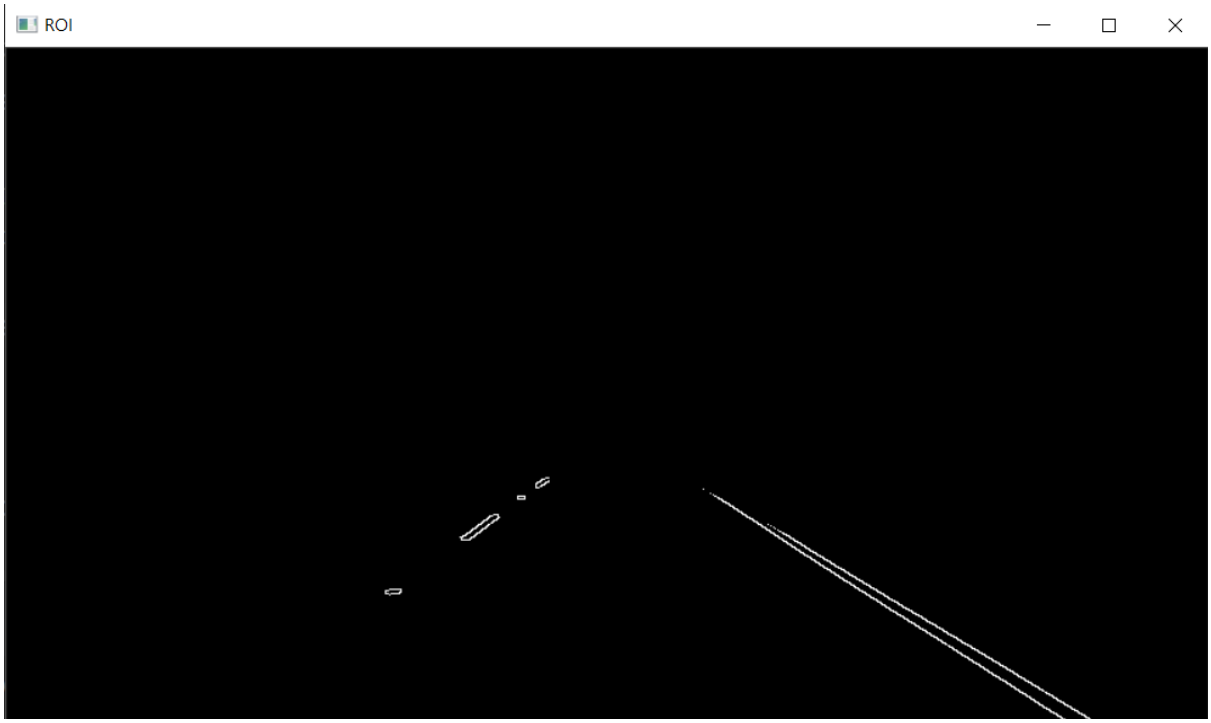
III – 2. 실행 결과

```
if __name__ == "__main__":
    cap = cv2.VideoCapture('solidWhiteRight.mp4')

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

        height, width = image.shape[:2]

        gray_img = grayscale(image)

        blur_img = gaussian_blur(gray_img, 3)

        canny_img = canny(blur_img, 70, 210)

        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)

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

        result = weighted_img(hough_img, image)
        cv2.imshow('result', result)

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

cap.release()
cv2.destroyAllWindows()
```

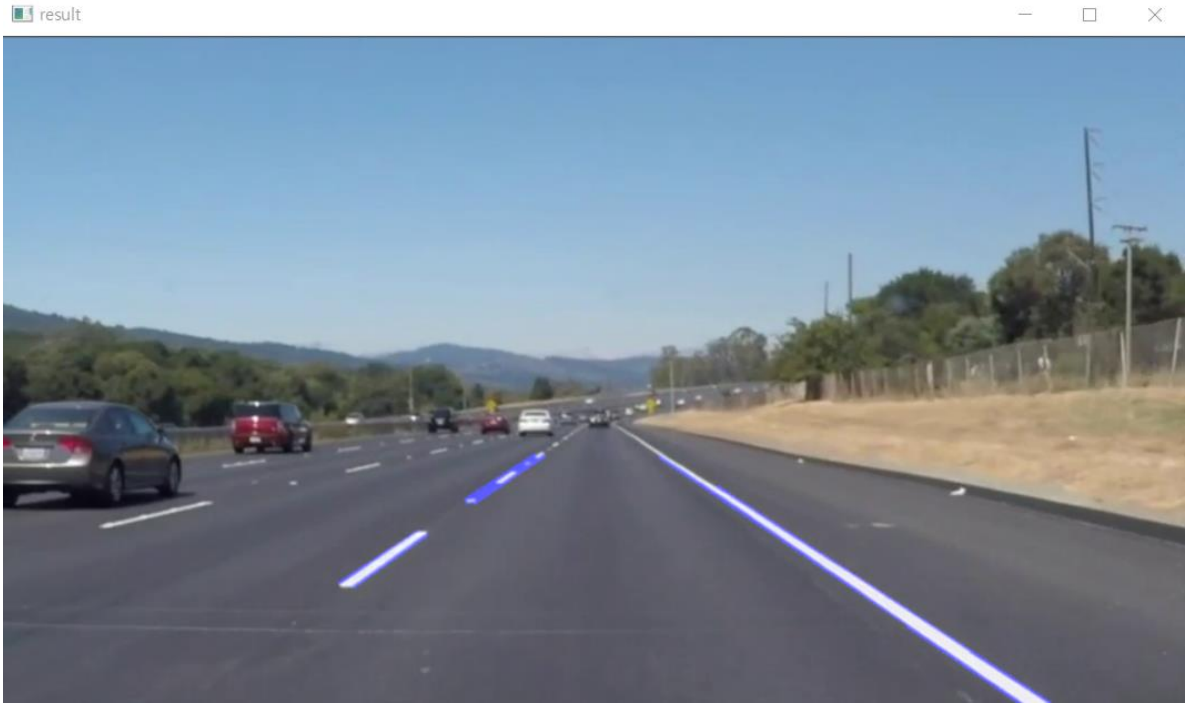
<main() 부분>



mask



ROI



result