차량지능기초

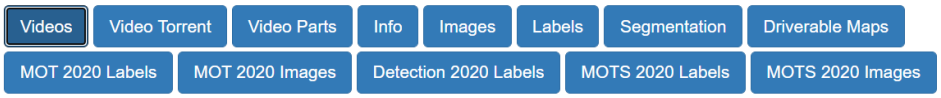
HW01

20171710 최근표

1.Data Set

1) Berkeley deep drive dataset(BDD)

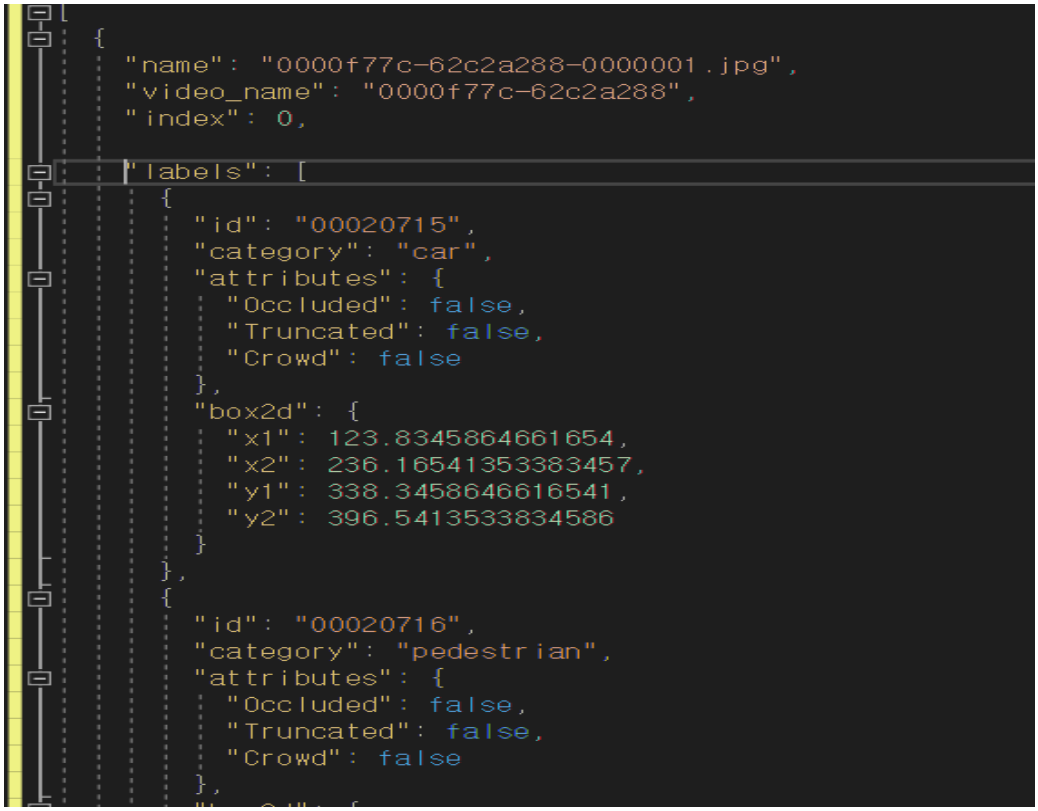
BDD의 데이터 셋은 뉴욕 및 샌프란시스코에서의 하루 중, 다양한 시간과 기후변화에 대한 정보로 구성되어있다.



기본적으로, BDD 데이터 셋은 종합된 연구를 위한 영상, 라벨과 이미지 등으로 구성되어있다.

또한, 교통표지, 사람, 신호등, 자전거 등을 구분하기위해 100,000개의 주석이 달린 이미지 바운딩 박스가 있고, 자전거 등을 구분하기위해 100,000개의 주석이 달린 이미지 바운딩 박스가 있고, 보행자 인스턴스도 그와 비슷한 수의 양이다. 이는 cityscapes과 비교하면 24,000배가 되는 굉장히 방대한 양의 데이터 셋이라고 할 수 있다.

데이터 셋의 바운딩을 담당하는 라벨을 확인해보면,



한 장의 사진에서 객체들을 구분하여 인식하는 것을 확인할 수 있다.

차의 id는 00020715이고, 속성을 통해 자율주행 차량을 막는지 여부등을 판단할 수 있는 것으로 보인다.

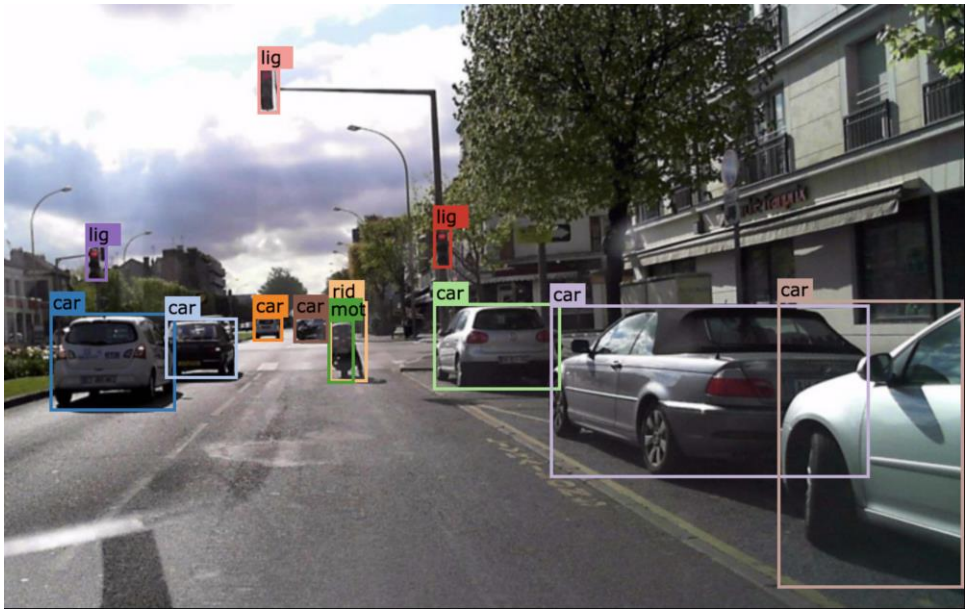
또한 보행자의 id는 00020716이고, 위와 같이 속성을 통해 주행에 영향을 끼치는지 판단했던 데이터를 볼 수 있다.

또 다른 detection label에서는 물체를 파악하고 교통 신호여부를 판단하는 data가 들어있음을 확인했다.

마지막 label인 segmentation label은 말 그대로 구분을 짓기 위한 label로 훈련용과 실제 데이터를 모아놓은 라벨이다.

앞서 살펴보았듯이 BDD는 거리에서의 객체를 감지하고, 구분을 짓는 게 가능한

데이터 셋을 갖고있다. 또한 차선을 구분할 수 있는 등의 기능을 가지는 것으로 미루어보아, 현재 자율주행 인지의 많은 요소들이 이미 데이터 셋에 포함되어 데이터를 활용하고 있다고 본다.



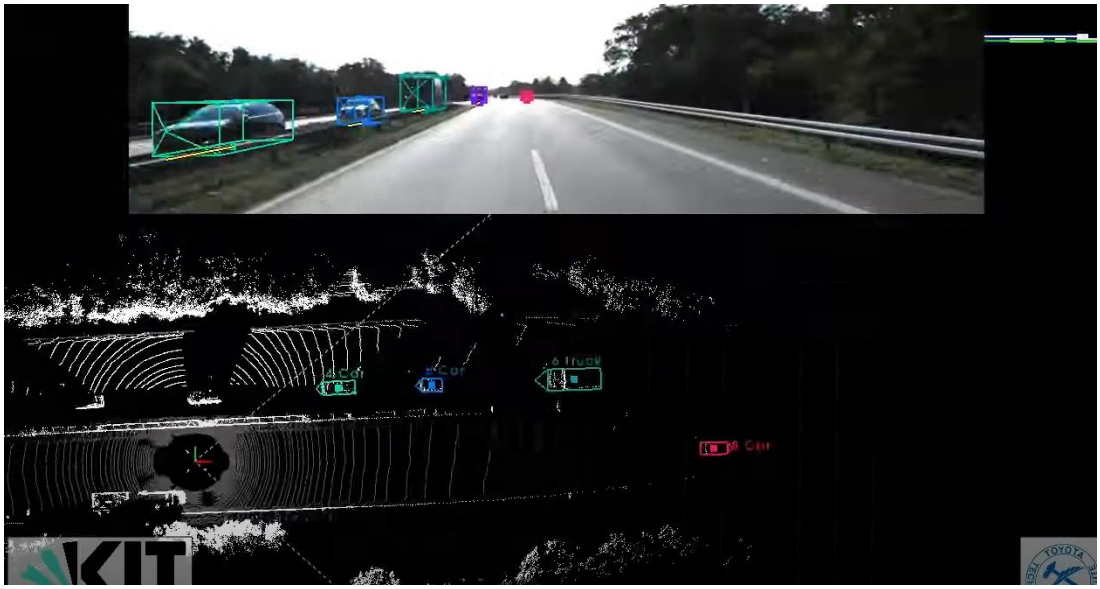
(객체 구분)

2)KITTI dataset

KITTI dataset은 카메라를 이용한 데이터 셋이다.

운행 중 촬영을 통해 객체를 인식하고, 객체와의 거리를 깊이 추정(Depth estimation)을 통해 값을 담아놓은 dataset이다.

앞서 말했듯이 KITTI의 데이터셋의 양은 BDD에 비하면 상당히 적은 편이라고 할 수 있는데, 이는 BDD가 방대한 면도 있지만 KITTI가 독일의 교통환경에서 수집된 정보로만 활용을 하였다. 또한, 맑은 날씨 한정이라는 치명적인 단점을 갖고 있기에 기후 변화에 따른 정보가 없는 것, 표지판같은 디자인을 인지 못한다는 점이 데이터의 양이 많지 않은 이유 중 하나이다.



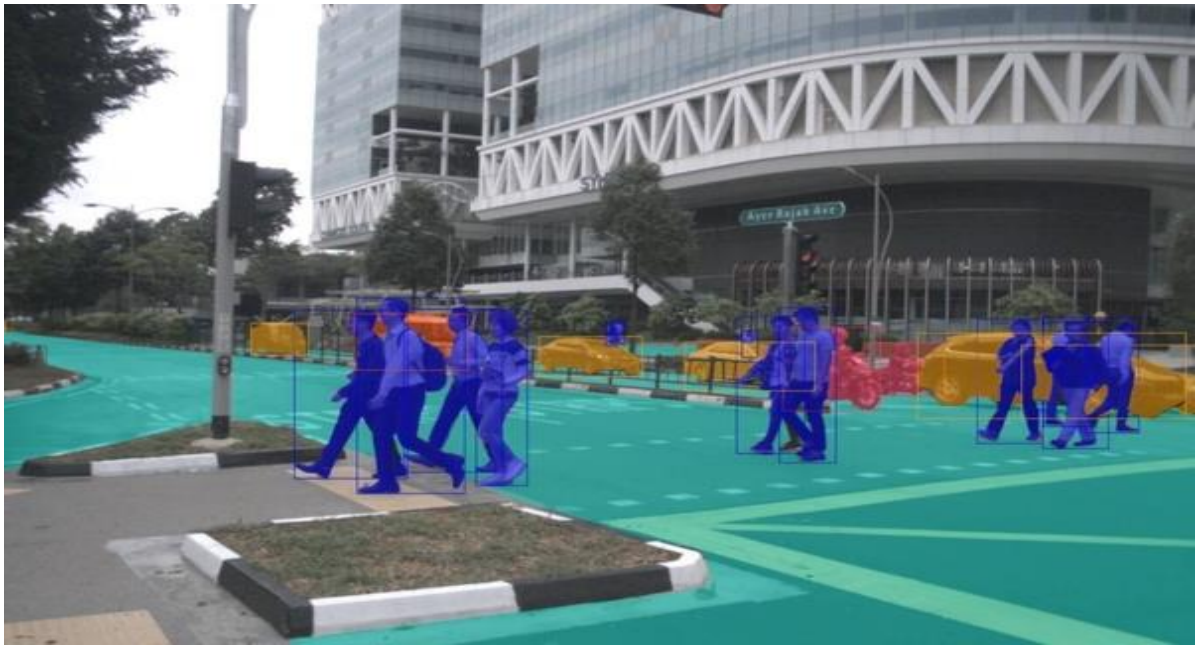
KITTI dataset은 카메라와 라이다를 활용한 자율주행 정보이므로, 이에 대한 기본적인 공부가 선행된다면 정보를 비교적 이해하기 쉬워질거라고 생각한다. 또한, 이것들을 통해 객체를 인식하고 구분하므로 객체인식 및 구분에 대해 학습이 선행되는 것도 이해하는 데 도움을 줄 수 있다.

3) nuScenes dataset

대규모 데이터셋을 활용하여 도시 주행 연구를 하기도 하는 dataset이다.

이 데이터셋은 기본적으로 라이다 1개, 레이더 5개, 카메라6개 를 활용해 객체를 인식하고 BDD와 같이 2D 바운딩 박스를 통해 객체를 구분한다.

여기에는 다양한 정보들이 들어있는데, 일단 보스턴과 싱가포르에 대한 운행정보가 들어있다. 또한 2D뿐만이 아닌 주석이 달린 3D 객체 바운딩 박스로 조금 더 정밀한 구분이 가능해진다.



그리고 KITTI에서는 구현되지 못 한 기후변화에 대한 정보가 많기 때문에 날씨가 자율운행에게 큰 영향을 끼치지않는다. 또한 데이터의 양은 KITTI의 7배 많은 주석과 100배에 달하는 이미지 양을 갖고있다. 이 데이터 셋 또한 객체인식 및 구분과 더불어 카메라와 레이더, 라이더에 대한 선행이 된다면 데이터 셋을 이해하는 데 도움이 될 것이라 생각한다.

2. Open Source

1) cvlib을 통한 객체인식

Sabina Pokhrel

AI Specialist | Machine Learning Engineer | Writer and former Editorial Associate at Towards Data Science

Follow

270 6

```
import cv2
import matplotlib.pyplot as plt
import cvlib as cv
from cvlib.object_detection import draw_bbox

im = cv2.imread('cars_4.jpeg')

bbox, label, conf = cv.detect_common_objects(im)

output_image = draw_bbox(im, bbox, label, conf)

plt.imshow(output_image)
plt.show()

print('Number of cars in the image is '+ str(label.count('car')))
```

cvlib을 통해 차량의 개수를 세는 간단한 소스코드이다.

처음에는 사진을 읽어들이고, bbox(객체 구분박스)와 label(각 객체의 이름표), conf(객체 구분 확률)로 구성한다. Detect_common_objects 라는 함수를 이용해 객체를 구분짓는다.

구분을 완료했으면 그에 걸맞는 박스와 라벨을 붙여 사진을 출력한다.

Car라는 label이 몇 개 붙었는지를 통해 차량의 수를 알 수 있다.

2)opencv를 통한 차선인식

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
%matplotlib inline

img = mpimg.imread('solidWhiteCurve.jpg')

plt.figure(figsize=(10,8))
print('This image is:', type(img), 'with dimensions:', img.shape)
plt.imshow(img)
plt.show()
```

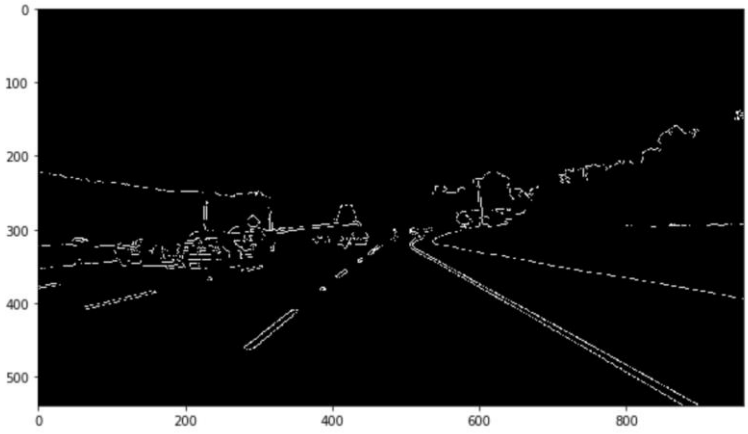
```
This image is: <class 'numpy.ndarray'> with dimensions: (540, 960, 3)
```


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

gray = grayscale(img)
plt.figure(figsize=(10,8))
plt.imshow(gray, cmap='gray')
plt.show()
```


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

kernel_size = 5
blur_gray = gaussian_blur(gray, kernel_size)

plt.figure(figsize=(10,8))
plt.imshow(blur_gray, cmap='gray')
plt.show()
```

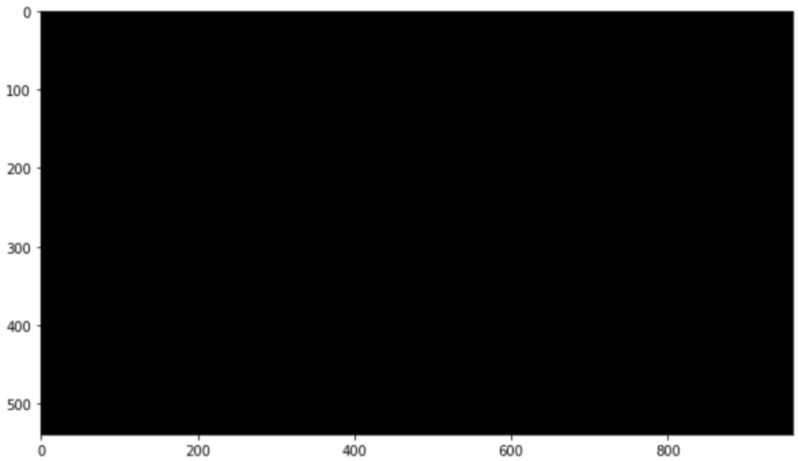
사진을 받아온 후, 보다 정확한 차선인식을 위해 흑백처리 및 가우시안 블러처리를 한다.

```python
def canny(img, low_threshold, high_threshold):
    """Applies the Canny transform"""
    return cv2.Canny(img, low_threshold, high_threshold)

low_threshold = 50
high_threshold = 200
edges = canny(blur_gray, low_threshold, high_threshold)

plt.figure(figsize=(10,8))
plt.imshow(edges, cmap='gray')
plt.show()
```



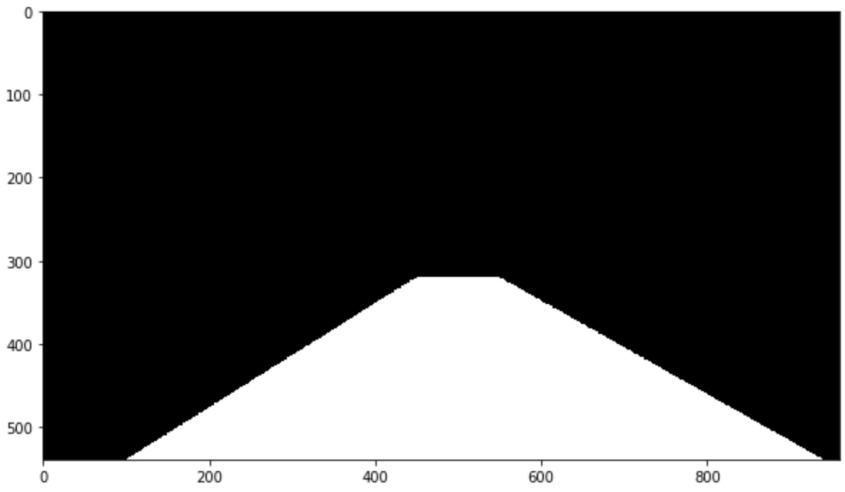
Canny edge detection을 활용하면 각 edge들을 딸 수는 있지만, 이는 차선을 구분하기는 힘들기 때문에 사용하지 않는다.

```python
import numpy as np

mask = np.zeros_like(img)

plt.figure(figsize=(10,8))
plt.imshow(mask, cmap='gray')
plt.show()
```



```
if len(img.shape) > 2:
    channel_count = img.shape[2]  # i.e. 3 or 4 depending on your image
    ignore_mask_color = (255,) * channel_count
else:
    ignore_mask_color = 255

imshape = img.shape
print(imshape)

vertices = np.array([[(100,imshape[0]),
                      (450, 320),
                      (550, 320),
                      (imshape[1]-20,imshape[0])]], dtype=np.int32)

cv2.fillPoly(mask, vertices, ignore_mask_color)

plt.figure(figsize=(10,8))
plt.imshow(mask, cmap='gray')
plt.show()
```

```
(540, 960, 3)
```

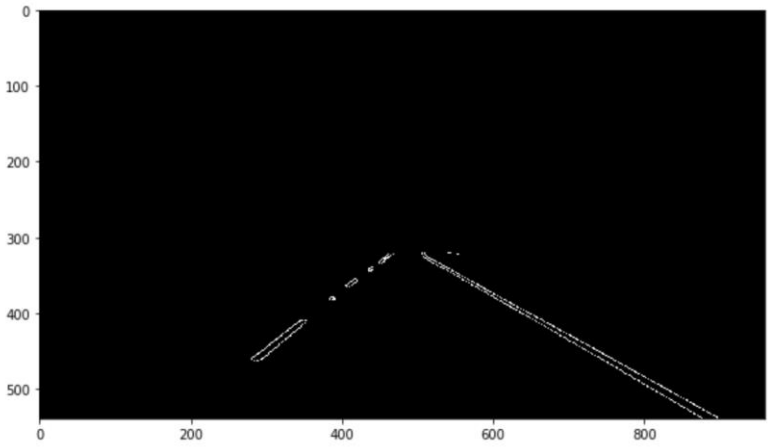


이후 화면을 검은색으로 채운 뒤, 하나의 도로에서만 차선을 인식하기 위해 vertices에서 범위를 설정한 뒤 인식하게 한다.

```
def region_of_interest(img, vertices):
    #defining a blank mask to start with
    mask = np.zeros_like(img)

    if len(img.shape) > 2:
        channel_count = img.shape[2]
        ignore_mask_color = (255,) * channel_count
    else:
        ignore_mask_color = 255

    cv2.fillPoly(mask, vertices, ignore_mask_color)

    #returning the image only where mask pixels are nonzero
    masked_image = cv2.bitwise_and(img, mask)
    return masked_image
```

```
imshape = img.shape
vertices = np.array([[(100,imshape[0]),
                      (450, 320),
                      (550, 320),
                      (imshape[1]-20,imshape[0])]], dtype=np.int32)
mask = region_of_interest(edges, vertices)

plt.figure(figsize=(10,8))
plt.imshow(mask, cmap='gray')
plt.show()
```

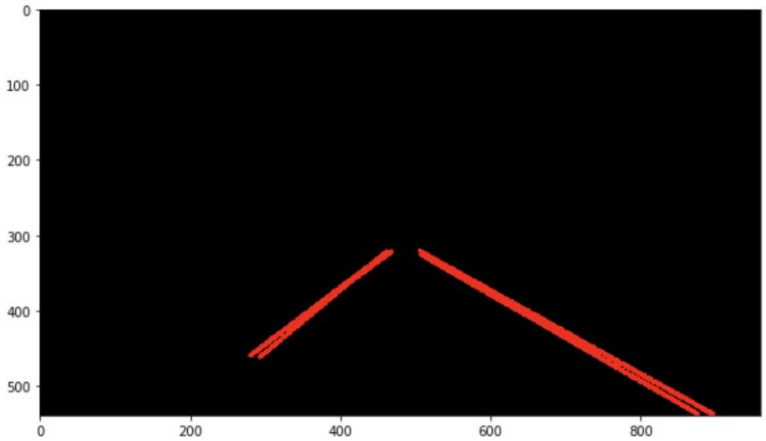
각 사이즈에 맞게 설정을 한 뒤, region_of_interest를 통해 edge로 인식되는 부분만 선을 딴다.



```
def draw_lines(img, lines, color=[255, 0, 0], thickness=5):
    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):
    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
```

```
rho = 2
theta = np.pi/180
threshold = 90
min_line_len = 120
max_line_gap = 150

lines = hough_lines(mask, rho, theta, threshold,
                    min_line_len, max_line_gap)

plt.figure(figsize=(10,8))
plt.imshow(lines, cmap='gray')
plt.show()
```

허프변환 함수를 구현하여, edge를 인식하여 그에 빨간색 선을 그어 차선이 어디까지 있는지를 알려주고 인식하게 한다.

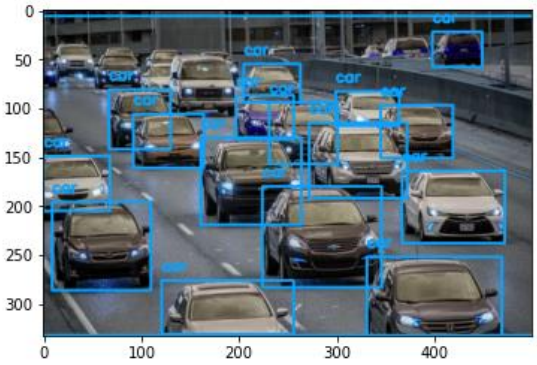



```
def weighted_img(img, initial_img, α=0.8, β=1., λ=0.):
    return cv2.addWeighted(initial_img, α, img, β, λ)

lines_edges = weighted_img(lines, img, α=0.8, β=1., λ=0.)

plt.figure(figsize=(10,8))
plt.imshow(lines_edges)
plt.show()
```

그 후, 검은색에서 선만을 따와 원본이미지에 덮으면 우리가 영상에서 보던 차선 인식과 비슷한 형태를 띄게된다.

3. 2)의 정리한 코드 중 하나 실행해서 결과 확인

In [2]:
```
import cv2
import matplotlib.pyplot as plt
import cvlib as cv
from cvlib.object_detection import draw_bbox
im = cv2.imread('img1.jpeg')
bbox, label, conf = cv.detect_common_objects(im)
output_image = draw_bbox(im, bbox, label, conf)
plt.imshow(output_image)
plt.show()
print('Number of cars in the image is '+ str(label.count('car')))
```




```
Number of cars in the image is 17
```

이 코드는 opencv2, cvlib 라이브러리를 통해 구현되었다. Cvlib은 파이썬에서 얼굴이나 사물과 같은 객체를 인식하는 기능을 갖춘 사용하기 쉬운 라이브러리이다. 이 코드는 그런 점을 활용하여 간단하면서도 이해하기 쉽게 구성이 되어있다.

이 코드를 실행하는 환경은 운영체제는 windows 10을 사용하였고,

코드는 아나콘다 명령 프롬프트를 통한 jupyter notebook에서 실행하였다.

4) Github

https://github.com/geunpyochoi

참고자료

https://bdd-data.berkeley.edu/

http://www.cvlibs.net/datasets/kitti/

https://www.nuscenes.org/

https://towardsdatascience.com/count-number-of-cars-in-less-than-10-lines-of-code-using-python-40208b173554

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