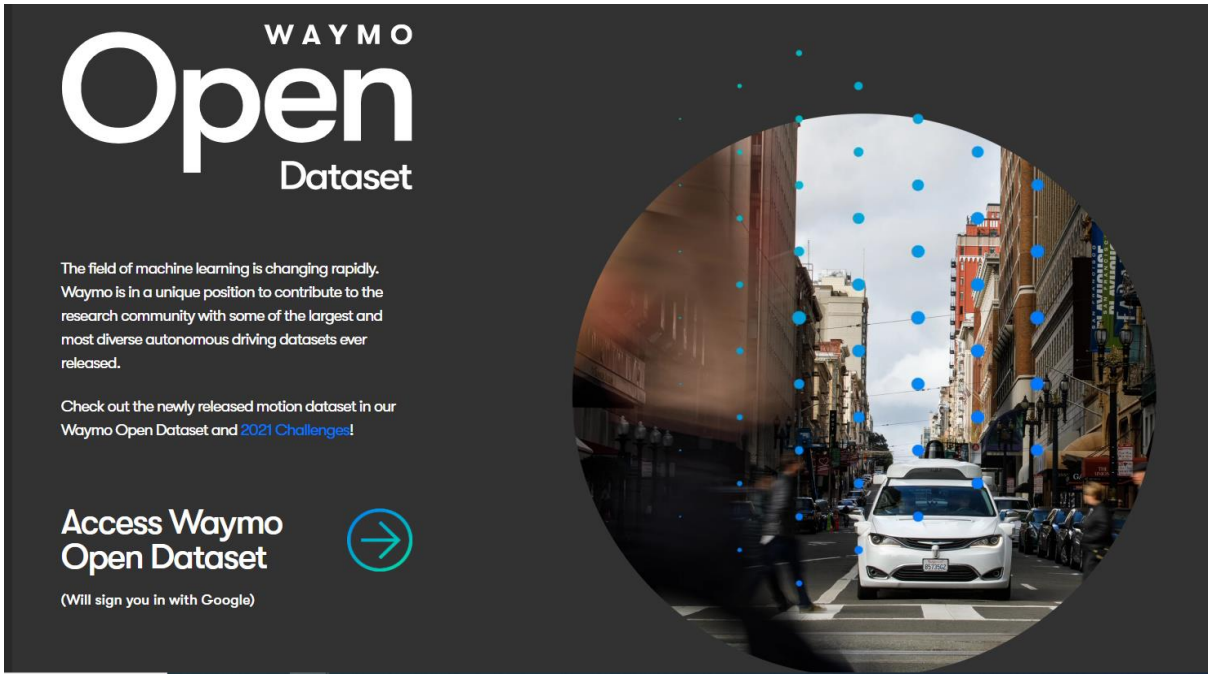
WAYMO Open Dataset



웨이모 오픈 데이터세트

웨이모는 25개 도시에서 약 1000만 마일에 달하는 자율주행 테스트를 하며 데이터를 수집했습니다. 2019 년 8 월에 고해상도 센서 데이터와 1,950 개의 세그먼트에 대한 레이블로 구성된 인식 데이터 세트로 처음 출시되었고. 2021 년 3 월, 데이터 세트를 확장하여 103,354 개의 세그먼트에 대한 객체 궤적과 해당 3D지도로 구성된 모션 데이터 세트도 포함했습니다. 웨이모는 연구 커뮤니티가 기계 인식 및 자율 주행 기술을 발전시킬 수 있도록 데이터세트를 공개적으로 출시했습니다

오픈 데이터세트에는 1,000개의 주행 세그먼트 데이터가 포함되어 있고, 각 세그먼트는 센서 당 10Hz(20만 프레임)로 수집된, 20초 동안의 주행 데이터로 구성되어 있습니다. 이러한 연속 주행 영상은 다른 도로에서 사용자의 행동을 추적하고 예측하는 모델을 개발하는 데 활용할 수 있습니다. 또한 각 세그먼트에는 5개의 고해상도 웨이모 라이더(LIDAR)와 5개의 전면 카메라 센서 데이터가 제공됩니다.
데이터세트는 지역과 시간, 날씨 등이 다른 다양한 주행 환경에서 수집된 자료들을 활용할 수 있게 되어 있다. 테스트 주행 구간은 피닉스, 커클랜드, 마운틴 뷰, 캘리포니아와 샌프란시스코 등으로, 도심과 교외 구간의 주행 기록이 포함되어 있습니다. 밤, 낮, 새벽, 황혼, 태양 빛이 강할 때와 비 내리는 날씨 등 시간과 날씨에 따른 다행한 주행 환경 데이터가 제공됩니다..

아울러 자동차, 보행자, 자전거, 표지만 4가지로 꼼꼼하게 구분해 표기한 라벨이 포함된 이미지와 라이다 프레임이 포함되어 있습니다. 전체 라벨은 120만 개의 2D 라벨과 약 1,200만 개의 3D 라벨을 포함하고 있습니다. 센서 데이터는 미드 레인지 라이더 1개, 단거리 라이더 4개, 전면과 측

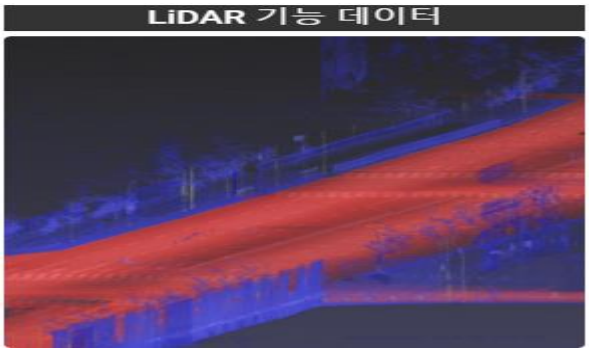
면에 장착된 카메라 5대, 동기화된 라이더와 카메라 데이터, 카메라 투영에 대한 라이더, 센서 교정 및 차량 자세 자료로 구성되어 있습니다. 또 웨이모 데이터세트는 광범위한 주행조건 예를 들면 낮과 밤, 새벽, 햇빛, 비 등을 캡처하 밀집한 도시 및 교외 환경을 포함하고 있습니다

네이버 데이터 세트

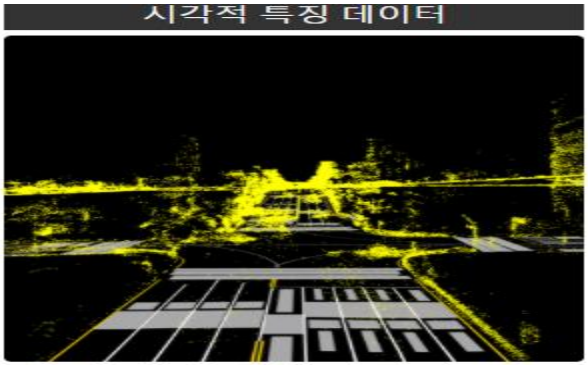
네이버 랩스 매핑 기술은 도시 규모의 항공 사진과 모바일 매핑 시스템의 데이터를 통합한 것입니다. 항공 사진에서 노면 레이아웃 (3D 도로 레이아웃) 정보를 추출합니다. 그런 다음이를 경량 모바일 매핑 시스템 (MMS) 인 R1에서 수집 한 3D 포인트 클라우드와 통합합니다. MMS 차량으로 제작된 기존의 HD지도와 비교하여 당사의 매핑 프로세스는 제작 비용과 시간을 크게 줄일 수 있습니다. 또한 평가 목적으로 사용할 수 있는 센서 데이터와 의사 지상 실측 포즈가 포함된 별도의 현지화 데이터 세트를 제공합니다.

센서 종류

유형	공급 업체	모델	수량	기술	장치 세부 정보			링크
					Hz	정확성	범위	
카메라	FLIR	GS3-U3-15S5C-C	8	1384 x 1032 (1.4 메가 픽셀) / 컬러, 45FPS, 글로벌 셔터	10Hz			URL
3D LiDAR	벨로 다인	VLP-32C	1	32 채널 LiDAR, 360 ° FoV	10Hz		200m	URL
	벨로 다인	VLP-16	4	16 채널 LiDAR, 360 ° FoV	10Hz		100m	URL
2D LiDAR	병든	LMS151-10100	1	1 채널 LiDAR, 270 ° FoV	50Hz	0.5 ° (분해능)		URL
안개	KVH	DSP-1760	1	3 축 광섬유 자이로	1000Hz	0.05 ° / 시간		URL
GPS / INS	SBG 시스템	Ekinox-D	1	소비자 수준의 GPS / INS	50Hz (필터링 됨) 200Hz (IMU)	위치 Acc : ~1cm (RTK / INS, RMS) 속도 Acc : 0.03 m / s (RMS) 방향 Acc : 0.1 °		URL
OBD	-	-	-	Prius V의 내부 신호	80Hz			URL
휠 인코더	인코더 제품 회사	TR1	2	-	100Hz	500 (CPR)		URL
* 현지화 데이터 세트와 함께 센서 교정 정보를 제공합니다.								

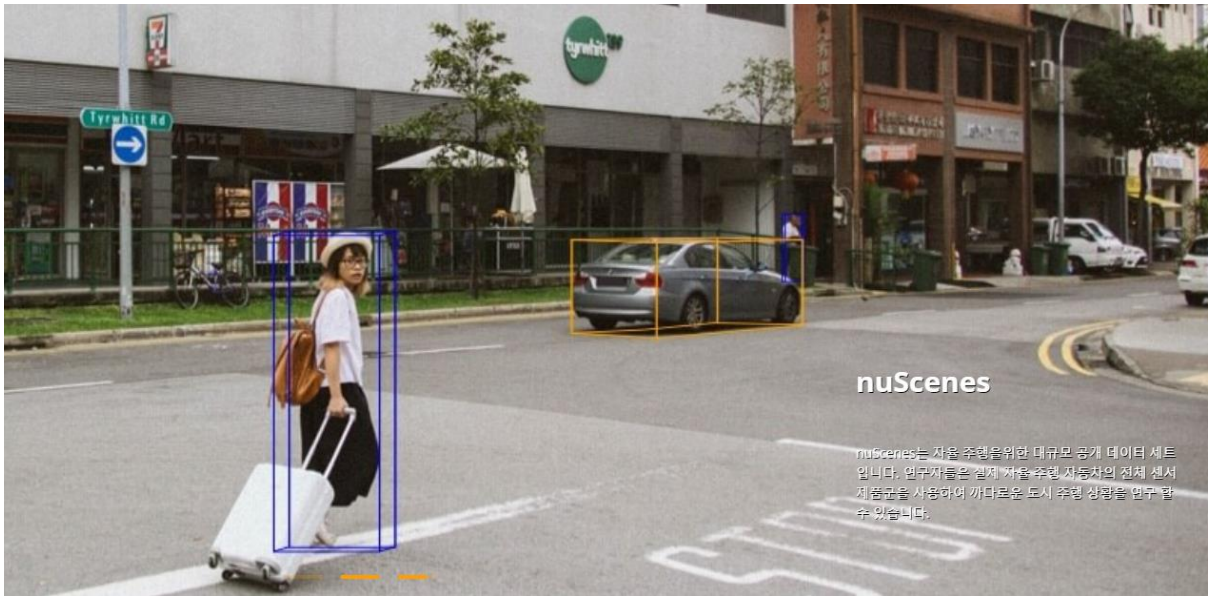


HD맵 데이터 세트에는 MMS차량에서 캡처 한 주변 도로 환경의 3D LiDAR 포인트 클라우드로 포함됩니다. 각 포인트는 그것이 속한 객체 또는 영역의 유형을 나타내는 의미 레이블과 연관됩니다. 차량 및 보행자와 같은 동적 개체가 자동으로 감지되어 포인트 클라우드에서 제거됩니다.



HD지도 데이터 세트의 마지막 구성요소는 도로 환경의 두드러진 지역에서 추출한 시각적 특징 세트입니다. 훈련 된 딥 러닝 모델을 통해 얻은 이러한 기능은 컴팩트하고 차별적이며 다양한 시각 조건에서 변하지 않으므로 안정적인 매칭 및 현지화가 가능합니다

nuScenes Dataset



nuScenes 데이터는 자율주행을 위한 공개 대규모 데이터 세트입니다. 무인 차량을 안전하고 신뢰할 수 있으며 접근 가능한 현실로 만들고 있습니다. 데이터의 일부를 대중에게 공개함으로써 컴퓨터 비전 및 자율 주행에 대한 대중 연구를 지원하는 것을 목표로 합니다.

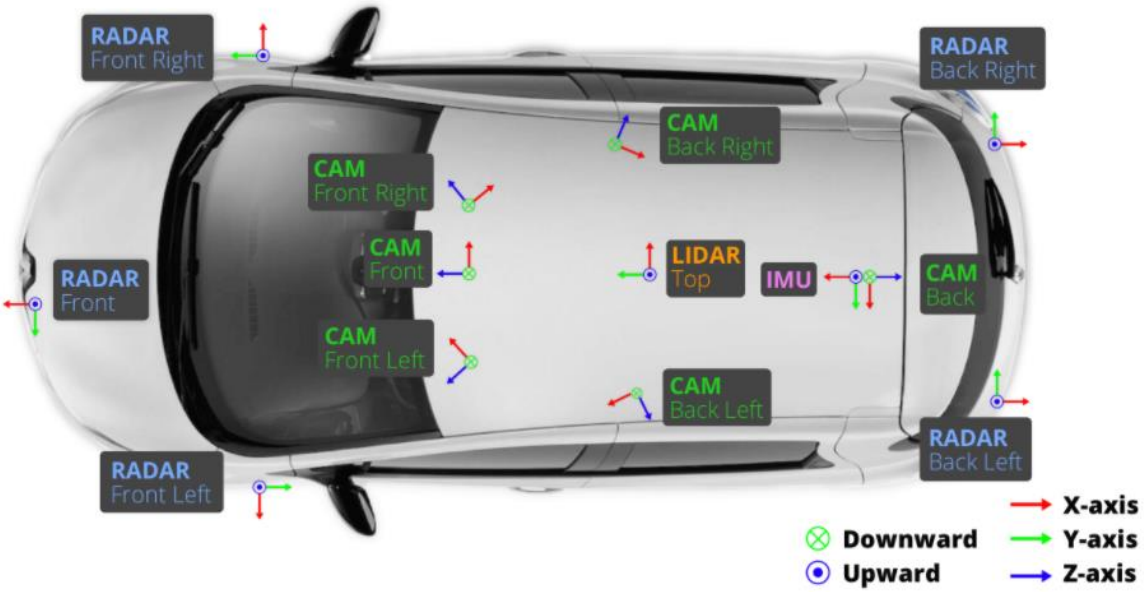
이를 위해 우리는 교통량이 많고 운전 상황이 매우 까다로운 두 도시인 보스턴과 싱가포르에서 1000 개의 운전 장면을 수집했습니다. 20 초 길이의 장면은 다양하고 흥미로운 운전 기동, 교통 상황 및 예상치 못한 행동을 보여주기 위해 수동으로 선택됩니다. nuScenes의 풍부한 복잡성은 장면 당 수십 개의 물체가 있는 도시 지역에서 안전한 운전을 가능하게 하는 방법의 개발을 장려합니다. 다른 대륙에서 데이터를 수집하면 다양한 위치, 기상 조건, 차량 유형, 초목, 도로 표시 및 왼손 대 오른손 교통 전반에 걸친 컴퓨터 비전 알고리즘의 일반화를 연구할 수 있습니다.

nuScenes 데이터 세트의 경우 보스턴과 싱가포르에서 약 15 시간의 운전 데이터를 수집합니다. 까다로운 시나리오를 포착하기 위해 운전 경로가 신중하게 선택됩니다. 다양한 위치, 시간 및 기상

조건을 목표로 합니다. 클래스 빈도 분포의 균형을 맞추기 위해 희귀 클래스가 있는 장면을 더 많이 포함합니다. 이러한 기준을 사용하여 각각 20 초 길이의 장면 1000 개를 수동으로 선택합니다.

센서

자동차에 있는 센서



LIDAR

우리는 레이저 라이너를 사용하여 자아 프레임에 대한 LIDAR의 상대적 위치를 정확하게 측정합니다.

카메라 외부

카메라와 LIDAR 센서 앞에 입방체 모양의 보정 대상을 배치합니다. 보정 대상은 알려진 패턴을 가진 세 개의 직교 평면으로 구성됩니다. 패턴을 감지한 후 보정 대상의 평면을 정렬하여 카메라에서 LIDAR 로의 변환 행렬을 계산합니다. 위에서 계산된 LIDAR에서 자아 프레임으로의 변환이 주어지면, 우리는 카메라에서 자아로의 프레임 변환과 그에 따른 외부 매개 변수를 계산할 수 있습니다.

RADAR

레이더를 수평 위치에 장착합니다. 그런 다음 도시 환경에서 운전하여 레이더 측정 값을 수집합니다. 움직이는 물체에 대한 레이더 리턴을 필터링 한 후 무차별 대입 방식을 사용하여 요 각도를 보정하여 정적 물체에 대한 보상 범위 비율을 최소화합니다.

2) 자율주행 인지에 관련된 2종 이상 Open Source 조사, 정리

구글 웨이모에서 제공해주는 카메라 이미지 및 카메라 레이블을 시각화 해주는 오픈 소스입니다.

아래 코드설명을 하기 전 웨이모에서 제공해주는 카메라 데이터와 2D 카메라 라벨데이터에 대해 먼저 소개하겠습니다.

카메라 데이터

먼저 웨이모의 카메라 데이터 세트에는 5개의 서로 다른 방향의 5개의 카메라 이미지가 포함되어 있습니다. 전면, 전면 죄측, 전면 우측, 측면 좌측, 측면 오른쪽 이렇게 5개의 카메라 이미지가 있습니다. 각 카메라 이미지는 하나씩 제공됩니다. JPEG 형식으로, 차량 포즈, 이미지 중심의 노출 시간에 해당하는 속도 및 롤링 셔터 타이밍 정보도 제공됩니다.

2D 카메라 라벨데이터

웨이모에서는 카메라 이미지에 2D 경계 상자 레이블을 제공합니다. 카메라 레이블은 세계적으로 고유한 추적ID가 있는 알맞은 축 정렬 2D경계 상자입니다. 이 경계 상자는 개체의 보이는 부문만 덮어줍니다. 차량, 보행자, 자전거의 개체에 2D레이블이 되어있습니다.

일반 라벨링 사양

2D 경계 상자는 카메라 이미지의 개체 주위에 가능한 한 단단하게 그려지고 개체의 모든 보이는 부분을 캡처합니다.

상자는 개체의 보이는 부분 만 캡처하고 부분적으로 가려진 개체의 범위를 추정하지 않습니다.

카메라 이미지와 반사에서 보이는 자율 주행 차량의 일부에는 라벨이 지정되어 있지 않습니다.

수평선에 개체가 너무 많아서 각 개체에 개별적으로 레이블을 지정하면 개별적으로 레이블을 지정할 수 있는 개체에 대해서만 레이블이 만들어지고 나머지 개체는 무시됩니다.

라벨 유형

2D 경계 상자 레이블은 주행 구간의 차량, 보행자 및 자전거에 포함됩니다. 다른 개체는 이 데이터 세트에서 라벨이 지정되지 않습니다.

카메라 이미지와 카메라 라벨을 시각화 하는 코드

#matplotlib의 patches와 pytlot을 받아서 도형을 추가하고 그릴 수 있게 해줍니다.

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
```

#카메라 이미지와 주어진 라벨을 표현해 주는 함수를 정의해줍니다.

```
  def show_camera_image(camera_image, camera_labels, layout, cmap=None)

      ax = plt.subplot(*layout)

      #카메라 라벨을 그려 줍니다
      for camera_labels in frame.camera_labels:
        # 카메라와 다른 카메라 라벨은 무시 합니다.
        if camera_labels.name != camera_image.name:
          continue

        # 개별 레이블을 반복합니다.
        for label in camera_labels.labels:
          # 개체별로 경계선을 그어 줍니다.
          ax.add_patch(patches.Rectangle(
            xy=(label.box.center_x - 0.5 * label.box.length,
                label.box.center_y - 0.5 * label.box.width),
            width=label.box.length,
            height=label.box.width,
            linewidth=1,
            edgecolor='red',
            facecolor='none'))

      # 카메라 이미지를 보여줍니다.
      plt.imshow(tf.image.decode_jpeg(camera_image.image), cmap=cmap)
      plt.title(open_dataset.CameraName.Name.Name(camera_image.name))
      plt.grid(False)
      plt.axis('off')

   plt.figure(figsize=(25, 20))

for index, image in enumerate(frame.images):
  show_camera_image(image, frame.camera_labels, [3, 3, index+1])
```

====실행결과====

웨이모에서 위 오픈 소스는 각각의 위치에서 카메라로 받아온 이미지들을 보고 그 이미지 속에 어떠한 개체들이 있는 지 확인해주며 개체에 빨간 직사각형 틀을 입혀 줍니다. 이를 활용하면 주변에 자동차나 사람 또는 어떠한 물체가 있는 지 확인을 해주고 판단할 수 있을 것 같습니다.

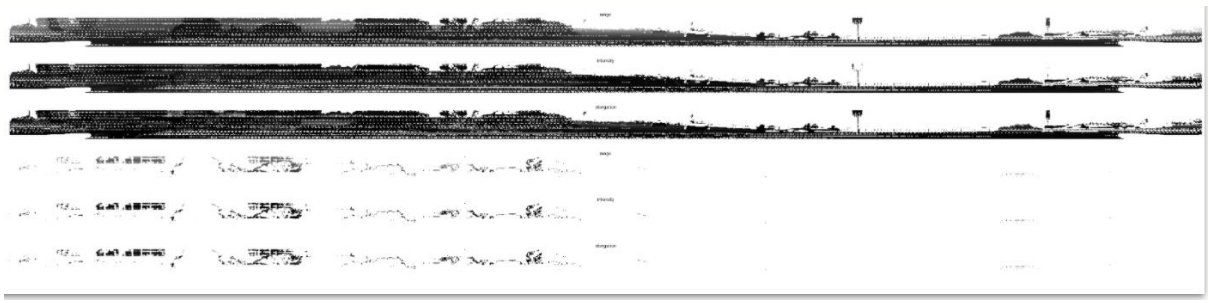
위의 웨이모의 코드를 보면 카메라의 받아온 이미지들을 분석하여서 판단하고 시각화해주는 것을 보여주는 것 같습니다. Plt를 잘 활용해서 각각의 개체들을 잘 나타내 주는 것 같습니다.

웨이모에서 제공해주는 범위 이미지 시각화 오픈소스

```
#이번 코드도 위 코드와 마찬가지로  plt 로 그래프를 그려줍니다.
  plt.figure(figsize=(64, 20))
  def plot_range_image_helper(data, name, layout, vmin = 0, vmax=1, cmap
  ='gray')
  #범위 이미지를 그립니다.
  데이터 : 범위 이미지 데이터
  이름 : 이미지 제목
  레이아웃 : plt 레이아웃
  vmin : 전달 된 데이터의 최솟값
  vmax : 전달 된 데이터의 최대값
  cmap : 컬러 맵

  plt.subplot(*layout)
    plt.imshow(data, cmap=cmap, vmin=vmin, vmax=vmax)
    plt.title(name)
    plt.grid(False)
    plt.axis('off')
def get_range_image(laser_name, return_index):
  #레이저 이름과 반환 인덱스가 지정한 범위 이미지를 반환합니다.
  return range_images[laser_name][return_index]

def show_range_image(range_image, layout_index_start = 1):
  범위 이미지를 표시합니다.

  # Args:
      Range_image : MatrixFloat 유형의 지정된 라이더로부터의 범위 이미지

  range_image_tensor = tf.convert_to_tensor(range_image.data)
  range_image_tensor = tf.reshape(range_image_tensor, range_image.shape
.dims)
  lidar_image_mask = tf.greater_equal(range_image_tensor, 0)
  range_image_tensor = tf.where(lidar_image_mask, range_image_tensor,
```

```
                                    tf.ones_like(range_image_tensor) * 1e10
)
  range_image_range = range_image_tensor[...,0]
  range_image_intensity = range_image_tensor[...,1]
  range_image_elongation = range_image_tensor[...,2]
  plot_range_image_helper(range_image_range.numpy(), 'range',
                    [8, 1, layout_index_start], vmax=75, cmap='gray')
  plot_range_image_helper(range_image_intensity.numpy(), 'intensity',
                    [8, 1, layout_index_start + 1], vmax=1.5, cmap='gray
')
  plot_range_image_helper(range_image_elongation.numpy(), 'elongation',
                    [8, 1, layout_index_start + 2], vmax=1.5, cmap='gray
')
frame.lasers.sort(key=lambda laser: laser.name)
show_range_image(get_range_image(open_dataset.LaserName.TOP, 0), 1)
show_range_image(get_range_image(open_dataset.LaserName.TOP, 1), 4)
```

====실행결과====



라이더로부터 받은 데이터들을 이용하여 물체와의 범위를 측정하고 이것을 시각화하여 그래프로 나타내 줍니다. Range_image라는 라이더 범위 이미지를 사용하여 데이터들을 표현해 줬다.

nuScenes에서 제공해주는 오픈소스

```
//먼저 초기화를 해줍니다
%matplotlib inline
```

```
from nuscenes.nuscenes import NuScenes
nusc = NuScenes(version='v1.0-
mini', dataroot='/data/sets/nuscenes', verbose=True)
   ====실행 결과====
       Loading nuImages tables for version v1.0-mini...
       Done loading in 0.000 seconds (lazy=True).
#카테고리화 해줍니다. 예 (고양이, 개, 사람 등등,,,), 카테고리 로드
nuim.category[0]
      =====실행 결과=====
      Loaded 25 category(s) in 0.003s,
#목록을 보려면 table_name 을 이용하면 된다
nuim . table_names
      ====실행 결과====
   [ 'attribute',
    'calibrated_sensor',
    'category',
    'ego_pose',
    'log',
    'object_ann',
    'sample',
    'sample_data',
    'sensor',
    'surface_ann']


#샘플 데이터를 가져와 줍니다.
sample_idx = 0
sample = nuim.sample[sample_idx]
sample
sample = nuim.get('sample', sample['token'])
sample
sample_idx_check = nuim.getind('sample', sample['token'])
assert sample_idx == sample_idx_check
key_camera_token = sample['key_camera_token']
print(key_camera_token)
```

#랜더링 함수 render_image()함수를 이용해서 차량이랑 도로들을 나타내줍니다. 색상으로 물체들을 표현해 줍니다. 차량 : 주황색, 자전거 및 오토바이 : 빨간색, 보행자 : 파란색, 벽 : 회색, 운전 가능 구역 : 녹색으로 표현합니다.

With_category = True인 경우에 개체의 범주의 이름을 표현해줍니다. With_attrubutes = True로 각 개체의 속성을 나타내도록 설정해 줬습니다.

nuim . render_image (key_camera_token , annotation_type = 'all' , with_category = **True** , with_attributes = **True** , box_line_width = -1 , render_scale = 5)

====실행결과====

```
Loaded 1300 sample_data(s) in 0.013s,
Loaded 58 surface_ann(s) in 0.002s,
Loaded 506 object_ann(s) in 0.003s,
Loaded 12 attribute(s) in 0.000s,
```

#get_segmentation()을 이용하여 인스턴스들을 분리해 줍니다.

import matplotlib.pyplot as plt

semantic_mask, instance_mask = nuim.get_segmentation(key_camera_token)

plt.figure(figsize=(32, 9))

plt.subplot(1, 2, 1)

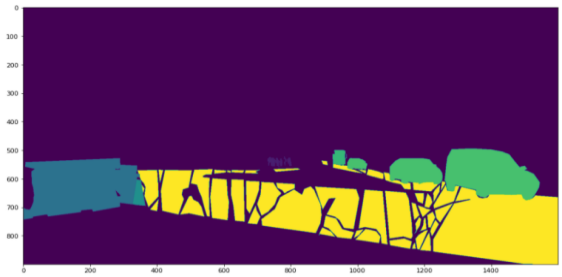
plt.imshow(semantic_mask)

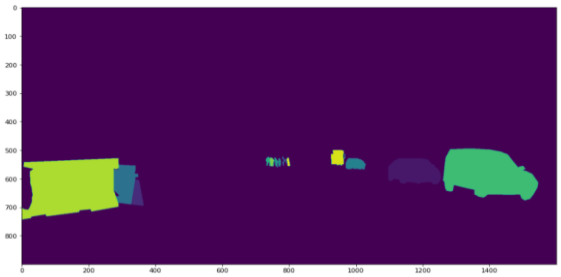
plt.subplot(1, 2, 2)

plt.imshow(instance_mask)

plt.show()

====실행결과====





nuScense에서 데이터 세트를 받아 카메라로 이미지를 받아오면 각각의 객체들은 인식하고 객체들을 카테고리화 시켜둬서 색깔별로 어떤 객체인지 분리할 수 있다. 이를 활용하여 전방에 자동

차, 사람 또는 나무 등 어떠한 사물이 있는지 인식할 수 있을 것 같다.

Waymo 오픈소스 개발환경

```
!rm -rf waymo-od > /dev/null
!git clone https://github.com/waymo-research/waymo-open-
dataset.git waymo-od
!cd waymo-od && git branch -a
!cd waymo-od && git checkout remotes/origin/master
!pip3 install --upgrade pip
```

웨이모 데이터 세트를 실행하기위해 웨이모의 git 저장소를 저장해주고 그 내용들을 받아옵니다

```
import os
import tensorflow.compat.v1 as tf
import math
import numpy as np
import itertools

tf.enable_eager_execution()

from waymo_open_dataset.utils import range_image_utils
from waymo_open_dataset.utils import transform_utils
from waymo_open_dataset.utils import  frame_utils
from waymo_open_dataset import dataset_pb2 as open_dataset
```

웨이모 오픈소스를 돌리기 위해서는 위의 모듈들을 가져와야 합니다.

#import os

파이썬에서 기본적으로 제공되는 os 모듈을 사용해야 합니다. Os 모듈을 사용해서 운영체제에서 제공되는 여러 기능들을 파이썬에서 수행 할 수 있게 해줍니다. 오픈소스를 돌리면서 필요한 파일을 복사하거나 디렉토리를 사용하기위해서 os 모듈을 이용해 줍니다.

import tensorflow.compat.v1 as tf

이 코드는 텐서플로 1.x 버전의 코드를 수정하지 않고 텐서플로 2.0에서 실행할 수 있게 해주는 코드 입니다. 이 작업을 통해서 성능을 최적화하고 간소화된 API의 이점을 이용할 수 있습니다.

Ex) cp_points_all_concat_tensor = tf.constant(cp_points_all_concat)

```
cp_points_all_tensor = tf.constant(cp_points_all, dtype=tf.int32)
```

#import math

파이썬 모듈인 math을 이용하여 더 많은 수학적 메서드를 이용 할 수 있게 해주며 다양한 수학 공식들을 사용 할 수 있습니다.

#import itertools

파이썬에서 제공하는 반복자를 만드는 모듈입니다. 이 모듈은 자체적으로 혹은 조합하여 유용하고 빠르게 메모리를 효율적이게 표준화합니다. 파이썬에서 간결하고 효율적을 특수화된 도구를 구성할 수 있도록 하는 이터레이터 대수를 형성합니다.

#from waymo_open_dataset.utils import range_image_utils

Waymo_open_dataset.utils에서 range_image_utils 모듈을 가지고 온다.

이 모듈은 이미지 범위를 관리하는 모듈입니다.

이 모듈 안에 있는 함수들 입니다.

```
def _combined_static_and_dynamic_shape(tensor):
```

```
static_tensor_shape = tensor.shape.as_list()
                          dynamic_tensor_shape = tf.shape(input=tensor)
                          combined_shape = []
                          for index, dim in enumerate(static_tensor_shape):
                            if dim is not None:
                              combined_shape.append(dim)
                            else:
                              combined_shape.append(dynamic_tensor_shape[index])
                          return combined_shape
```

차원에 대해서 정적 및 동적 값이 있는 목록을 리턴합니다. 이 함수는 모양 변경 작업에서 사용 가능한 경우 정적 모양을 보존하는데 유용합니다.

```
def _encode_range(r):
```

```
encoded_r = r / _RANGE_TO_METERS
                                      with tf.control_dependencies([

                                    tf.compat.v1.assert_non_negative(encoded_r),
```

```
                                        tf.compat.v1.assert_less_equal(encoded_r,
                                    math.pow(2, 16) - 1.)
                                      ]):
                                        return tf.cast(encoded_r, dtype=tf.uint16)
```

라이더 범위를 float에서 uint16으로 바꿔줍니다

r은 라이더 범위를 나타냅니다. 그리고 uint16인 인코딩된 범위를 리턴해 줍니다

```
def _decode_range(r):
return tf.cast(r, dtype=tf.float32) * _RANGE_TO_METERS
```

라이더의 범위를 정수에서 float32로 바꿔 줍니다.

이 함수는 디코딩 된 범위를 리턴 합니다.

```
def _encode_intensity(intensity):
```

```
If intensity.dtype !=tf.float32:
                                        raise TypeError('intensity must be of type
                                    float32')

                                      intensity_uint32 = tf.bitcast(intensity,
                                    tf.uint32)
                                      intensity_uint32_shifted =
                                    tf.bitwise.right_shift(intensity_uint32, 16)
                                      return tf.cast(intensity_uint32_shifted,
                                    dtype=tf.uint16)
```

라이더의 intensity를 float에서 uint16으로 인코딩 해줍니다. 여기에 저장된 정수 값은 float 타입의 16비트입니다. 소수점 3자리 까지 나타내도록 해줍니다. 그리고 uint32인 인코딩된 intensity를 리턴해 줍니다.

```
def _decode_intensity(intensity):
```

```
if
intensity.dtype !=
tf.uint16:
                          raise TypeError('intensity must be of type uint16')

                        intensity_uint32 = tf.cast(intensity, dtype=tf.uint32)
                        intensity_uint32_shifted =
                      tf.bitwise.left_shift(intensity_uint32, 16)
                        return tf.bitcast(intensity_uint32_shifted, tf.float32)
```

라이더의 intensity를 uint16에서 float32로 디코딩해줍니다

```
def _encode_elongation(elongation):
```

```
          encoded_elongation = elongation / _RANGE_TO_METERS
           with tf.control_dependencies([
              tf.compat.v1.assert_non_negative(encoded_elongation),
```

```
            tf.compat.v1.assert_less_equal(encoded_elongation, math.pow(2, 8) - 1.)
        ]):
          return tf.cast(encoded_elongation, dtype=tf.uint8)
```

Elongation을 float에서 uint8로 바꿔줍니다 그리고 리턴해 줍니다.

```
def _decode_elongation(elongation):
return tf.cast(elongation, dtype=tf.float32) * _RANGE_TO_METERS
```

이 함수는 uint8에서 float로 바꿔주고 리턴해 줍니다.

```
def encode_lidar_features(lidar_point_feature):
  if lidar_point_feature.dtype != tf.float32:
      raise TypeError('lidar_point_feature must be of type float32.')

  r, intensity, elongation = tf.unstack(lidar_point_feature, axis=-1)
  encoded_r = tf.cast(_encode_range(r), dtype=tf.uint32)
  encoded_intensity = tf.cast(_encode_intensity(intensity), dtype=tf.uint32)
  encoded_elongation = tf.cast(_encode_elongation(elongation), dtype=tf.uint32)

  encoded_r_shifted = tf.bitwise.left_shift(encoded_r, 16)

  encoded_intensity = tf.cast(
      tf.bitwise.bitwise_or(encoded_r_shifted, encoded_intensity),
      dtype=tf.int64)
  encoded_elongation = tf.cast(
      tf.bitwise.bitwise_or(encoded_r_shifted, encoded_elongation),
      dtype=tf.int64)
  encoded_r = tf.cast(encoded_r, dtype=tf.int64)

  return tf.stack([encoded_r, encoded_intensity, encoded_elongation], axis=-1)
```

이 함수는 모든 형상이 다음 기능을 가질 수 있도록 라이더 포인트 기능을 인코딩 해줍니다.

```
def decode_lidar_features(lidar_point_feature):
  r, intensity, elongation = tf.unstack(lidar_point_feature, axis=-1)

    decoded_r = _decode_range(r)
    intensity = tf.bitwise.bitwise_and(intensity, int(0xFFFF))
```

```
  decoded_intensity =
_decode_intensity(tf.cast(intensity, dtype=tf.uint16))
  elongation = tf.bitwise.bitwise_and(elongation,
int(0xFF))
  decoded_elongation =
_decode_elongation(tf.cast(elongation,
dtype=tf.uint8))

  return tf.stack([decoded_r, decoded_intensity,
decoded_elongation], axis=-1)
```

이 함수는 encode_lidar_features로 인코딩 된 라이더 포인트 기능을 디코딩합니다.

그리고 [N, 3] lidar_point_feature를 리턴해 줍니다.

```
def scatter_nd_with_pool(index,
                                                    value,
                                                    shape,

                                  pool_method=tf.math.unsorted_segment_max):
          if len(shape) != 2:
            raise ValueError('shape must be of size 2')
          height = shape[0]
          width = shape[1]
          # idx: [N]
          index_encoded, idx = tf.unique(index[:, 0] * width + index[:, 1])
          value_pooled = pool_method(value, idx, tf.size(input=index_encoded))
          index_unique = tf.stack(
              [index_encoded // width,
               tf.math.mod(index_encoded, width)], axis=-1)
          shape = [height, width]
          value_shape = _combined_static_and_dynamic_shape(value)
          if len(value_shape) > 1:
            shape = shape + value_shape[1:]

          image = tf.scatter_nd(index_unique, value_pooled, shape)
          return image
```

이 함수는 중복 인덱스가 있는 경우 수를 합하여 줍니다. 그리고 이미지를 리턴해 줍니다.(값이 흩어져 있는 모양의 tensor, 누락 된 픽셀은 0으로 설정해줍니다.)

```
Defcompute_range_image_polar(range_image,
                                                        extrinsic,
                                                        inclination,
```

```
                                      dtype=tf.float32,
                                                                  scope=None):
  _, height, width = _combined_static_and_dynamic_shape(range_image)
  range_image_dtype = range_image.dtype
  range_image = tf.cast(range_image, dtype=dtype)
  extrinsic = tf.cast(extrinsic, dtype=dtype)
  inclination = tf.cast(inclination, dtype=dtype)

  with tf.compat.v1.name_scope(scope, 'ComputeRangeImagePolar',
                               [range_image, extrinsic, inclination]):
    with tf.compat.v1.name_scope('Azimuth'):
      # [B].
      az_correction = tf.atan2(extrinsic[..., 1, 0], extrinsic[..., 0, 0])
      # [W].
      ratios = (tf.cast(tf.range(width, 0, -1), dtype=dtype) - .5) /
tf.cast(
          width, dtype=dtype)
      # [B, W].
      azimuth = (ratios * 2. - 1.) * np.pi - tf.expand_dims(az_correction,
-1)

    # [B, H, W]
    azimuth_tile = tf.tile(azimuth[:, tf.newaxis, :], [1, height, 1])
    # [B, H, W]
    inclination_tile = tf.tile(inclination[:, :, tf.newaxis], [1, 1,
width])
    range_image_polar = tf.stack([azimuth_tile, inclination_tile,
range_image],
                                 axis=-1)
    return tf.cast(range_image_polar, dtype=range_image_dtype)
```

이미지 범위의 극좌표를 계산하는 함수 입니다. range_image는 라이더 범위의 이미지 입니다. extrinstic역시 라이더의 extrinsic입니다 inclination은 범위 이미지의 기울기에 해당하며 0번째 항목은 이미지의 0번째 항목에 포함됩니다.

```
def compute_range_image_cartesian(range_image_polar,
                                  extrinsic,
                                  pixel_pose=None,
                                  frame_pose=None,
                                  dtype=tf.float32,
                                  scope=None):
  range_image_polar_dtype = range_image_polar.dtype
  range_image_polar = tf.cast(range_image_polar, dtype=dtype)
```

```
  extrinsic = tf.cast(extrinsic, dtype=dtype)
  if pixel_pose is not None:
    pixel_pose = tf.cast(pixel_pose, dtype=dtype)
  if frame_pose is not None:
    frame_pose = tf.cast(frame_pose, dtype=dtype)

  with tf.compat.v1.name_scope(
      scope, 'ComputeRangeImageCartesian',
      [range_image_polar, extrinsic, pixel_pose, frame_pose]):
    azimuth, inclination, range_image_range = tf.unstack(
        range_image_polar, axis=-1)

    cos_azimuth = tf.cos(azimuth)
    sin_azimuth = tf.sin(azimuth)
    cos_incl = tf.cos(inclination)
    sin_incl = tf.sin(inclination)

    # [B, H, W].
    x = cos_azimuth * cos_incl * range_image_range
    y = sin_azimuth * cos_incl * range_image_range
    z = sin_incl * range_image_range

    # [B, H, W, 3]
    range_image_points = tf.stack([x, y, z], -1)
    # [B, 3, 3]
    rotation = extrinsic[..., 0:3, 0:3]
    # translation [B, 1, 3]
    translation = tf.expand_dims(tf.expand_dims(extrinsic[..., 0:3, 3], 1), 1)

    # To vehicle frame.
    # [B, H, W, 3]
    range_image_points = tf.einsum('bkr,bijr->bijk', rotation,
                                   range_image_points) + translation
    if pixel_pose is not None:
      # To global frame.
      # [B, H, W, 3, 3]
      pixel_pose_rotation = pixel_pose[..., 0:3, 0:3]
      # [B, H, W, 3]
      pixel_pose_translation = pixel_pose[..., 0:3, 3]
      # [B, H, W, 3]
      range_image_points = tf.einsum(
          'bhwij,bhwj->bhwi', pixel_pose_rotation,
          range_image_points) + pixel_pose_translation
      if frame_pose is None:
        raise ValueError('frame_pose must be set when pixel_pose is set.')
```

```
            # To vehicle frame corresponding to the given frame_pose
            # [B, 4, 4]
            world_to_vehicle = tf.linalg.inv(frame_pose)
            world_to_vehicle_rotation = world_to_vehicle[:, 0:3, 0:3]
            world_to_vehicle_translation = world_to_vehicle[:, 0:3, 3]
            # [B, H, W, 3]
            range_image_points = tf.einsum(
                'bij,bhwj->bhwi', world_to_vehicle_rotation,
                range_image_points) + world_to_vehicle_translation[:, tf.newaxis,
                                                                   tf.newaxis, :]

          range_image_points = tf.cast(
              range_image_points, dtype=range_image_polar_dtype)
          return range_image_points
```

이 함수는 극좌표로부터 범위이미지의 데카르트 좌표를 구합니다. Range_imag_cartesian의 데카르트 좌표를 리턴 해 줍니다.[B, H, W, 3]

```
    def build_camera_depth_image(range_image_cartesian,
                                 extrinsic,
                                 camera_projection,
                                 camera_image_size,
                                 camera_name,
                                 pool_method=tf.math.unsorted_segment_min,
                                 scope=None):
      with tf.compat.v1.name_scope(
          scope, 'BuildCameraDepthImage',
          [range_image_cartesian, extrinsic, camera_projection]):
        # [B, 4, 4]
        vehicle_to_camera = tf.linalg.inv(extrinsic)
        # [B, 3, 3]
        vehicle_to_camera_rotation = vehicle_to_camera[:, 0:3, 0:3]
        # [B, 3]
        vehicle_to_camera_translation = vehicle_to_camera[:, 0:3, 3]
        # [B, H, W, 3]
        range_image_camera = tf.einsum(
            'bij,bhwj->bhwi', vehicle_to_camera_rotation,
            range_image_cartesian) + vehicle_to_camera_translation[:, tf.newaxis,
                                                                   tf.newaxis, :]
        # [B, H, W]
        range_image_camera_norm = tf.norm(tensor=range_image_camera, axis=-1)
        camera_projection_mask_1 = tf.tile(
            tf.equal(camera_projection[..., 0:1], camera_name), [1, 1, 1, 2])
        camera_projection_mask_2 = tf.tile(
            tf.equal(camera_projection[..., 3:4], camera_name), [1, 1, 1, 2])
        camera_projection_selected = tf.ones_like(
```

```
            camera_projection[..., 1:3], dtype=camera_projection.dtype) * -1
        camera_projection_selected = tf.compat.v1.where(camera_projection_mask_2,
                                                        camera_projection[..., 4:6],
                                                        camera_projection_selected)
        # [B, H, W, 2]
        camera_projection_selected = tf.compat.v1.where(camera_projection_mask_1,
                                                        camera_projection[..., 1:3],
                                                        camera_projection_selected)
        # [B, H, W]
        camera_projection_mask = tf.logical_or(camera_projection_mask_1,
                                               camera_projection_mask_2)[..., 0]
```

주어진 카메라 투영에 대한 카메라 깊이 이미지를 만드는 함수입니다. 여기서 말하는 카메라 킾이는 라이더 포인트와 카메라 프레임 원점 사이의 거리입니다. 차량 프레임과 카메라의 직교 좌표에 의해 결정됩니다. Image를 리턴해 주며[B,width,height] 깊이 이미지가 만들어 집니다.

```
        def extract_point_cloud_from_range_image(range_image,
                                                 extrinsic,
                                                 inclination,
                                                 pixel_pose=None,
                                                 frame_pose=None,
                                                 dtype=tf.float32,
                                                 scope=None):
        with tf.compat.v1.name_scope(
              scope, 'ExtractPointCloudFromRangeImage',
              [range_image, extrinsic, inclination, pixel_pose, frame_pose]):
            range_image_polar = compute_range_image_polar(
                range_image, extrinsic, inclination, dtype=dtype)
            range_image_cartesian = compute_range_image_cartesian(
                range_image_polar,
                extrinsic,
                pixel_pose=pixel_pose,
                frame_pose=frame_pose,
                dtype=dtype)
            return range_image_cartesian
```

범위 이미지에서 포인트 클라우드를 추출합니다. 그리고 range_image_cartestian을 리턴 해 줍니다.[B, H, W, 3], {x,y,z}로 차량내부의 차원을 의미합니다.

```
def crop_range_image(range_images, new_width, shift=None, scope=None):
```

```
      shape = _combined_static_and_dynamic_shape(range_images)
      batch = shape[0]
      width = shape[2]
      if width == new_width:
        return range_images
      if new_width < 1:
        raise ValueError('new_width must be positive.')
```

```
    if width is not None and new_width >= width:
      raise ValueError('new_width {} should be < the old width {}.'.format(
          new_width, width))

    if shift is None:
      shift = [0] * batch

    diff = width - new_width
    left = [diff // 2 + i for i in shift]
    right = [i + new_width for i in left]

    for l, r in zip(left, right):
      if l < 0 or r > width:
        raise ValueError(
            'shift {} is invalid given new_width {} and width {}.'.format(
                shift, new_width, width))

    range_image_crops = []
    with tf.compat.v1.name_scope(scope, 'CropRangeImage', [range_images]):
      for i in range(batch):
        range_image_crop = range_images[i, :, left[i]:right[i], ...]
        range_image_crops.append(range_image_crop)
      return tf.stack(range_image_crops, axis=0)
```

범위 이미지의 범위를 잘라 너비를 줄입니다. 요구 사항으로 new_width가 기존 너비보다 작아야 합니다. 그리고 range_image_crops를 리턴 해 줍니다. 범위 의미지의 형태는 [B, H, W,..] 입니다.

```
def compute_inclination(inclination_range, height, scope=None):
```

```
      with tf.compat.v1.name_scope(scope, 'ComputeInclination',
                                   [inclination_range]):
        diff = inclination_range[..., 1] - inclination_range[..., 0]
        inclination = (
            (.5 + tf.cast(tf.range(0, height), dtype=inclination_range.dtype)) /
            tf.cast(height, dtype=inclination_range.dtype) *
            tf.expand_dims(diff, axis=-1) + inclination_range[..., 0:1])
```

```
            return inclination
```

주어진 범위와 높이를 기준으로 균일 한 inclination-range를 계산합니다.inclination_range는 텐서 내부 차원의[최소 inclination, 최대 inclination]을 의미합니다. 그리고 계산된 linclination을 리턴해 줍니다

#from waymo_open_dataset.utils import transform_utils

Geometry를 반환하고 관리하는 유틸리티 입니다.

```
        def get_yaw_rotation(yaw, name=None):
          with tf.compat.v1.name_scope(name, 'GetYawRotation', [yaw]):
            cos_yaw = tf.cos(yaw)
            sin_yaw = tf.sin(yaw)
            ones = tf.ones_like(yaw)
            zeros = tf.zeros_like(yaw)

            return tf.stack([
                tf.stack([cos_yaw, -1.0 * sin_yaw, zeros], axis=-1),
                tf.stack([sin_yaw, cos_yaw, zeros], axis=-1),
                tf.stack([zeros, zeros, ones], axis=-1),
            ],
                            axis=-2)
```

Rotation matrix를 가져 옵니다. Yaw는 x-rotation in radians의 형태입니다. 이 tensor는 비어있는 것을 제외한 다른 shape으로 바뀔 수 있습니다. 입력된 데이터 타입과 같은 형태로 tensor이 리턴이 됩니다. [input_shape, 3, 3]

```
def get_yaw_rotation_2d(yaw):
```

```
          with tf.name_scope('GetYawRotation2D'):
            cos_yaw = tf.cos(yaw)
            sin_yaw = tf.sin(yaw)

            return tf.stack([
                tf.stack([cos_yaw, -1.0 * sin_yaw], axis=-1),
                tf.stack([sin_yaw, cos_yaw], axis=-1),
            ],
                            axis=-2)
```

2차원에 대해서 yaw가 주어진 rotation matrix를 가지고 옵니다 그리고 입력 데이터와 같은 형식으로 yaw가 리턴이 됩니다.

```
def get_rotation_matrix(roll, pitch, yaw, name=None):
```

```
            cos_roll = tf.cos(roll)
            sin_roll = tf.sin(roll)
            cos_yaw = tf.cos(yaw)
            sin_yaw = tf.sin(yaw)
            cos_pitch = tf.cos(pitch)
            sin_pitch = tf.sin(pitch)

            ones = tf.ones_like(yaw)
            zeros = tf.zeros_like(yaw)

            r_roll = tf.stack([
                tf.stack([ones, zeros, zeros], axis=-1),
                tf.stack([zeros, cos_roll, -1.0 * sin_roll], axis=-1),
                tf.stack([zeros, sin_roll, cos_roll], axis=-1),
            ],
                            axis=-2)
            r_pitch = tf.stack([
                tf.stack([cos_pitch, zeros, sin_pitch], axis=-1),
                tf.stack([zeros, ones, zeros], axis=-1),
                tf.stack([-1.0 * sin_pitch, zeros, cos_pitch], axis=-1),
            ],
                             axis=-2)
            r_yaw = tf.stack([
                tf.stack([cos_yaw, -1.0 * sin_yaw, zeros], axis=-1),
                tf.stack([sin_yaw, cos_yaw, zeros], axis=-1),
                tf.stack([zeros, zeros, ones], axis=-1),
            ],
                            axis=-2)

            return tf.matmul(r_yaw, tf.matmul(r_pitch, r_roll))
```

Roll, pitch, yaw에 대한 rotation matrix를 가지고옵니다. Roll-pitch-yaw 는 z-y-x의 고유 회전입니다.

라디안 단위의 x 회전 라디안 단위의 y회전 라디안 단위인 z 회전을 할 수 있습니다. 입력 데이터 유형과 동일하고 그 형태는 [input_shape_of_yaw, 3, 3]입니다.

```
def get_transform(rotation, translation):
```

```
            # [..., N, 1]
            translation_n_1 = translation[..., tf.newaxis]
            # [..., N, N+1]
            transform = tf.concat([rotation, translation_n_1], axis=-1)
            # [..., N]
            last_row = tf.zeros_like(translation)
            # [..., N+1]
            last_row = tf.concat([last_row, tf.ones_like(last_row[..., 0:1])], axis=-1)
            # [..., N+1, N+1]
```

```
            transform = tf.concat([transform, last_row[..., tf.newaxis, :]], axis=-2)
            return transform
```

N*N 회전과 N*1 변환을 (N+1) *(N+1) 변환으로 결합합니다. 그리고 이를 리턴해 줍니다.

#from waymo_open_dataset.utils import frame_utils

프레임 proto를 관리하는 유티릴티 입니다.

```
def parse_range_image_and_camera_projection(frame):
```

```
"
          range_images = {}
          camera_projections = {}
          range_image_top_pose = None
          for laser in frame.lasers:
            if len(laser.ri_return1.range_image_compressed) > 0:  # pylint: disable=g-
        explicit-length-test
              range_image_str_tensor = tf.io.decode_compressed(
                  laser.ri_return1.range_image_compressed, 'ZLIB')
              ri = dataset_pb2.MatrixFloat()
              ri.ParseFromString(bytearray(range_image_str_tensor.numpy()))
              range_images[laser.name] = [ri]

              if laser.name == dataset_pb2.LaserName.TOP:
                range_image_top_pose_str_tensor = tf.io.decode_compressed(
                    laser.ri_return1.range_image_pose_compressed, 'ZLIB')
                range_image_top_pose = dataset_pb2.MatrixFloat()
                range_image_top_pose.ParseFromString(
                    bytearray(range_image_top_pose_str_tensor.numpy()))

              camera_projection_str_tensor = tf.io.decode_compressed(
                  laser.ri_return1.camera_projection_compressed, 'ZLIB')
              cp = dataset_pb2.MatrixInt32()
              cp.ParseFromString(bytearray(camera_projection_str_tensor.numpy()))
              camera_projections[laser.name] = [cp]
            if len(laser.ri_return2.range_image_compressed) > 0:  # pylint: disable=g-
        explicit-length-test
              range_image_str_tensor = tf.io.decode_compressed(
```

```
                laser.ri_return2.range_image_compressed, 'ZLIB')
            ri = dataset_pb2.MatrixFloat()
            ri.ParseFromString(bytearray(range_image_str_tensor.numpy()))
            range_images[laser.name].append(ri)

            camera_projection_str_tensor = tf.io.decode_compressed(
                laser.ri_return2.camera_projection_compressed, 'ZLIB')
            cp = dataset_pb2.MatrixInt32()
            cp.ParseFromString(bytearray(camera_projection_str_tensor.numpy()))
            camera_projections[laser.name].append(cp)
        return range_images, camera_projections, range_image_top_pose
```

위 함수는 근거리 범위 이미지와 프레임이 주어진 카메라를 투영합니다. Frame은 데이터세트의 frame proto입니다.

```
e
    def convert_range_image_to_cartesian(frame,
                                         range_images,
                                         range_image_top_pose,
                                         ri_index=0,
                                         keep_polar_features=False):
"
      cartesian_range_images = {}
      frame_pose = tf.convert_to_tensor(
          value=np.reshape(np.array(frame.pose.transform), [4, 4]))

      # [H, W, 6]
      range_image_top_pose_tensor = tf.reshape(
          tf.convert_to_tensor(value=range_image_top_pose.data),
          range_image_top_pose.shape.dims)
      # [H, W, 3, 3]
      range_image_top_pose_tensor_rotation = transform_utils.get_rotation_matrix(
          range_image_top_pose_tensor[..., 0], range_image_top_pose_tensor[..., 1],
          range_image_top_pose_tensor[..., 2])
      range_image_top_pose_tensor_translation = range_image_top_pose_tensor[..., 3:]
      range_image_top_pose_tensor = transform_utils.get_transform(
          range_image_top_pose_tensor_rotation,
          range_image_top_pose_tensor_translation)

      for c in frame.context.laser_calibrations:
        range_image = range_images[c.name][ri_index]
        if len(c.beam_inclinations) == 0:  # pylint: disable=g-explicit-length-test
          beam_inclinations = range_image_utils.compute_inclination(
              tf.constant([c.beam_inclination_min, c.beam_inclination_max]),
```

```
            height=range_image.shape.dims[0])
        else:
          beam_inclinations = tf.constant(c.beam_inclinations)

        beam_inclinations = tf.reverse(beam_inclinations, axis=[-1])
        extrinsic = np.reshape(np.array(c.extrinsic.transform), [4, 4])

        range_image_tensor = tf.reshape(
            tf.convert_to_tensor(value=range_image.data), range_image.shape.dims)
        pixel_pose_local = None
        frame_pose_local = None
        if c.name == dataset_pb2.LaserName.TOP:
          pixel_pose_local = range_image_top_pose_tensor
          pixel_pose_local = tf.expand_dims(pixel_pose_local, axis=0)
          frame_pose_local = tf.expand_dims(frame_pose, axis=0)
        range_image_cartesian = range_image_utils.extract_point_cloud_from_range_image(
            tf.expand_dims(range_image_tensor[..., 0], axis=0),
            tf.expand_dims(extrinsic, axis=0),
            tf.expand_dims(tf.convert_to_tensor(value=beam_inclinations), axis=0),
            pixel_pose=pixel_pose_local,
            frame_pose=frame_pose_local)

        range_image_cartesian = tf.squeeze(range_image_cartesian, axis=0)

        if keep_polar_features:
          # If we want to keep the polar coordinate features of range, intensity,
          # and elongation, concatenate them to be the initial dimensions of the
          # returned Cartesian range image
          range_image_cartesian = tf.concat(
              [range_image_tensor[..., 0:3], range_image_cartesian], axis=-1)

        cartesian_range_images[c.name] = range_image_cartesian

      return cartesian_range_images
```

극좌표에서 데카르트 좌표로 범위 이미지를 반환합니다.

```
    def convert_range_image_to_point_cloud(frame,
                                           range_images,
                                           camera_projections,
                                           range_image_top_pose,
                                           ri_index=0,
                                           keep_polar_features=False):
      calibrations = sorted(frame.context.laser_calibrations, key=lambda c: c.name)
      points = []
      cp_points = []
```

```
        cartesian_range_images = convert_range_image_to_cartesian(
            frame, range_images, range_image_top_pose, ri_index, keep_polar_features)

        for c in calibrations:
          range_image = range_images[c.name][ri_index]
          range_image_tensor = tf.reshape(
              tf.convert_to_tensor(value=range_image.data), range_image.shape.dims)
          range_image_mask = range_image_tensor[..., 0] > 0

          range_image_cartesian = cartesian_range_images[c.name]
          points_tensor = tf.gather_nd(range_image_cartesian,
                                       tf.compat.v1.where(range_image_mask))

          cp = camera_projections[c.name][ri_index]
          cp_tensor = tf.reshape(tf.convert_to_tensor(value=cp.data), cp.shape.dims)
          cp_points_tensor = tf.gather_nd(cp_tensor,
                                          tf.compat.v1.where(range_image_mask))
          points.append(points_tensor.numpy())
          cp_points.append(cp_points_tensor.numpy())

        return points, cp_points
```

범위 이미지를 포인트 클라우드로 변환합니다

Waymo 듀토리얼 코드의 Dockerfile

```
FROM tensorflow/tensorflow:latest-py3

RUN apt-get update && apt-get install -y \
  git build-essential wget vim findutils curl \
  pkg-config zip g++ zlib1g-dev unzip python3 python3-pip

RUN apt-get install -y wget
RUN wget https://github.com/bazelbuild/bazel/releases/download/0.28.0/bazel-0.28.0-
installer-linux-x86_64.sh
RUN chmod +x bazel-0.28.0-installer-linux-x86_64.sh
RUN bash ./bazel-0.28.0-installer-linux-x86_64.sh

RUN pip3 install jupyter matplotlib jupyter_http_over_ws &&\
  jupyter serverextension enable --py jupyter_http_over_ws

RUN git clone https://github.com/waymo-research/waymo-open-dataset.git waymo-od
WORKDIR /waymo-od

RUN bash ./configure.sh && \
    bash bazel query ... | xargs bazel build -c opt && \
    bash bazel query 'kind(".*_test rule", ...)' | xargs bazel test -c opt ...

EXPOSE 8888
RUN python3 -m ipykernel.kernelspec

CMD ["bash", "-c", "source /etc/bash.bashrc && bazel run -c opt
//tutorial:jupyter_kernel"]
```

최신버전 tensorflow의 도커이미지를 받아서 실행합니다.

FROM tensorflow/tensorflow:latest-py3

wget, git을 설치해 줍니다

RUN apt-get update && apt-get install -y ₩
 git build-essential wget vim findutils curl ₩
 pkg-config zip g++ zlib1g-dev unzip python3 python3-pip
RUN apt-get install -y wget
RUN wget https://github.com/bazelbuild/bazel/releases/download/0.28.0/bazel-0.28.0-installer-linux-x86_64.sh
RUN chmod +x bazel-0.28.0-installer-linux-x86_64.sh
RUN bash ./bazel-0.28.0-installer-linux-x86_64.sh
#jupyter matplotlib를 설치해 줍니다.

RUN pip3 install jupyter matplotlib jupyter_http_over_ws &&₩
   jupyter serverextension enable --py jupyter_http_over_ws

#소스파일을 받아옵니다..

RUN git clone https://github.com/waymo-research/waymo-open-dataset.git waymo-od
WORKDIR /waymo-od

#오픈할 포트를 지정해 줍니다.

EXPOSE 8888
RUN python3 -m ipykernel.kernelspec

#컨테이너에서 실행될 명령어를 지정해줍니다.

CMD ["bash", "-c", "source /etc/bash.bashrc && bazel run -c opt //tutorial:jupyter_kernel"]

느낀점

이번에 과제를 하게 되면서 오픈 소스와 데이터세트, 그리고 docker 파일을 처음 접하게 되었다.

오픈소스와 데이터세트를 이용하기 위해 영어공부를 해야 할 것 같다는 생각이 크게 들었다.

데이터들을 서로 공유하고 개발해 나간다는 모습이 정말 멋있는 것 같다.

코드들의 내용이나 아직 개념적으로 이해하기 힘든 부분이 많았다. 많이 부족한 것 같았다. 열심히 공부를 해야 할 것 같다.

이번기회에 데이터 세트, 오픈소스를 처음 접해 본 시간이 되어서 뿌듯했다.