차량지능기초

과제1

20163331

자동차공학과 전진우

1) 자율주행 인지에 관련된 3종 이상의 공개 Data Set 조사, 정리

(1) Kitti Object Detection Datasets

이 데이터 셋은 "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite" 라는 2012년 CVPR에 의해 생성되었다. 이 커널은 자율주행 용으로 다양한 데이터 세트의 객체 감지 파일이 포함되어 있으며 경계 상자 라벨이 있는 이미지 세트가 있다.

데이터 셋에는 KITTI 2D 객체 감지 및 방향 추정 데이터를 담고 있다. 각 작업에 대해 7518개의 테스트 이미지 또는 포인트 클라우드를 제공하며 8가지의 라벨이 존재한다. 훈련 및 테스트 데이터는 Train, Test 파일 안에 존재하고 image, label 파일 내부에 카메라 이미지와 라벨에 대한 png 파일이 존재한다. 데이터의 세부요소는 아래와 같다.

```
#Values    Name      Description
----------------------------------------------------------------------------
   1    type         Describes the type of object: 'Car', 'Van', 'Truck',
                     'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram',
                     'Misc' or 'DontCare'
   1    truncated    Float from 0 (non-truncated) to 1 (truncated), where
                     truncated refers to the object leaving image boundaries
   1    occluded     Integer (0,1,2,3) indicating occlusion state:
                     0 = fully visible, 1 = partly occluded
                     2 = largely occluded, 3 = unknown
   1    alpha        Observation angle of object, ranging [-pi..pi]
   4    bbox         2D bounding box of object in the image (0-based index):
                     contains left, top, right, bottom pixel coordinates
   3    dimensions   3D object dimensions: height, width, length (in meters)
   3    location     3D object location x,y,z in camera coordinates (in meters)
   1    rotation_y   Rotation ry around Y-axis in camera coordinates [-pi..pi]
   1    score        Only for results: Float, indicating confidence in
                     detection, needed for p/r curves, higher is better.
```

- 2D Object Detection Benchmark

2D 물체 탐지 task의 목적은 'Car', 'Pedestrian', 'Cyclist' 클래스로 물체 탐지기를 훈련시키는 것이다. 물체 탐지기는 2D 0기반의 경계 상자를 이미지 내에 표시하고 또한 탐지 스코어를 통해 평가할 수 있다. 각 이미지 당 하나의 텍스트 파일을 가지고 있고 각 평가에서는 25 pixel 이상의 물체만 대상으로 한다.

- Object Orientation Estimation Benchmark

앞의 물체 탐지와 비슷하지만, 추가적으로 각 탐지마다 가장 가능성이 높은 객체 관찰 각도를 볼 수 있다. 각 탐지에 대해 알고리즘 추정 성능과 탐지 성능 및 방향을 고려할 수 있다.

EX) train image and label data

```
Car 0.88 3 -0.69 0.00 192.37 402.31 374.00 1.60 1.57 3.23 -2.70 1.74 3.68 -1.29
Car 0.00 1 2.04 334.85 178.94 624.50 372.04 1.57 1.50 3.68 -1.17 1.65 7.86 1.90
Car 0.34 3 -1.84 937.29 197.39 1241.00 374.00 1.39 1.44 3.08 3.81 1.64 6.15 -1.31
Car 0.00 1 -1.33 597.59 176.18 720.90 261.14 1.47 1.60 3.66 1.07 1.55 14.44 -1.25
Car 0.00 0 1.74 741.18 168.83 792.25 208.43 1.70 1.63 4.08 7.24 1.55 33.20 1.95
Car 0.00 0 -1.65 884.52 178.31 956.41 240.18 1.59 1.59 2.47 8.48 1.75 19.96 -1.25
DontCare -1 -1 -10 800.38 163.67 825.45 184.07 -1 -1 -1 -1000 -1000 -1000 -10
DontCare -1 -1 -10 859.58 172.34 886.26 194.51 -1 -1 -1 -1000 -1000 -1000 -10
DontCare -1 -1 -10 801.81 163.96 825.20 183.59 -1 -1 -1 -1000 -1000 -1000 -10
DontCare -1 -1 -10 826.87 162.28 845.84 178.86 -1 -1 -1 -1000 -1000 -1000 -10

Pedestrian 0.00 1 0.30 883.68 144.15 937.35 259.01 1.90 0.42 1.04 5.06 1.43 12.42 0.68
Pedestrian 0.00 2 0.29 873.70 152.10 933.44 256.07 1.87 0.50 0.90 5.42 1.50 13.43 0.67
Car 0.00 0 1.74 444.29 171.04 504.95 225.82 1.86 1.57 3.83 -4.95 1.83 26.64 1.55
Pedestrian 0.00 0 -0.39 649.28 168.10 664.61 206.40 1.78 0.53 0.95 2.20 1.57 34.08 -0.33
Car 0.98 0 2.42 0.00 217.12 85.92 374.00 1.50 1.46 3.70 -5.12 1.85 4.13 1.56
Pedestrian 0.00 0 2.02 240.35 190.31 268.02 261.61 1.54 0.57 0.41 -7.92 1.94 15.95 1.57
DontCare -1 -1 -10 0.00 226.06 88.58 373.00 -1 -1 -1 -1000 -1000 -1000 -10
DontCare -1 -1 -10 567.39 173.95 574.86 190.60 -1 -1 -1 -1000 -1000 -1000 -10
DontCare -1 -1 -10 727.58 165.75 737.08 192.75 -1 -1 -1 -1000 -1000 -1000 -10
```

출처: https://www.kaggle.com/twaldo/kitti-object-detection

(2) Ford Campus Vision and Lidar Data Set

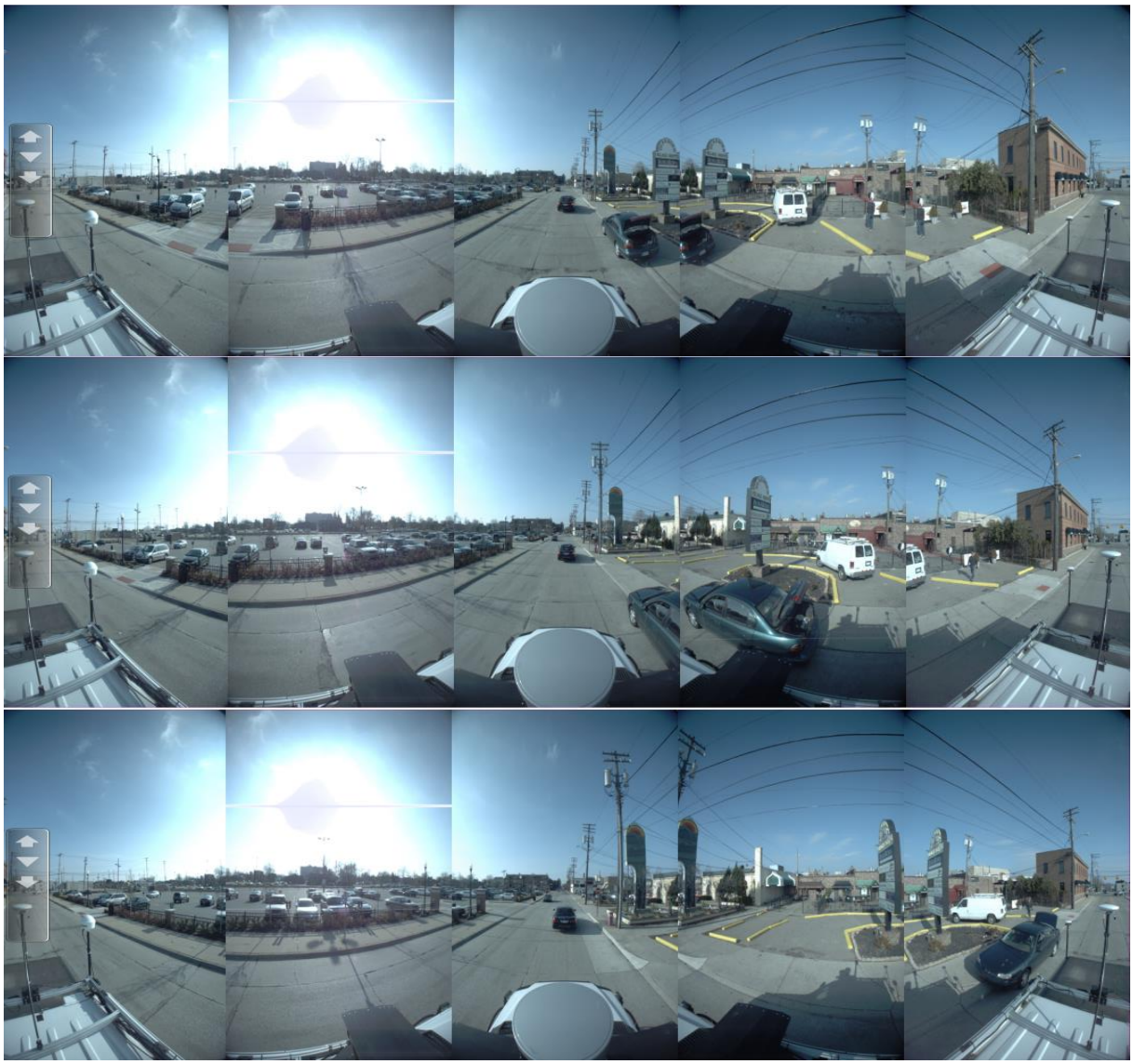
이 데이터 셋은 Ford F-250 픽업 트럭이 자율주행 기반으로 수집한 데이터 셋이다. 해당 차량에는 전문가용 및 소비자용 관성 측정 장치(IMU), 벨로다인 3D 라이다 스캐너, 두 개의 Push-broom 전방 Riegl 라이다, 전 방향 카메라(Point Grey Ladybug3)가 장착되어 있다. 2009년 11월부터 12월까지 Ford 연구 캠퍼스와 미시간 주 Dearborn 지역을 주행하는 동안 장착된 센서로부터 등록 데이터를 취득하였다.

데이터 셋에는 크게 LCM, IMAGES, SCANS, VELODYNE 폴더가 있으며 내부 파일은 Timestamp.log, Pose-Applanix.log, Pose-Mtig.log, Gps.log, PARAM.mat 등이 있다. LCM 폴더에는 각 시험에 해당하는 LCM 로그 파일이 포함되어 있다. 각 LCM 로그 파일에는 벨로 다인 레이저 스캐너의 원시 3D 포인트 클라우드, 라이더 데이터 및 네비게이션 센서의 데이터 등을 제공한다. 그 내부에 00초 이후로 마이크로 초 단위로 시간을 측정한 데이터 TIMEstamp.log, time stamp와 함께 차량의 6가지 자유도 포즈를 포함된 Pose-Applanix.log 등이 있다. IMAGES 폴더에는 전 방향 카메라에서 찍은 왜곡되지 않은 이미지 파일이 있다. 각 내부 폴더는 전 방향 카메라 시스템의 개별 카메라에 해당하는 폴더가 존재한다. 전 방향 카메라의 고유 및 외부 매개 변수를 포함하는 PARAM.mat, 그 내부는 카메라 내부 매개 변수와 행렬, 이동 벡터 등 시스템이 존재한다. SCANS 폴ㄹ더에는 벨로다인 레이저 스캐너와 3D 스캔 데이터가 포함되어 있다. Applanix에서 제공하는 차량의 6자

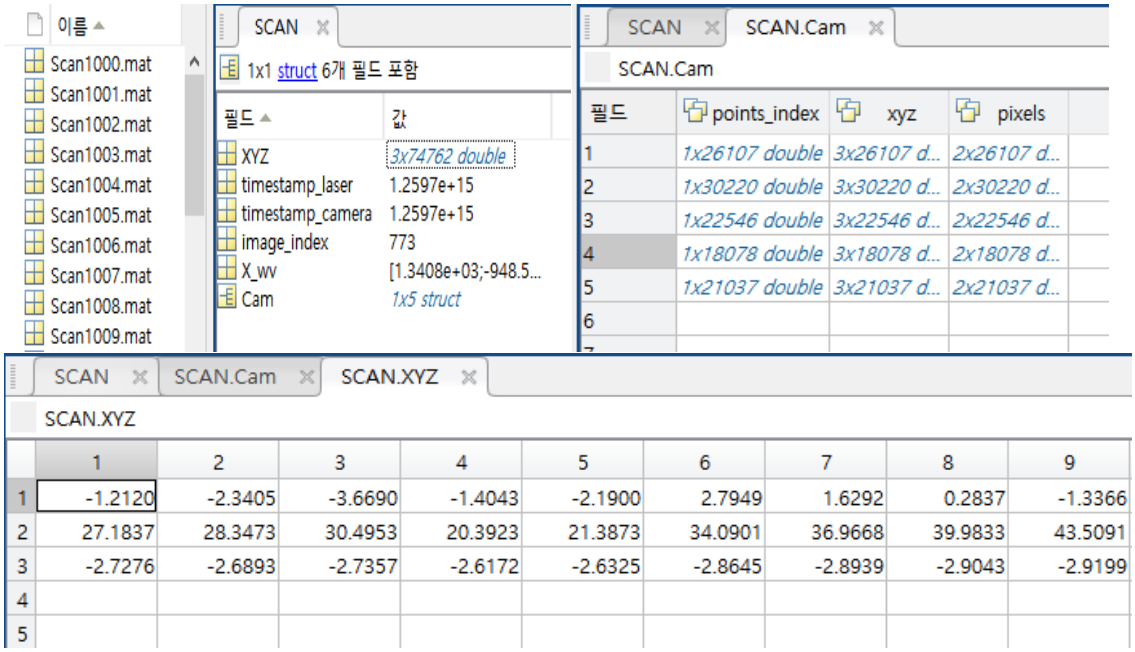
유도, 각 내부에는 Scan.XYZ, SCAN.timestamp, SCAN.cam 등 다양한 스캔 데이터가 존재한다.

해당 데이터 셋을 통해서 카메라 이미지에 3D 포인트 클라우드를 투영할 수 있다. 자율주행차량은 주변 환경을 이해할 수 있고 자가 위치 인식 또는 경로를 판단할 수 있다.

EX) IMAGES and SCAN data



- Scan data in matlab



SCAN

1x1 struct 6개 필드 포함

필드	값
XYZ	3x74762 double
timestamp_laser	1.2597e+15
timestamp_camera	1.2597e+15
image_index	773
X_wv	[1.3408e+03;-948.5...
Cam	1x5 struct

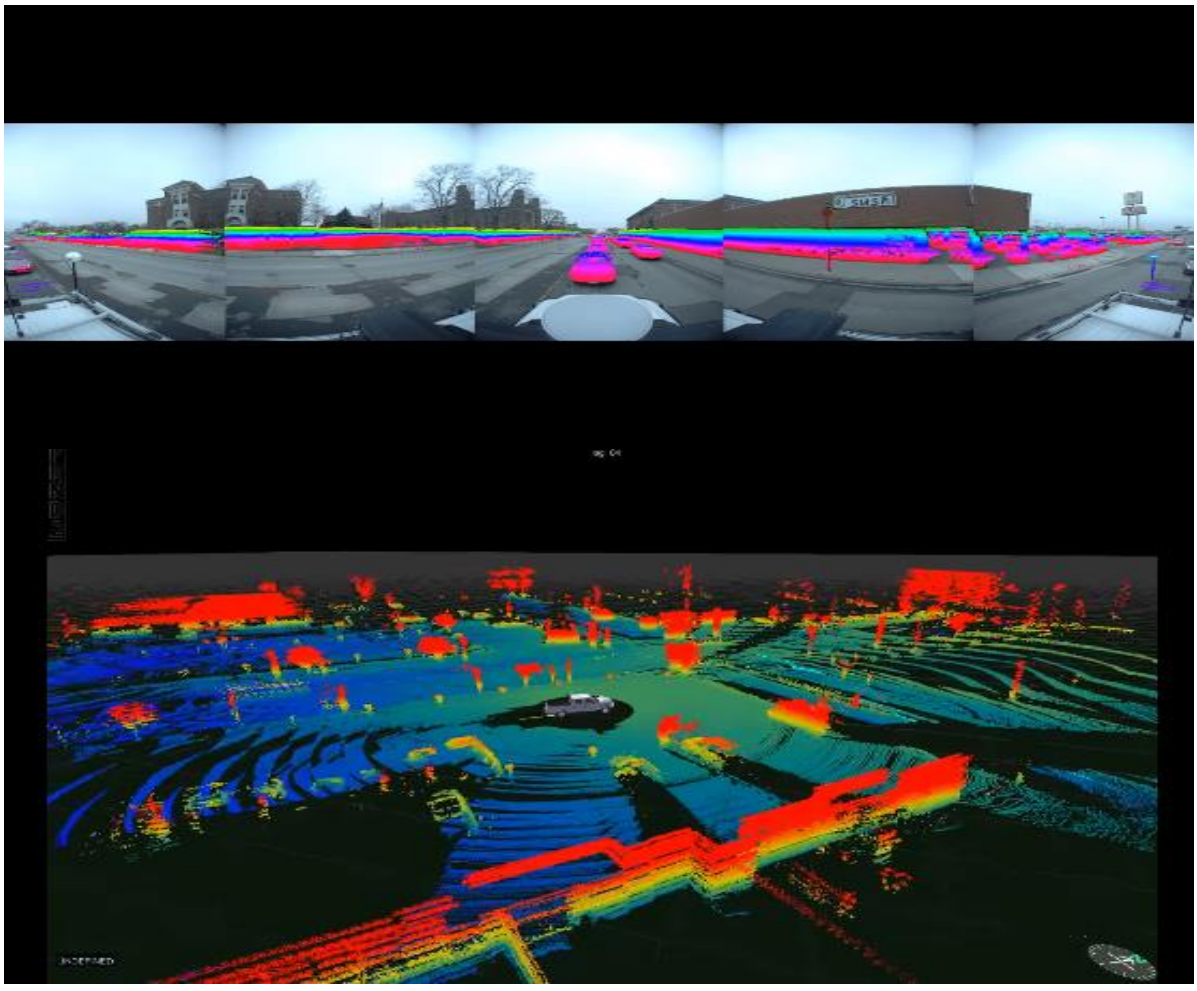
SCAN.Cam

필드	points_index	xyz	pixels
1	1x26107 double	3x26107 d...	2x26107 d...
2	1x30220 double	3x30220 d...	2x30220 d...
3	1x22546 double	3x22546 d...	2x22546 d...
4	1x18078 double	3x18078 d...	2x18078 d...
5	1x21037 double	3x21037 d...	2x21037 d...
6			

SCAN.XYZ

	1	2	3	4	5	6	7	8	9
1	-1.2120	-2.3405	-3.6690	-1.4043	-2.1900	2.7949	1.6292	0.2837	-1.3366
2	27.1837	28.3473	30.4953	20.3923	21.3873	34.0901	36.9668	39.9833	43.5091
3	-2.7276	-2.6893	-2.7357	-2.6172	-2.6325	-2.8645	-2.8939	-2.9043	-2.9199
4									
5									

- RESULT IMAGE



출처: http://robots.engin.umich.edu/SoftwareData/Ford

(3) Daimler Urban Segmentation Data Set

이 데이터 셋은 T. Scharwächter, M. Enzweiler, S. Roth, and U. Franke. "Efficient Multi-Cue Scene Segmentation" 논문에서 출간되었다. 해당 데이터 셋은 도시 교통에 기록된 비디오 시퀀스로 구성된다. 데이터 세트는 1024*440 프레임의 해상도를 가진 5000개의 stereo 이미지로 구성되었으며 각 클래스는 ground, building, vehicle, pedestrian, sky로 구성된다. 데이터 셋을 통해 각 클래스를 segmentation하여 이미지 영상을 처리하여 자율주행 기술에 기여할 수 있다.

데이터 셋은 5개의 비디오 시퀀스에서 500개의 레이블이 지정된 이미지로 구성된다. 세 가지 시퀀스가 훈련 데이터(train_1, train_2, train_3)로 구성되고 시험 데이터(test_1, test_2)가 존재한다. 모든 해상도는 1024*440 픽셀이다. 모든 영상은 PGM grayscale 형식을 따르며 이 이미지는 16bit binary값이며 0~4095 사이의 값(12bit image data)을 가진다. 라벨 데이터는 8비트 ASCII값이 객체 라벨 ID를 나타낸다. 자세한 클래스는 아래와 같다.

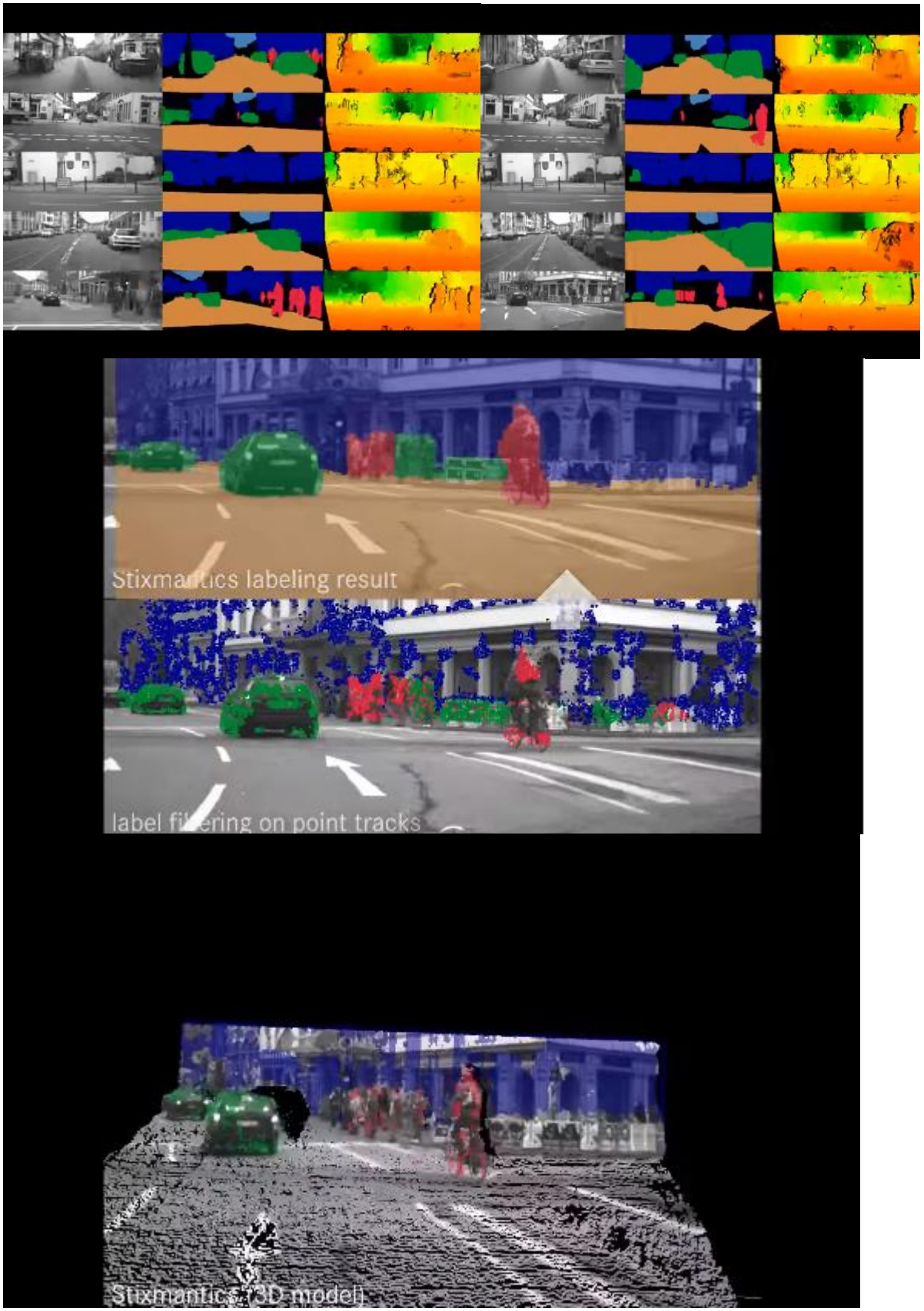
ID	Object class
0	Ground
1	Vehicle
2	Pedestrian
3	Crash barrier
4	Sky
5	Bicycle
6	Reflector post
7	Delineator post
8	Building
9	Curb
10	Lane marking
12	Cyclist
....	
255	Unknown / Background

특히 이 데이터 셋에서는 MATLAB을 활용하여 데이터 셋을 읽고 시각화하며 작업할 수 있는 기본 코드를 제공한다. MATLAB에서 Image Processing Toolbox를 이용하여 모든 이미지를 비디오 시퀀스로 표현할 수 있다. 아래는 그 예시이다.

+2 | get_stereo_cam_params.m | image_to_world_full.m | read_colormap.m | read_dataset_entity.m

```
1   function [ xImg, yImg, zImg ] = image_to_world_full(dispImage, seqDir)
2
3 –     xImg = zeros(size(dispImage));
4 –     yImg = zeros(size(dispImage));
5 –     zImg = zeros(size(dispImage));
6
7 –     for v=1:size(dispImage,1)
8 –         for u=1:size(dispImage,2)
9
10 –            [ x, y, z ] = image_to_world(u, v, dispImage(v,u), seqDir);
```

- RESULT IMAGE



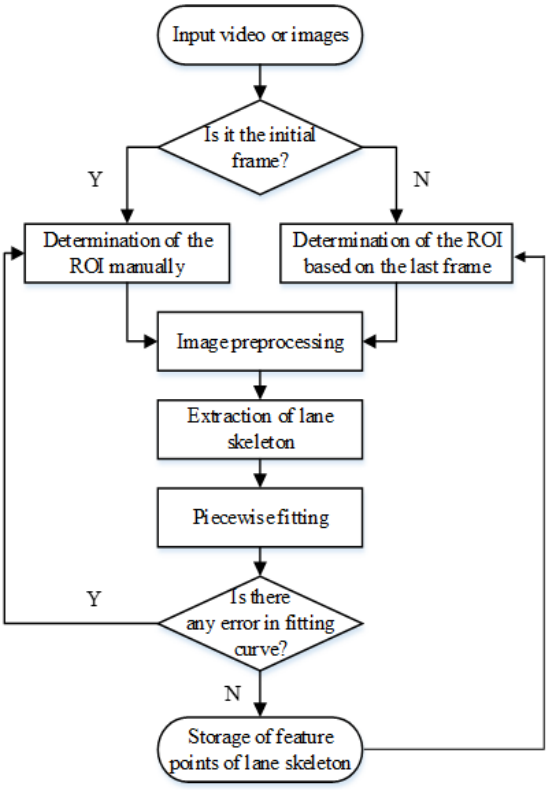
출처: http://www.6d-vision.com/scene-labeling/

(4) Jiqing Expressway Dataset

이 데이터 셋은 다중 차선 탐지 알고리즘을 테스트하고 평가하는데 사용할 수 있는 데이터이다. 중국 지칭 (제남-청도) 고속도로의 일부 구간에서 교통 기록기에 의해 수집되었으며 40개의 비디오 클립이 있다. 각 비디오는 약 3분이며 30fps 프레임 속도, 해상도는 1920*1080이다. 조명과 도로 조건이 다른 총 210,610개의 도로 이미지가 있다.

각 프레임의 line에 라벨을 지정하고 라벨 결과를 txt파일로 저장한다. 서로 다른 레인의 주요 특징점 좌표(x,y)는 서로 다른 행에 저장되며 레인 "x"는 레인 일련 번호를 나타낸다.

각 데이터 셋은 다음의 단계를 통해 차선을 추출한다.



1. 비디오 또는 이미지 파일을 불러와 초기 프레임에 대한 ROI 초기 위치를 표시한다.

2. ROI 이미지에 Gauss smoothing을 적용하여 이미지의 차선 가장자리 회색 값과 중앙 회색 값의 차이를 발견한다.

3. 차선 골격을 추출하기 위해 이미지에 non-maximum suppression을 수행한다.

4. 차선 골격의 픽셀에 대해 segment least square을 적용한다.

5. 결과값과 테스트 결과가 일치하면 segment로 sampling하여 저장하고 다음 프레임의 ROI결정 기준으로 사용한다.

EX)

```
lane0: (956,610)(944,612)(932,614)(920,616)(908,618)(897,620)(885,622)(874,624)(863,626)(853,628)(842,630)(832,632)
(821,634)(811,636)(801,638)(791,640)(781,642)(771,644)(710,656)(700,658)(690,660)(612,675)(535,690)(457,705)
(380,720)(302,735)(225,750)(147,765)(122,770)(70,780)(18,790)(-32,800)(-84,810)(-136,820)(-187,830)(-239,840)(-
286,849)
lane1: (1002,611)(997,613)(992,615)(987,617)(983,619)(978,621)(974,623)(970,625)(966,627)(962,629)(958,631)
(955,633)(951,635)(948,637)(945,639)(941,641)(938,643)(935,645)(917,657)(914,659)(911,661)(889,676)(867,691)
(844,706)(822,721)(800,736)(778,751)(756,766)(748,771)(733,781)(719,791)(704,801)(689,811)(674,821)(659,831)
(645,841)(633,849)
lane2: (1047,609)(1047,611)(1048,613)(1049,615)(1050,617)(1052,619)(1053,621)(1055,623)(1057,625)(1059,627)
(1061,629)(1064,631)(1066,633)(1069,635)(1072,637)(1074,639)(1077,641)(1080,643)(1083,645)(1102,657)(1105,659)
(1129,674)(1152,689)(1176,704)(1200,719)(1223,734)(1247,749)(1270,764)(1278,769)(1294,779)(1310,789)(1326,799)
(1341,809)(1357,819)(1373,829)(1389,839)(1404,849)(1404,849)
lane3: (1084,608)(1088,610)(1094,612)(1100,614)(1106,616)(1113,618)(1120,620)(1127,622)(1135,624)(1143,626)
(1151,628)(1160,630)(1168,632)(1177,634)(1186,636)(1194,638)(1203,640)(1212,642)(1220,644)(1273,656)(1282,658)
(1350,673)(1418,688)(1486,703)(1554,718)(1622,733)(1690,748)(1758,763)(1781,768)(1826,778)(1871,788)(1917,798)
(1962,808)(2007,818)(2053,828)(2098,838)(2143,848)(2148,849)
```

출처: https://github.com/vonsj0210/Multi-Lane-Detection-Dataset-with-Ground-Truth

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

(1) vehicle_detection

https://github.com/MehdiSv/VehicleDetection

(2) road_lane_line_detection

https://github.com/georgesung/road_lane_line_detection

3) 2)에서 정리한 코드 중 하나를 실행시켜 결과 확인

위 두 문제는 google의 colaboratory를 이용하여 조사, 정리 및 실행하였습니다.

Google.Colaboratory의 구현 환경은 간단한 명령어를 통해 표시하였습니다.

2)에서 정리한 코드와 실행시킨 코드를 본인의 github의 repository에 공개하였습니다. Road_lane_line_Detection 모델을 돌려보아 나온 결과물을 white.mp4, yellow.mp4 파일로 저장하였습니다. 또한 과제 전체 pdf파일을 첨부하였습니다.

본인 git 링크: https://github.com/jinu9688/VI_lecture_homework-1

```
# Colab OS 종류
!cat /etc/issue.net

# Colab CPU 사양
!head /proc/cpuinfo

# 메모리 사양
!head -n 3 /proc/meminfo

# 디스크 사양
!df -h
```

```
Ubuntu 18.04.5 LTS
processor	: 0
vendor_id	: GenuineIntel
cpu family	: 6
model		: 63
model name	: Intel(R) Xeon(R) CPU @ 2.30GHz
stepping	: 0
microcode	: 0x1
cpu MHz		: 2299.998
cache size	: 46080 KB
physical id	: 0
MemTotal:       13333568 kB
MemFree:        10722148 kB
MemAvailable:   12476332 kB
Filesystem      Size  Used Avail Use% Mounted on
overlay         108G   39G   70G  36% /
tmpfs            64M     0   64M   0% /dev
tmpfs           6.4G     0  6.4G   0% /sys/fs/cgroup
shm             5.9G     0  5.9G   0% /dev/shm
tmpfs           6.4G   24K  6.4G   1% /var/colab
/dev/sda1       114G   41G   74G  36% /etc/hosts
tmpfs           6.4G     0  6.4G   0% /proc/acpi
tmpfs           6.4G     0  6.4G   0% /proc/scsi
tmpfs           6.4G     0  6.4G   0% /sys/firmware
drive           108G   42G   66G  39% /content/drive
```

20163331 전진우 차량지능기초 과제1

2) (1) vehicle_detection

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import glob
import matplotlib.image as mpimg
import time

start=time.time()

images = glob.glob('/content/drive/MyDrive/Colab_Notebooks/colab_git/VehicleDetection/vehicles/KITTI_extracted/*.png')
images += glob.glob('/content/drive/MyDrive/Colab_Notebooks/colab_git/VehicleDetection/non-vehicles/**/*.png')
cars = []
notcars = []

for image in images:
    if 'non-vehicles' in image:
        notcars.append(mpimg.imread(image))
    else:
        cars.append(mpimg.imread(image))

print("time :",time.time() - start)
print('Images loaded.')

#glob를 사용하여 image를 불러온다.
#각 이미지는 google-drive와 연동하여 불러오고 car image와 car가 아닌 notcar image로 분류한다.
```

```
import cv2
import matplotlib.image as mpimg
```

```
import numpy as np
from skimage.feature import hog
#함수 설정에 필요한 모듈들을 불러온다.
#cv2는 openCV를 통해 이미지 파일을 읽고 보고 저장할 수 있는 모듈이다
#matplotlib는 그래프를 그릴 수 있게 도와주는 모듈이다
#아래에서 hog를 사용하기 위해서 skimage.feature모듈에서 가져온다.

def get_hog_features(img, orient, pix_per_cell, cell_per_block, vis=False, feature_vec=True):
    if vis:
        features, hog_image = hog(img, orientations=orient,
                                  pixels_per_cell=(pix_per_cell, pix_per_cell),
                                  cells_per_block=(cell_per_block, cell_per_block),
                                  transform_sqrt=True,
                                  visualise=vis, feature_vector=feature_vec)
        return features, hog_image

    else:
        features = hog(img, orientations=orient,
                       pixels_per_cell=(pix_per_cell, pix_per_cell),
                       cells_per_block=(cell_per_block, cell_per_block),
                       transform_sqrt=True,
                       visualise=vis, feature_vector=feature_vec)
        return features
#get_hog_features함수는 HOG기능 및 시각화를 반환하는 함수이다.
#HOG는 대상 영역을 일정 크기의 셀로 분할하고, 각 셀마다 edge 픽셀(gradient magnitude가
#일정 값 이상인 픽셀)들의 방향에 대한 히스토그램을 구한 후 이들 히스토그램 bin 값들을 일렬로
#연결한 벡터이다. 즉, HOG는 edge의 방향 히스토그램 템플릿으로 볼 수 있다.
#feature와 hog_image가 같으면 두 출력으로, 그렇지 않으면 한 출력으로 호출한다.


def bin_spatial(img, size=(32, 32)):
    features = cv2.resize(img, size).ravel()
    return features
#bin_spatial함수는 빈 색상의 특징을 계산하는 함수이다.
#cv2.resize().ravel()을 사용하여 형상 벡터를 생성한다.
#그 후 feature를 반환한다.


def color_hist(img, nbins=32, bins_range=(0., 1.)):
```

```
    channel1_hist = np.histogram(img[:, :, 0], bins=nbins, range=bins_range)
    channel2_hist = np.histogram(img[:, :, 1], bins=nbins, range=bins_range)
    channel3_hist = np.histogram(img[:, :, 2], bins=nbins, range=bins_range)
    hist_features = np.concatenate((channel1_hist[0], channel2_hist[0], channel3_hist[0]))

    return hist_features
#color_hist함수는 색상 히스토그램 기능을 계산하는 함수이다.
#mpimg로 .png파일을 읽는 경우 bins_range을 변경해야하기 때문에 사용한다.
#커널 채널의 히스토그램을 별도로 계산하여 단일 형상 벡터에 연결하고 이들을 반환한다.


def extract_features(imgs, color_space='RGB', spatial_size=(32, 32),
                     hist_bins=32, orient=9,
                     pix_per_cell=8, cell_per_block=2, hog_channel=0,
                     spatial_feat=True, hist_feat=True, hog_feat=True):
 #extract_features함수는 영상 목록에서 형상을 추출하는 기능을 한다.
 #이 기능들을 bin_spatial(), color_hist()라고 정의한다.

    features = []    #형상 벡터를 추가할 목록을 작성할 빈 리스트를 작성한다.
    for image in imgs:
        file_features = []
        if color_space != 'RGB': #RGB 이외의 경우에 색상 변환을 적용한다.
            if color_space == 'HSV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
            elif color_space == 'LUV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2LUV)
            elif color_space == 'HLS':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
            elif color_space == 'YUV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YUV)
            elif color_space == 'YCrCb':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YCrCb)
        else:
            feature_image = np.copy(image)

        if spatial_feat:
            spatial_features = bin_spatial(feature_image, size=spatial_size)
            file_features.append(spatial_features)
```

```
            if hist_feat:
                hist_features = color_hist(feature_image, nbins=hist_bins)
                file_features.append(hist_features)
            if hog_feat: #새로운 feature들을 기존 feature목록에 추가한다.
                if hog_channel == 'ALL':
                    hog_features = []
                    for channel in range(feature_image.shape[2]):
                        hog_features.append(get_hog_features(feature_image[:, :, channel],
                                                             orient, pix_per_cell, cell_per_block,
                                                             vis=False, feature_vec=True))
                    hog_features = np.ravel(hog_features)
                else:
                    hog_features = get_hog_features(feature_image[:, :, hog_channel], orient,
                                                    pix_per_cell, cell_per_block, vis=False, feature_vec=True)

                file_features.append(hog_features)
            features.append(np.concatenate(file_features))

        return features #최종적으로 형상 벡터의 feature를 반환한다.


    def slide_window(img, x_start_stop=[None, None], y_start_stop=[None, None],
                     xy_window=(64, 64), xy_overlap=(0.5, 0.5)):
    #slide_window함수는 이미지를 촬영하는 기능을 정의한다.
    #x와 y의 시작과 정지 위치, 창 크기(x,y 크기), 겹치는 부분에 대해 정의한다

        if x_start_stop[0] == None:
            x_start_stop[0] = 0
        if x_start_stop[1] == None:
            x_start_stop[1] = img.shape[1]
        if y_start_stop[0] == None:
            y_start_stop[0] = 0
        else:
            y_start_stop[0] = int(img.shape[0] * y_start_stop[0] / 100.)
        if y_start_stop[1] == None:
            y_start_stop[1] = img.shape[0]
        else:
            y_start_stop[1] = int(img.shape[0] * y_start_stop[1] / 100.)
    #x또는 y. 시작과 정지 위치가 정의되지 않은 경우에 영상 크기로 설정한다.
```

```
    print("Y start and stop: {}".format(y_start_stop))

    xspan = x_start_stop[1] - x_start_stop[0]
    yspan = y_start_stop[1] - y_start_stop[0]
    nx_pix_per_step = np.int(xy_window[0] * (1 - xy_overlap[0]))
    ny_pix_per_step = np.int(xy_window[1] * (1 - xy_overlap[1]))
    nx_windows = np.int(xspan / nx_pix_per_step) - 1
    ny_windows = np.int(yspan / ny_pix_per_step) - 1
#위는 검색할 영역의 범위를 계산한다.
#step당 pixel수를 x/y단위로 계산하고 창 수도 계산한다.
#목록을 초기화한다.
    window_list = []

    for ys in range(ny_windows):
        for xs in range(nx_windows):
            startx = xs * nx_pix_per_step + x_start_stop[0]
            endx = startx + xy_window[0]
            starty = ys * ny_pix_per_step + y_start_stop[0]
            endy = starty + xy_window[1]
            window_list.append(((startx, starty), (endx, endy)))
    return window_list
#x와 y윈도우 위치를 찾는 for 반복문이다.
#창을 위치를 계산하고 목록에 창 위치를 추가해 창 목록을 반환한다.

def draw_boxes(img, bboxes, color=(0, 0, 255), thick=6):
    imcopy = np.copy(img)
    for bbox in bboxes:
        cv2.rectangle(imcopy, bbox[0], bbox[1], color, thick)
    return imcopy
#draw_boxex는 경계 상자를 그리는 기능을 정의한다.
#이미지를 복사하고 경계 상자 사이를 반복한다.
#주어진 bbox 좌표를 그리고 최종적으로 box가 그려진 이미지 복사본을 반환한다.



import cv2
import numpy as np
from lesson_functions import bin_spatial, color_hist, get_hog_features
#위와 마찬가지로 필요한 모듈들을 가져온다.
```

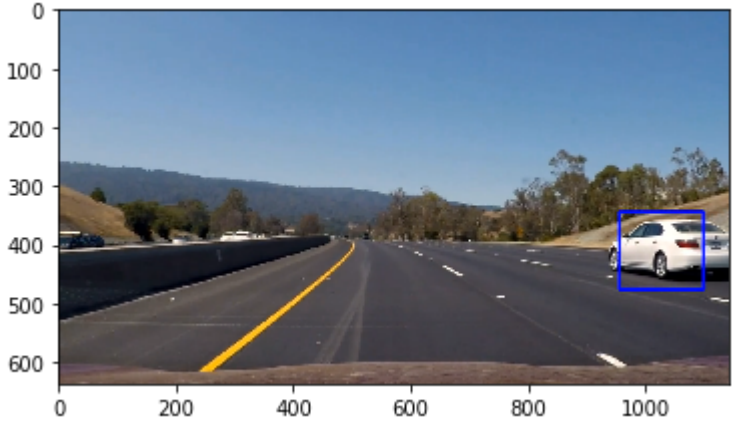
```
def single_img_features(img, color_space='RGB', spatial_size=(32, 32),
                        hist_bins=32, orient=9,
                        pix_per_cell=8, cell_per_block=2, hog_channel=0,
                        spatial_feat=True, hist_feat=True, hog_feat=True):
#single_img_features함수는 단일 이미지 창에서 형상을 추출하는 기능을 정의한다.
#extract_feature과 유사하며 이미지 리스트가 아닌 단일 이미지용이다.
    img_features = [] #기능을 수신할 빈 목록을 정의한다.
    if color_space != 'RGB': #RGB이외의 경우 색상 변환을 적용한다.
        if color_space == 'HSV':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
        elif color_space == 'LUV':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2LUV)
        elif color_space == 'HLS':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
        elif color_space == 'YUV':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
        elif color_space == 'YCrCb':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2YCrCb)
    else:
        feature_image = np.copy(img)
    if spatial_feat:  #flag가 설정된 경우 공간 특성을 계산한다.
        spatial_features = bin_spatial(feature_image, size=spatial_size)
        img_features.append(spatial_features) #img_feature목록에 특정 feature를 추가한다.
    if hist_feat:  #flag가 설정된 경우 히스토그램 feature를 계산한다.
        hist_features = color_hist(feature_image, nbins=hist_bins)
        img_features.append(hist_features) #img_feature목록에 특정 feature를 추가한다.
    if hog_feat: #flag가 설정된 경우 HOG feature를 계산한다.
        if hog_channel == 'ALL':
            hog_features = []
            for channel in range(feature_image.shape[2]):
                hog_features.extend(get_hog_features(feature_image[:, :, channel],
                                                     orient, pix_per_cell, cell_per_block,
                                                     vis=False, feature_vec=True))
        else:
            hog_features = get_hog_features(feature_image[:, :, hog_channel], orient,
                                            pix_per_cell, cell_per_block, vis=False, feature_vec=True)
        img_features.append(hog_features) #img_feature목록에 특정 feature를 추가한다.
    return np.concatenate(img_features)
```

```
#최종적으로 img_feature를 반환한다.

def search_windows(img, windows, clf, scaler, color_space='RGB',
                   spatial_size=(32, 32), hist_bins=32,
                   hist_range=(0, 256), orient=9,
                   pix_per_cell=8, cell_per_block=2,
                   hog_channel=0, spatial_feat=True,
                   hist_feat=True, hog_feat=True):
#search_windows함수는 이미지를 전달하는 기능을 정의한다.

    on_windows = [] #빈 목록을 생성한다.
    for window in windows: #목록의 모든 window창에 반복한다.
        test_img = cv2.resize(img[window[0][1]:window[1][1], window[0][0]:window[1][0]], (64, 64))
        #원본 영상에서 test_window를 추출한다.
        features = single_img_features(test_img, color_space=color_space,
                                       spatial_size=spatial_size, hist_bins=hist_bins,
                                       orient=orient, pix_per_cell=pix_per_cell,
                                       cell_per_block=cell_per_block,
                                       hog_channel=hog_channel, spatial_feat=spatial_feat,
                                       hist_feat=hist_feat, hog_feat=hog_feat)
        #single_img_feature를 사용하여 해당 window창의 feature를 추출한다.
        test_features = scaler.transform(np.array(features).reshape(1, -1))
        #추출된 feature를 분류기로 전달한다.
        prediction = clf.predict(test_features)
        #분류기를 사용하여 예측한다
        if prediction == 1:
            on_windows.append(window)
    return on_windows
        #최종적으로on_windows를 반환한다.


color_space = 'HSV'  #color_space를 HSY로 지정한다.
orient = 8           #HOG orient=8로 지정한다.
pix_per_cell = 8     #HOG의 셀 당 pixel수=8로 지정한다.
cell_per_block = 2   #HOG의 block 당 셀 수=2로 지정한다.
hog_channel = 'ALL'  #HOG 채널은 ALL로 지정한다.
spatial_size = (32, 32)  #차원 공간 치수를 지정한다.
hist_bins = 16           #히스토그램의 bin을 지정한다.
spatial_feat = False     #공간적 특징은 끈다.
```

```
    hist_feat = True         #히스토그램 feature기능을 켠다.
    hog_feat = True          #HOG feature기능을 켠다.
    y_start_stop = [50, 100] #slide_window에서 검색할 최소값과 최대값을 정의한다.

    car_features = extract_features(cars, color_space=color_space,
                                    spatial_size=spatial_size, hist_bins=hist_bins,
                                    orient=orient, pix_per_cell=pix_per_cell,
                                    cell_per_block=cell_per_block,
                                    hog_channel=hog_channel, spatial_feat=spatial_feat,
                                    hist_feat=hist_feat, hog_feat=hog_feat)
    notcar_features = extract_features(notcars, color_space=color_space,
                                       spatial_size=spatial_size, hist_bins=hist_bins,
                                       orient=orient, pix_per_cell=pix_per_cell,
                                       cell_per_block=cell_per_block,
                                       hog_channel=hog_channel, spatial_feat=spatial_feat,
                                       hist_feat=hist_feat, hog_feat=hog_feat)

    X = np.vstack((car_features, notcar_features)).astype(np.float64)

    X_scaler = StandardScaler().fit(X)
    #각 열당 scaler를 정의한다.
    scaled_X = X_scaler.transform(X)
    #scaler를 x에 저장한다.
    y = np.hstack((np.ones(len(car_features)), np.zeros(len(notcar_features))))
    #라벨 벡터를 정의하고, 데이터를 랜덤화된 훈련 및 시험 세트로 분할한다.
    rand_state = np.random.randint(0, 100)
    X_train, X_test, y_train, y_test = train_test_split(
        scaled_X, y, test_size=0.2, random_state=rand_state)

    print('Using:', orient, 'orientations', pix_per_cell,
          'pixels per cell and', cell_per_block, 'cells per block')
    print('Feature vector length:', len(X_train[0]))

    svc = LinearSVC()
    #선형 SVC를 사용한다.
    t = time.time()
    svc.fit(X_train, y_train)
    t2 = time.time()
    print(round(t2 - t, 2), 'Seconds to train SVC...')
```

```
print('Test Accuracy of SVC = ', round(svc.score(X_test, y_test), 4))
t = time.time()
image = mpimg.imread('test_images/test6.jpg')
draw_image = np.copy(image)
#SVC의 time.time 훈련 시간을 확인하고 SVC score를 확인한다.

image = image.astype(np.float32)/255
windows = slide_window(image, x_start_stop=[None, None], y_start_stop=y_start_stop,
                       xy_window=(96, 96), xy_overlap=(0.5, 0.5))
hot_windows = search_windows(image, windows, svc, X_scaler, color_space=color_space,
                             spatial_size=spatial_size, hist_bins=hist_bins,
                             orient=orient, pix_per_cell=pix_per_cell,
                             cell_per_block=cell_per_block,
                             hog_channel=hog_channel, spatial_feat=spatial_feat,
                             hist_feat=hist_feat, hog_feat=hog_feat)
window_img = draw_boxes(draw_image, hot_windows, color=(0, 0, 255), thick=6)
#훈련을 진행한 후 다음 열의 주석을 제거한다.

plt.imshow(window_img)
plt.show()
#최종적으로 차량에 대한 경계box가 생성된 모습을 볼 수 있다.



import matplotlib.pyplot as plt
import matplotlib.image as mpimg
image = mpimg.imread('/content/drive/MyDrive/Colab_Notebooks/colab_git/VehicleDetection/1.png')
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7f9a3a8cb510>
```



✓ 0초 오전 12:46에 완료됨

20163331 전진우 차량지능기초 과제1

2) (2) road_lane_line_detection

이 프로젝트의 목표는 학습한 도구를 사용하여 도로의 차선을 식별하는 것이다. 각각의 훈련 데이터에 대한 도구를 설정하고 결과를 비디오로 보여준다. line_segment를 식별하기 위해서 평균화하거나 추정하여 차선 전체를 그려야 한다. 사용하는 도구로는 color selection, region of interest selection, grayscaling, Gaussian smoothing, Canny Edge Detection and Hough Tranform line detection을 사용한다. 이미지의 line_segment를 식별하여 감지한 후 평균/축소하여 표시할 이미지에 그릴 수 있다.

더블클릭 또는 Enter 키를 눌러 수정

```
from google.colab import drive
drive.mount('/content/drive')
#구글 드라이브와 코렙을 연결하는 코드
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

이미지를 불러오기

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import cv2
%matplotlib inline
#이미지와 사진을 가져오는 모듈과 패키지를 가져옴
```

```
image = mpimg.imread('/content/drive/MyDrive/Colab_Notebooks/road_lane_line_detection/test_images/solidWhiteRight.jpg')
print('This image is:', type(image), 'with dimesions:', image.shape)
plt.imshow(image)
#이미지를 파일이 위치하는 경로를 따라 읽은 후에 이미지의 type과 shape을 보여준다
```

```
This image is: <class 'numpy.ndarray'> with dimesions: (540, 960, 3)
<matplotlib.image.AxesImage at 0x7fd73534e450>
```

필요한 함수 설정

```
import math
#아래부터는 함수를 설정한다.

def grayscale(img):
    return cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#grayscale함수는 이미지를 grayscale로 변환한다.

def canny(img, low_threshold, high_threshold):
    return cv2.Canny(img, low_threshold, high_threshold)
#canny함수는 canny tranform을 정의한다.

def gaussian_blur(img, kernel_size):
    return cv2.GaussianBlur(img, (kernel_size, kernel_size), 0)
#gaussian_blur함수는 Gauss noice kernal을 적용한다.

def region_of_interest(img, vertices):
    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)

        masked_image = cv2.bitwise_and(img, mask)
        return masked_image
    #region_of_interest함수는 먼저 빈 mask를 정의한다.
    #이후 입력 이미지에 따라 마스클르 채울 1 또는 3 channel color를 표시한다
    #'필링'에 의해 정의된 poligon내부에 pixel를 담는다
    #poligon에 의해 정의된 영상의 영역만 유지학 나머지 부분은 검은색으로 표시한다.
    #마스크 pixel이 00이 아닌 경우에만 영상을 표시한다.


    def draw_lines(img, lines, color=[255, 0, 0], thickness=10):
    #draw_lines함수는 시작점으로 사용하는 경우에 사용할 수 있다. 전체 지도를 작성하기 위해서 탐지한 line segment을 평균화/축소한다.
    #line segment는 기울기를 이용하여 right/left 중 어느 segment가 왼쪽의 일부인지 결정해 다음 위치의 평균을 구한다.

        draw_right = True
        draw_left = True

    #아래는 모든 선의 기울기를 찾는다
    #기울기의 절댓값이 경사 임계값(slope_threshold)보다 큰 선만을 확인한다.
        slope_threshold = 0.5
        slopes = []
        new_lines = []
        for line in lines:
            x1, y1, x2, y2 = line[0]
    #경사 계산
            if x2 - x1 == 0.:
                slope = 999.
            else:
                slope = (y2 - y1) / (x2 - x1)

            if abs(slope) > slope_threshold:
                slopes.append(slope)
                new_lines.append(line)
```

```
        lines = new_lines

    #우측 차선 및 좌측 차선 표시로 선을 right과 left 라인으로 분할
    #이 때, 우측/좌측 차선에는 양/음의 기울기가 있어야 하며 영상의 오른쪽/왼쪽에 있어야 한다.
        right_lines = []
        left_lines = []
        for i, line in enumerate(lines):
            x1, y1, x2, y2 = line[0]
            img_x_center = img.shape[1] / 2
            if slopes[i] > 0 and x1 > img_x_center and x2 > img_x_center:
                right_lines.append(line)
            elif slopes[i] < 0 and x1 < img_x_center and x2 < img_x_center:
                left_lines.append(line)

    #선형 회귀 분석을 적용하여 우측/좌측 차선 표시에 가장 적합한 선을 찾는다.
    #우측 차선 찾기
        right_lines_x = []
        right_lines_y = []

        for line in right_lines:
            x1, y1, x2, y2 = line[0]

            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)
        else:
            right_m, right_b = 1, 1
            draw_right = False

    #좌측 차선 찾기
        left_lines_x = []
        left_lines_y = []

        for line in left_lines:
```

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

            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)
        else:
            left_m, left_b = 1, 1
            draw_left = False

    #선을 잇기에 사용하는 우측/좌측의 끝점 2개를 설정한다.
        y1 = img.shape[0]
        y2 = img.shape[0] * (1 - trap_height)

        right_x1 = (y1 - right_b) / right_m
        right_x2 = (y2 - right_b) / right_m

        left_x1 = (y1 - left_b) / left_m
        left_x2 = (y2 - left_b) / left_m

    #위에서 찾은 끝점을 부동점에서 int형으로 변환
        y1 = int(y1)
        y2 = int(y2)
        right_x1 = int(right_x1)
        right_x2 = int(right_x2)
        left_x1 = int(left_x1)
        left_x2 = int(left_x2)

    #이미지에 우측/좌측 선을 표시한다.
        if draw_right:
            cv2.line(img, (right_x1, y1), (right_x2, y2), color, thickness)
        if draw_left:
            cv2.line(img, (left_x1, y1), (left_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, 3), dtype=np.uint8)
    draw_lines(line_img, lines)

    return line_img
#hough_line함수는 image를 canny변환의 결과값을 사용해서 hough_line이 그려진 이미지를 반환한다.
#단일 채널 이미지를 생성한 후 라인을 그린다.


def weighted_img(img, initial_img, α=0.8, β=1., λ=0.):
    return cv2.addWeighted(initial_img, α, img, β, λ)
#weighted_img함수는 hough_line, 선이 그려진 image에 가중치를 더하여
#initial_img*α + img*β+β로 계산한다.

def filter_colors(image):

    white_threshold = 200
    lower_white = np.array([white_threshold, white_threshold, white_threshold])
    upper_white = np.array([255, 255, 255])
    white_mask = cv2.inRange(image, lower_white, upper_white)
    white_image = cv2.bitwise_and(image, image, mask=white_mask)

    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    lower_yellow = np.array([90,100,100])
    upper_yellow = np.array([110,255,255])
    yellow_mask = cv2.inRange(hsv, lower_yellow, upper_yellow)
    yellow_image = cv2.bitwise_and(image, image, mask=yellow_mask)

    image2 = cv2.addWeighted(white_image, 1., yellow_image, 1., 0.)

    return image2
#filter_colors함수는 아래 white와 yellow pixel를 포함하도록 이미지를 필터링하는 함수이다.
#아래는 white pixel로, yello pixel로 나타내는 코드를 보여준다.
#최종적으로 이들을 합쳐 image2라는 값으로 return한다.
```

이미지를 테스트한다.

```
import os
os.listdir("/content/drive/MyDrive/Colab_Notebooks/road_lane_line_detection/test_images/")
```

```
['solidWhiteRight_annotated.jpg',
 'solidWhiteCurve_annotated.jpg',
 'solidWhiteRight.jpg',
 'solidWhiteCurve.jpg',
 'solidYellowLeft.jpg',
 'solidYellowLeft_annotated.jpg',
 'whiteCarLaneSwitch.jpg',
 'whiteCarLaneSwitch_annotated.jpg',
 'solidYellowCurve.jpg',
 'solidYellowCurve_annotated.jpg',
 'solidYellowCurve2.jpg',
 'solidYellowCurve2_annotated.jpg',
 'solidWhiteRight_annotated_annotated.jpg',
 'solidWhiteCurve_annotated_annotated.jpg',
 'solidYellowLeft_annotated_annotated.jpg',
 'whiteCarLaneSwitch_annotated_annotated.jpg',
 'solidYellowCurve_annotated_annotated.jpg',
 'solidYellowCurve2_annotated_annotated.jpg']
```

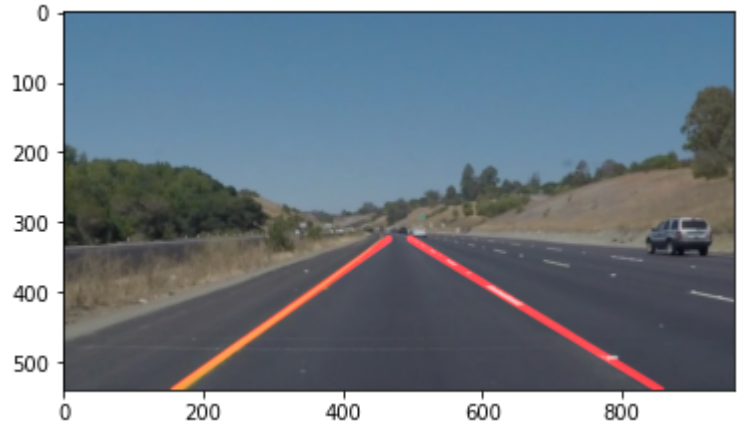
차선을 찾는 pipe_line를 구축한다. 모든 훈련데이터에서 이를 진행하여 확인한다.

```
kernel_size = 3

low_threshold = 50
high_threshold = 150

trap_bottom_width = 0.85
trap_top_width = 0.07
trap_height = 0.4

rho = 2
theta = 1 * np.pi/180
threshold = 15
min_line_length = 10
max_line_gap = 20
```

```
max_line_gap = 20

#global 변수를 설정한다
#Gaussian smoothing에 사용할 kernal_size를 결정하고 Canny edge검출에 임계값을 설정한다
#사다리꼴 모양으로 이미지 아래 엣지를 정하기 위해서 하단/상단에 대해 너비,이미지 폭의 백분율로 표시한다.
#trap_height는 사다리꼴의 영상 백분율로 표현했을 때의 높이이다.
#Hough 변환에 사용되는 hough grid의 거리 분해능, 각도 분해능 등을 지정하고
#연결 가능한 line_segment의 최대 간격을 설정한다.


def annotate_image(image_in):

    image = filter_colors(image_in)

    gray = grayscale(image)

    blur_gray = gaussian_blur(gray, kernel_size)

    edges = canny(blur_gray, low_threshold, high_threshold)

    imshape = image.shape
    vertices = np.array([[W
        ((imshape[1] * (1 - trap_bottom_width)) // 2, imshape[0]),W
        ((imshape[1] * (1 - trap_top_width)) // 2, imshape[0] - imshape[0] * trap_height),W
        (imshape[1] - (imshape[1] * (1 - trap_top_width)) // 2, imshape[0] - imshape[0] * trap_height),W
        (imshape[1] - (imshape[1] * (1 - trap_bottom_width)) // 2, imshape[0])]]W
        , dtype=np.int32)
    masked_edges = region_of_interest(edges, vertices)

    line_image = hough_lines(masked_edges, rho, theta, threshold, min_line_length, max_line_gap)

    initial_image = image_in.astype('uint8')
    annotated_image = weighted_img(line_image, initial_image)

    return annotated_image

#annotate_image함수는 이미지 numpy배열이 주어지면 주석이 달린 이미지를 numpy배열로 반환한다.
#이미지에 흰색과 노란색 픽셀을 유지하면 다른 모든 픽셀은 검은색으로 변환시키고
#위에서 정의한 다양한 함수들을 사용한다.(grayscale, grassian_blur, hough_lines 등)
#최종적으로 원본 영상에 차선을 그린다.
```

```
#예시 이미지를 보여준다
#우측/좌측 차선에 맞추어 원본 이미지의 차선에 그려진 것을 확인할 수 있다.
annotated_image = annotate_image(mpimg.imread('/content/drive/MyDrive/Colab_Notebooks/road_lane_line_detection/test_images/solidYellowCur
plt.imshow(annotated_image)
```

```
<matplotlib.image.AxesImage at 0x7fd7352af990>
```




```
test_images = os.listdir('/content/drive/MyDrive/Colab_Notebooks/road_lane_line_detection/test_images')

for image in test_images:
    annotated_image = annotate_image(mpimg.imread((os.path.join('/content/drive/MyDrive/Colab_Notebooks/road_lane_line_detection/test_ima
    fname = image.split('.')[0] + '_annotated.jpg'
    plt.imsave(os.path.join('/content/drive/MyDrive/Colab_Notebooks/road_lane_line_detection/test_images', fname), annotated_image)

#모든 test_images/*.jpg에 주석을 다는 작업이다.
#주석이 달린 영상을 test_images/<ori~> 로 저장한다.
```

비디오 테스트를 진행한다.

```
from moviepy.editor import VideoFileClip
from IPython.display import HTML

#비디오 편집/저장/보기에 필요한 모듈과 함수를 불러온다
```

```
def process_image(image):

    result = annotate_image(image)

    return result
#process_image함수는 image를 input으로 받아 annota_image함수에 넣어 반환한다.


white_output = 'white.mp4'
clip1 = VideoFileClip("/content/drive/MyDrive/Colab_Notebooks/road_lane_line_detection/solidWhiteRight.mp4")
white_clip = clip1.fl_image(process_image)
%time white_clip.write_videofile(white_output, audio=False)
#최종적으로 동영상으로 표시한다.
#차선 전체를 식별하고 비디오를 볼 수 있다.
```

```
[MoviePy] >>>> Building video white.mp4
[MoviePy] Writing video white.mp4
100%|██████████| 221/222 [00:09<00:00, 22.26it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: white.mp4

CPU times: user 4.66 s, sys: 368 ms, total: 5.02 s
Wall time: 11.3 s
```

```
HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
</video>
""".format(white_output))
```

```
yellow_output = 'yellow.mp4'
clip2 = VideoFileClip('/content/drive/MyDrive/Colab_Notebooks/road_lane_line_detection/solidYellowLeft.mp4')
yellow_clip = clip2.fl_image(process_image)
%time yellow_clip.write_videofile(yellow_output, audio=False)
```

```
[MoviePy] >>>> Building video yellow.mp4
[MoviePy] Writing video yellow.mp4
100%|██████████| 681/682 [00:33<00:00, 20.19it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: yellow.mp4

CPU times: user 14.9 s, sys: 1.06 s, total: 16 s
Wall time: 35.2 s
```

```
HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
```