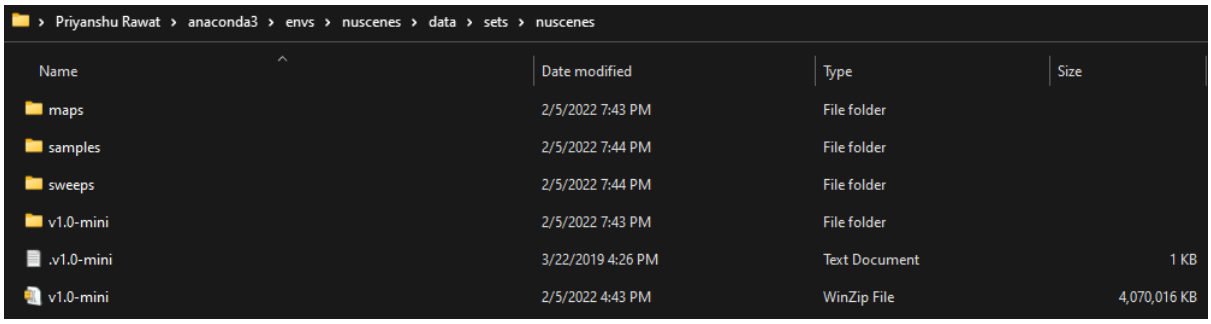
AuE 8200: Machine Perception and Intelligence

Homework 2

Submitted by: *- Priyanshu Rawat*

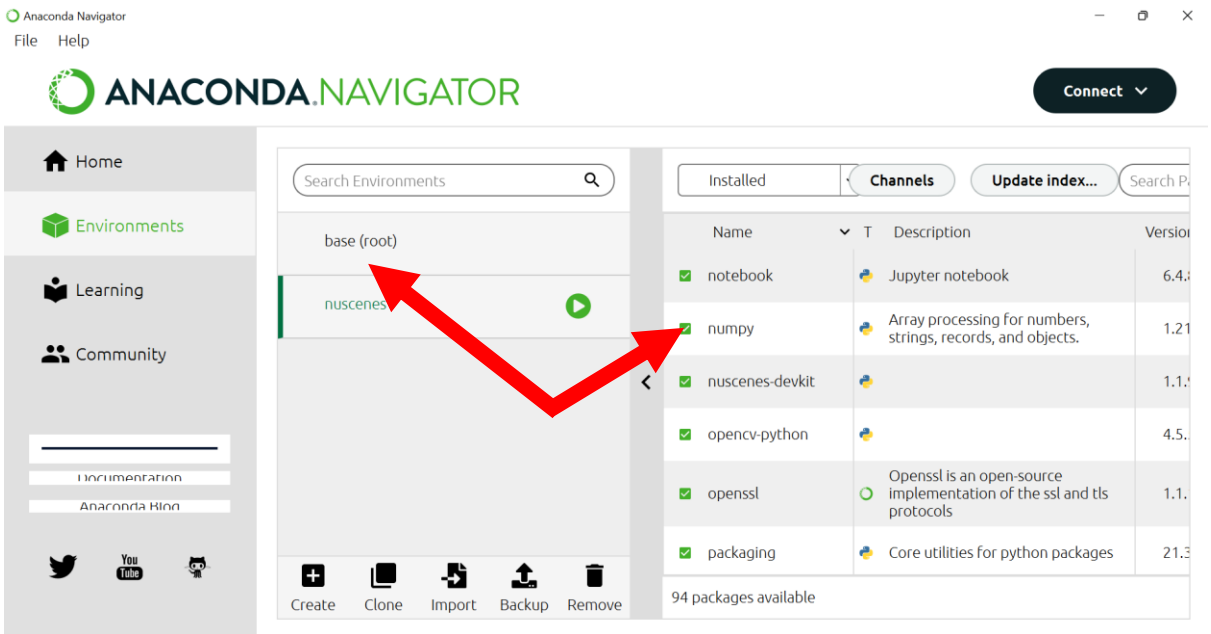
1. **For NuScene dataset access, you may need to register on that website. To save time, you can download only the Full dataset/Mini set: (5 point)**

Screenshot of downloaded and extracted the required dataset.



2. **If you use Python, set up the NuScene develop kit locally, you may need to install Anaconda and Jupyter notebook; If you use Matlab, setup your Matlab for this data process. (5 point)**

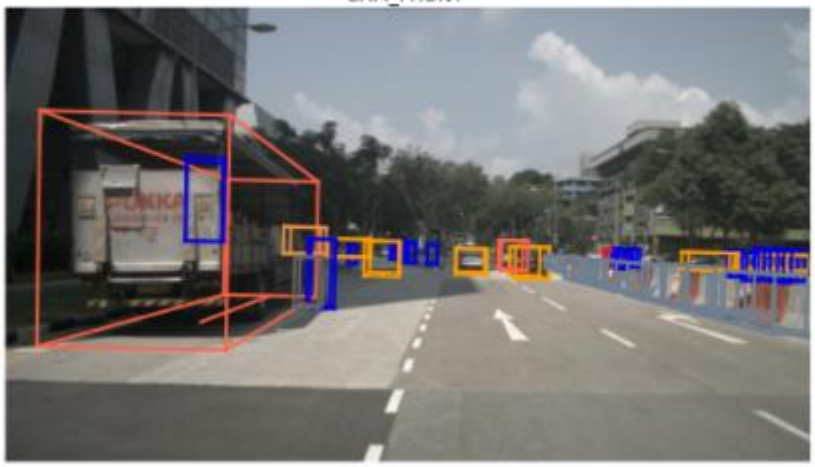
Screenshot of '<u>nuscenes</u>' environment in Anaconda and installed '<u>nuscenes-devkit</u>' in it.



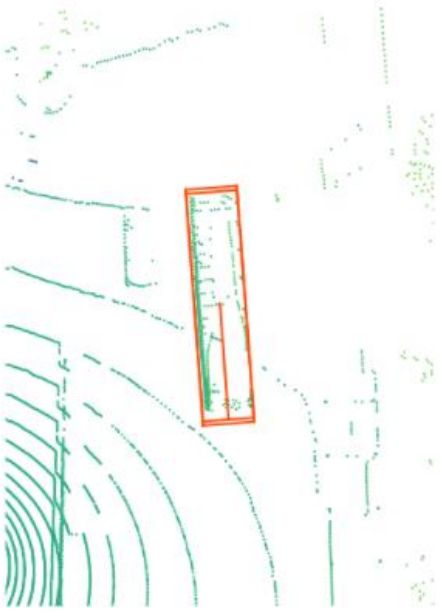
3. **Pickup a set of data, including Image, Lidar, and Radar data. Visualize them respectively. If you use Python, you can refer to NuScene dev-kit tutorial reference code. (10 points)**

```
%matplotlib inline
from nuscenes.nuscenes import NuScenes

nusc = NuScenes(version='v1.0-mini', dataroot='/users/priya/data/sets/nuscenes/', verbose=True)
```

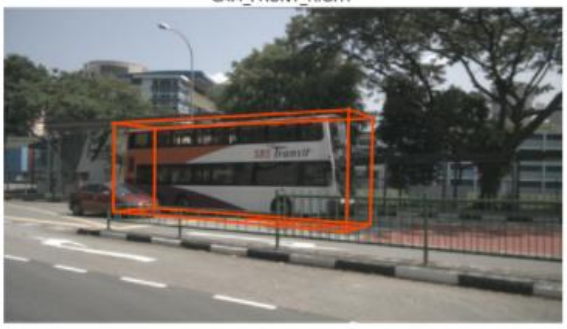
CAM_FRONT



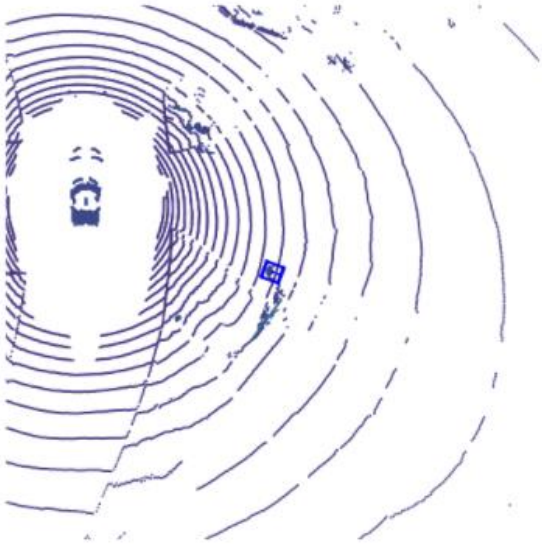
First annotated sample of this instance:



CAM_FRONT_RIGHT

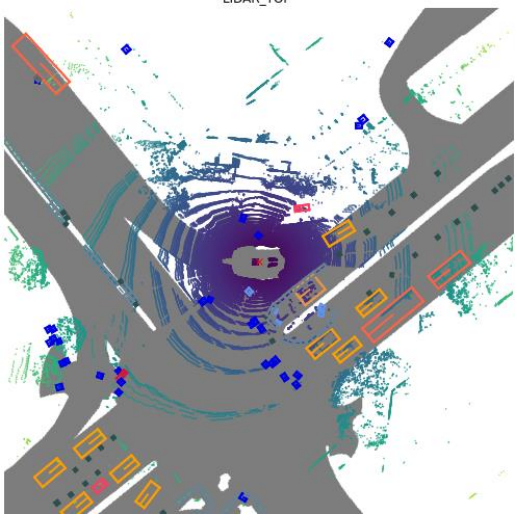


Visibility: {'description': 'visibility of whole object is between 80 and 100%', 'token': '4', 'level': 'v80-100'}

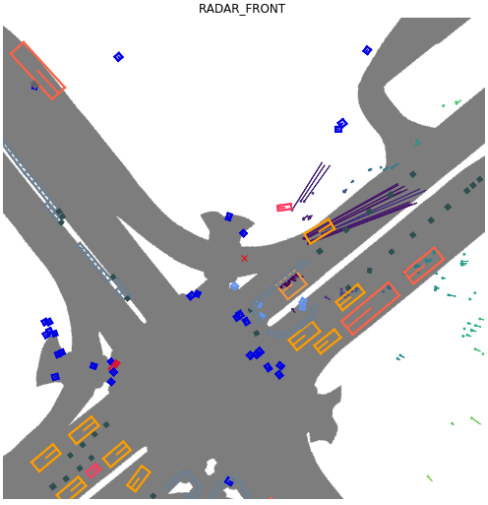


CAM_BACK_RIGHT

LIDAR_TOP



RADAR_FRONT



4. Rather than using NuScene dev-kit, implement below by yourself (total 35 points):

(1) Visualize images (you can use library OpenCV or others), Sample code. (5')

```
# Visualizing camera image using opencv

import numpy as np
import cv2
import sys

img = cv2.imread("/users/priya/data/sets/nuscenes/samples/CAM_FRONT/n008-2018-08-01-15-16-36-0400__CAM_FRONT__153151603512
window_name = 'image'

cv2.imshow(window_name, img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

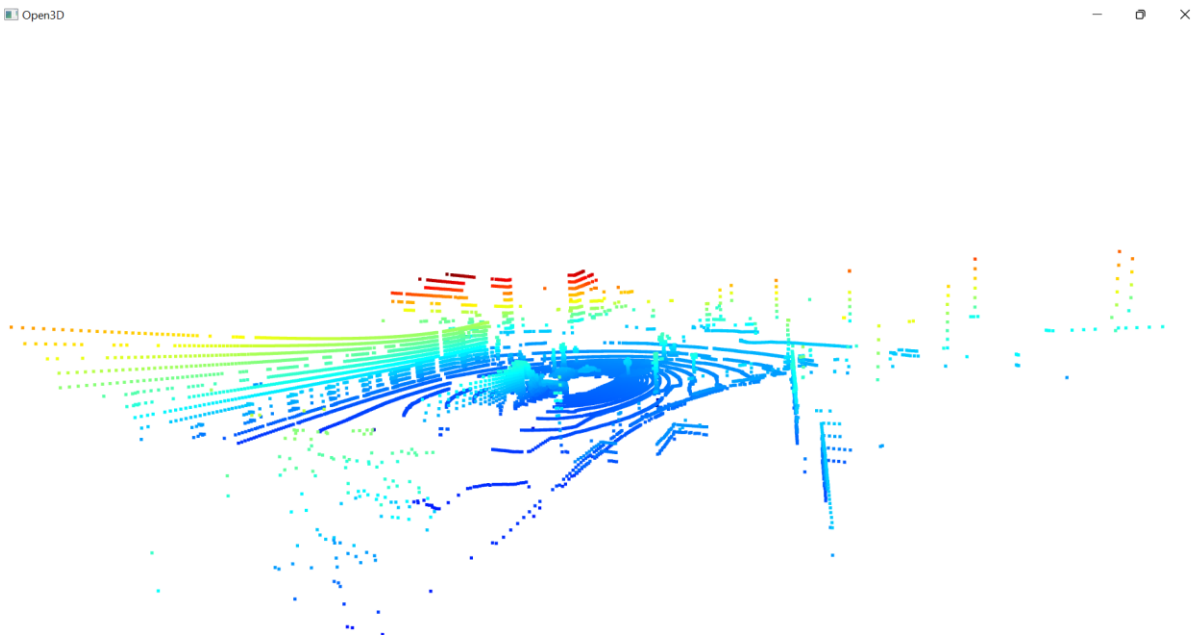
(2) Visualize Lidar point cloud data

Colorize points by height, intensity, and semantic label respectively.

i. Height is the Z value for a point. (5')
ii. You can get intensity referring the code here. (5')
iii. You can get semantic label from the sample above code. (5')

Visualizing and colorizing lidar points by height and intensity -

```
##### Visualizing and colorizing lidar points by height and intensity  #####

import open3d as o3d
import numpy as np

seg_name='/users/priya/lidarseg/v1.0-mini/4484110755904050a880043268149497_lidarseg.bin'
seg=np.fromfile(seg_name, dtype=np.int8)

pcd_name='/users/priya/data/sets/nuscenes/samples/LIDAR_TOP/n008-2018-08-28-16-43-51-0400__LIDAR_TOP__1535489296047917.pcd.b
scan=np.fromfile(pcd_name, dtype=np.float32)
points = scan.reshape((-1, 5))[:, :4]

pcd = o3d.geometry.PointCloud()
pcd.points = o3d.utility.Vector3dVector(points[:, :3])

o3d.visualization.draw_geometries([pcd])
```



Colorizing LiDAR points by semantic label -

```
##### Visualizing liDAR point cloud data and Colorizing LiDAR points by semantic label #####

import open3d as o3d
import numpy as np

seg_name='/users/priya/lidarseg/v1.0-mini/4484110755904050a880043268149497_lidarseg.bin'
seg=np.fromfile(seg_name, dtype=np.int8)

color = np.zeros([len(seg), 3])
color[:, 0] = seg/32
color[:, 1] = seg/32
color[:, 2] = seg/32

pcd_name='/users/priya/data/sets/nuscenes/samples/LIDAR_TOP/n008-2018-08-28-16-43-51-0400__LIDAR_TOP__1535489296047917.pcd.b
scan=np.fromfile(pcd_name, dtype=np.float32)
points = scan.reshape((-1, 5))[:, :4]

pcd = o3d.geometry.PointCloud()
pcd.points = o3d.utility.Vector3dVector(points[:, :3])
pcd.colors = o3d.utility.Vector3dVector(color)

o3d.visualization.draw_geometries([pcd])
```

(3) Visualize Radar data

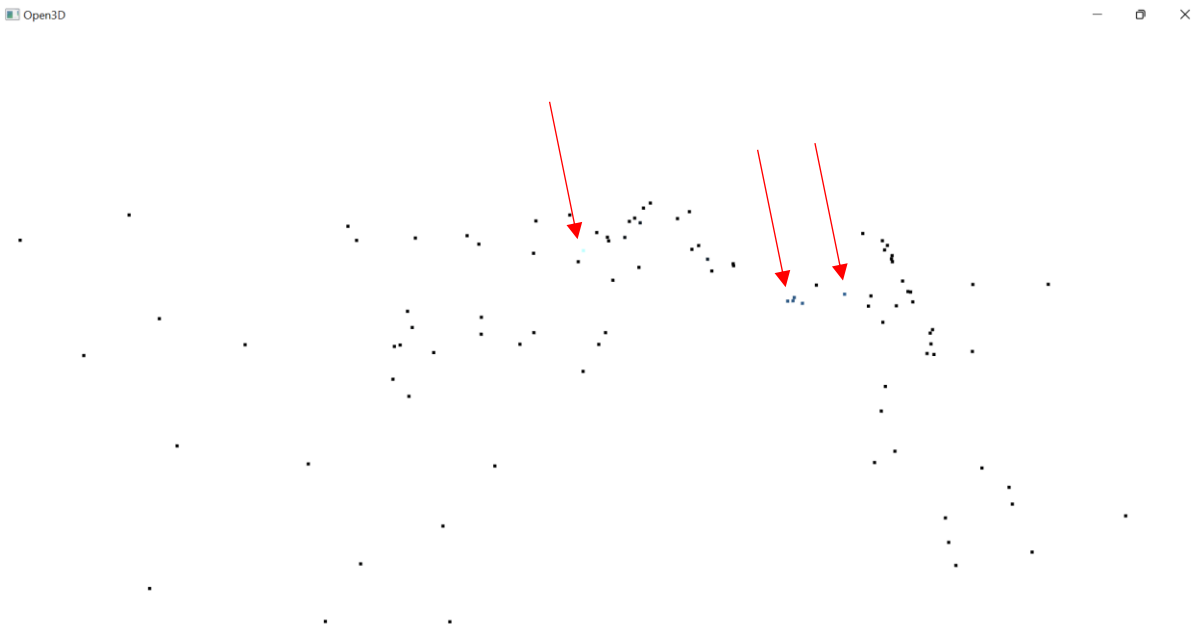
c. Use any other library (e.g., Open3D, PCL, etc) or modify the previous sample code to visualize the Radar data which you chosen. (5')

```
# Visualizing Radar data points

pcd = o3d.io.read_point_cloud("/users/priya/data/sets/nuscenes/samples/RADAR_FRONT/n008-2018-08-01-15-16-36-0400__RADAR_FRONT
o3d.visualization.draw_geometries([pcd])
```

d. Colorize points by below two variable aspects respectively.

i. For height, (5')
ii. For velocity, you can find some velocity information from here.

Colorizing Radar points by velocity -



5. Using NuScene dev-kit for the set of data which you picked up: (45 points)

(1) Visualize Radar data projection on image

a. Print calibration info (between Radar and Camera sensors) by referring code here. (5')


```
{'token': '1d31c729b073425e8e0202c5c6e66ee1', 'sensor_token': '725903f5b62f56118f4094b46a4470d8', 'translation': [1.70079118
954, 0.0159456324149, 1.51095763913], 'rotation': [0.4998015430569128, -0.5030316162024876, 0.4997798114386805, -0.497370838
24542755], 'camera_intrinsic': [[1266.417203046554, 0.0, 816.2670197447984], [0.0, 1266.417203046554, 491.50706579294757],
[0.0, 0.0, 1.0]]}
```


b. Explain the above calibration info, and pipeline of First Fifth steps in the code. (10')

The above calibration information from 'RADAR data projection on image' gives the details about the token of RADAR along with the details of 'Rotation' and 'Translation' (the two quaternions).

The token in the ID generated for each sensor is unique and it refers to a particular data frame of a scene. On the other hand, the quaternions represent the spatial orientations and rotations of the sensor is a 3D space. In other words, the quaternions display the alignment and position of the respected sensors in terms of a common global coordinate frame of reference.

Rotation Quaternion is represented in [w, x, y, z]; here, x, y, z are the coordinate axes and w is the rotation factor. Also, 'w' is the inverse proportional of the amount of sensor rotation during calibration. With a value close to '1' representing no rotation of sensor. In the above, calibration data the value of 'w' lies very close to '1' meaning there Is no rotation. It is reasonable since the camera and Front RADAR both are mounted in the front of the vehicle.

The process of calibration can be explained in the pipeline of five steps:

1. First step deals with the transformation of three-dimensional point cloud data to the ego vehicle frame for the timestamp of the sweep.

```
# First step: transform the pointcloud to the ego vehicle frame for the timestamp of the sweep.
cs_record = nusc.get('calibrated_sensor', pointsensor['calibrated_sensor_token'])
pc.rotate(Quaternion(cs_record['rotation']).rotation_matrix)
pc.translate(np.array(cs_record['translation']))
```

2. Second step deals with the transformation of this data from ego vehicle frame to the global frame.

```
# Second step: transform from ego to the global frame.
poserecord = nusc.get('ego_pose', pointsensor['ego_pose_token'])
pc.rotate(Quaternion(poserecord['rotation']).rotation_matrix)
pc.translate(np.array(poserecord['translation']))
```

3. Third step deals with the transformation of data from global back to the ego vehicle frame for the timestamp of the image.

```
# Third step: transform from global into the ego vehicle frame for the timestamp of the image.
poserecord = nusc.get('ego_pose', cam['ego_pose_token'])
pc.translate(-np.array(poserecord['translation']))
pc.rotate(Quaternion(poserecord['rotation']).rotation_matrix.T)
```

4. Fourth step relates to the transforming of ego vehicle coordinates to the coordinate frame of the camera. In this step, the local coordinates from RADAR go through the rotation process to align with the axes of the Camera.

```
# Fourth step: transform from ego into the camera.
cs_record = nusc.get('calibrated_sensor', cam['calibrated_sensor_token'])
pc.translate(-np.array(cs_record['translation']))
pc.rotate(Quaternion(cs_record['rotation']).rotation_matrix.T)
```

5. In this last step, the calibrated sensor data is combined by extracting all the necessary information.

```
# Fifth step: actually take a "picture" of the point cloud.
# Grab the depths (camera frame z axis points away from the camera).
depths = pc.points[2, :]
```

c. Visualize Radar data projection on image based on calibration info. (10')

(2) Visualize LiDAR data projection on image

d. Print and explain the calibration info (between LiDAR and Camera sensors) by referring here. (5')


```
{'token': '1d31c729b073425e8e0202c5c6e66ee1', 'sensor_token': '725903f5b62f56118f4094b46a4470d8', 'translation': [1.70079118
954, 0.0159456324149, 1.51095763913], 'rotation': [0.4998015430569128, -0.5030316162024876, 0.4997798114386805, -0.497370838
24542755], 'camera_intrinsic': [[1266.417203046554, 0.0, 816.2670197447984], [0.0, 1266.417203046554, 491.50706579294757],
[0.0, 0.0, 1.0]]}
```


The above calibration information from 'LiDAR data projection on image' gives the details about the token of LiDAR along with the details of 'Rotation' and 'Translation' (the two quaternions).

The token is the unique ID generated for each sensor is unique and it refers to a particular data frame of a scene. On the other hand, the quaternions represent the spatial orientations and

rotations of the sensor is a 3D space. In other words, the quaternions display the alignment and position of the respected sensors in terms of a common global coordinate frame of reference.

Rotation Quaternion is represented in [w, x, y, z]; here, x, y, z are the coordinate axes and w is the rotation factor. Also, 'w' is the inverse proportional of the amount of sensor rotation during calibration. With a value close to '1' representing no rotation of sensor.

e. Visualize LiDAR data projection on image based on calibration info. (15')