BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS

DEPARTMENT OF TRANSPORTATION ENGINEERING AND VEHICLE ENGINEERING

MACHINE VISION

3D Object Detection with Point cloud using unsupervised machine learning

BY*:* SHYAMSUNDAR SUDHARSANAM

SUPERVISOR*:* GAZDAG SÁNDOR

25 November 2022

TABLE OF CONTENTS

INTRODUCTION

PROBLEM STATEMENT

In this homework, the goal is to perform the 3D Object detection using the LiDAR point cloud data, as lidar data provides real world scene.

NECESSESSITY OF 3D OBJECT DETECTION

The fundamental and difficult issue is object detection in computer vision. The objective of object detection is to identify each instance of an object in the provided image and determine its location and category. Applications like autonomous driving, however, demand not just the positions of objects in image space but also the precise depth at which they were detected.

In 3D space, 3D object detection attempts to predict the positions, dimensions, and classes of critical components, such as vehicles, people, bicycles, and other objects. While 2D object detection merely creates 2D bounding boxes on images and ignores the actual distance information of objects from the ego-vehicle, 3D object detection concentrates on the localization and identification of things in the real-world 3D coordinate system. The geometric information predicted by 3D object detection in real-world coordinates can be directly utilized to measure the distances between the ego-vehicle and critical objects, and to further help plan driving routes and avoid collision. Furthermore, besides in autonomous cars, 3D object detection is also used for robotics

AIM

We perform the 3D object detection using unsupervised machine learning technique with the help of DBSCAN and RANSAC to process the point cloud before generating the 3D bounding box.

DEVELOPMENT

ENVIRONMENT

Program is written in the jupyter notebook and runs in windows environment, and the open3d library is utilized for processing the point cloud data. The visualization of the point cloud works currently in open3d only on jupyter notebook and this is the reason I restricted to run the program in jupyter notebook, else would have used COLAB. Additional libraries such as Numpy, Pandas and Matplotlib were also used.

DATASET

KITTI dataset is used for this project , which is used as common for working with lidar point cloud data for autonomous driving. KITTI portal provides Velodyne point clouds, Camera Calibration , Labels and color images and only downloaded the velodyne point cloud and color images from the portal. Within the point cloud and color image file, training and testing data exists. Unsupervised learning doesn't require the splitting of data and we can use any of the datasets, and similarly we don't need labels file. The dataset has classes such as car, sign, road, pedestrian, fence, pole, sidewalk, and bicyclist.

INPUTS AND OUTPUTS

In the kitti velodyne point clouds there are 7481 Training files and 7518 testing files, we restricted to use only 53 point cloud files which are stored in binary format.

During processing of each stage like voxel down sampling, segmentation and clustering , open3d library opens new window to display the output. As mentioned earlier, the visualization is not compatible with other environments like COLAB.

PIPELINE STAGES

In unsupervised machine learning, we can detect 3d object in point cloud with series of pipeline steps as shown below

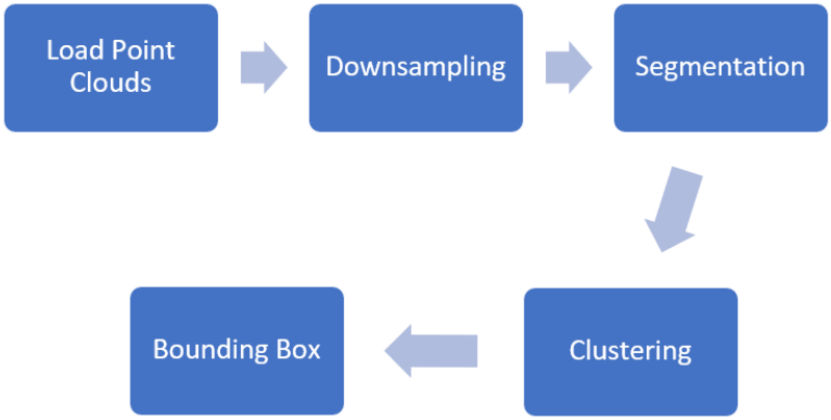

Figure 1: pipeline of unsupervised 3D object detection technique

SOURCE CODE

Initially , all the required libraries are imported in the first cell, which is shown below

```
import numpy as np
import time
import open3d
import pandas as pd
import matplotlib.pyplot as plt
import struct
```

Within the training dataset in KITTI, there are almost 7481 point cloud files are present, we are only interested in few and therefore selecting 53 point cloud files and then searching for the training datasets using the list of selected image files.

```
imageFileList  = os.listdir(r'C:\Users\shyam\OneDrive\Pictures\KITTI')
print(len(imageFileList))
```

```
53
```

```
folderPath = r'D:\3rd Semester\Machine Vision\Project\training\data_object_velodyne\training\velodyne'
destination = r'C:\Users\shyam\OneDrive\Pictures\KITTIBIN'

KittiDataset = os.listdir(folderPath)

for filename in KittiDataset:
    for binfiles in imageFileList:
        binfiles = os.path.splitext(binfiles)[0]+".bin"
        file = os.path.join(folderPath,binfiles)
        if binfiles in filename:
            print(file)
            shutil.copy(file,destination)
        else:
             pass
```

Figure 2: code for selecting particular bin files from velodyne point clouds

Before processing the point cloud file which was stored in binary format, it has to be converted to the point cloud format as pcd , we use open3d library to convert from bin to pcd file.

```
def generate_pcd(file_to_open,file_to_save):
    size_float = 4
    list_pcd = []
    with open (file_to_open, "rb") as f:
        byte = f.read(size_float*4)
        while byte:
            x,y,z,intensity = struct.unpack("ffff", byte)
            list_pcd.append([x, y, z])
            byte = f.read(size_float*4)
    np_pcd = np.asarray(list_pcd)
    pcd = open3d.geometry.PointCloud()
    v3d = open3d.utility.Vector3dVector
    pcd.points = v3d(np_pcd)
    open3d.io.write_point_cloud(file_to_save, pcd)
```

Figure 3: Code for converting pcd to bin.

Below is the visualization of the point cloud generated from the image:

```
pcd = open3d.io.read_point_cloud(r'C:\Users\shyam\OneDrive\Pictures\KITTIPCD\000339.pcd')

open3d.visualization.draw_geometries([pcd])
```

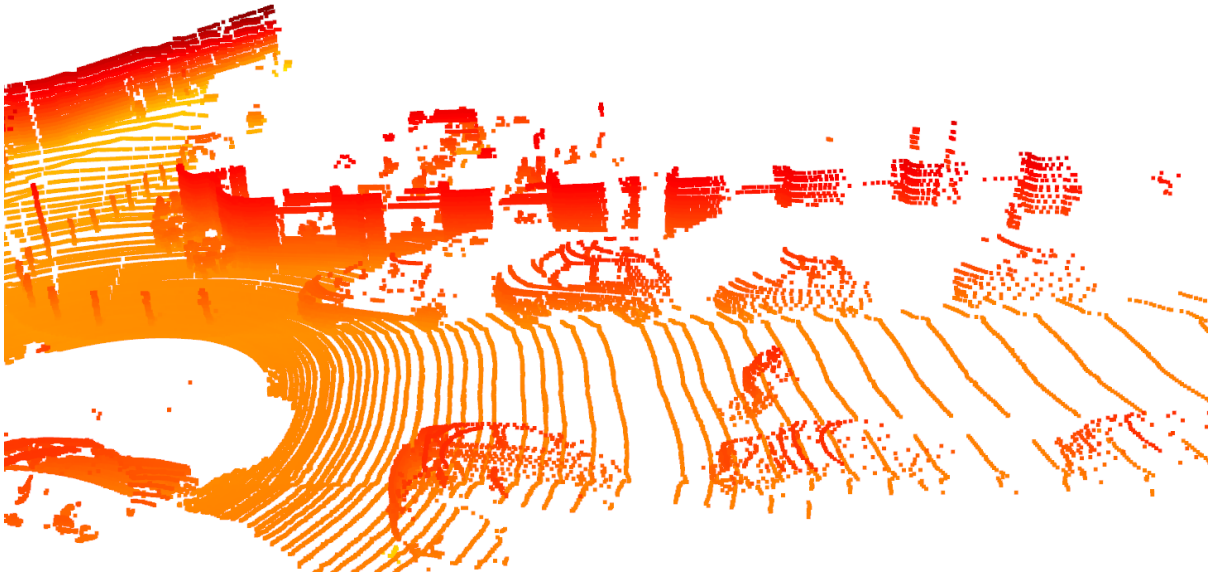

Figure 4: Visualization of point cloud

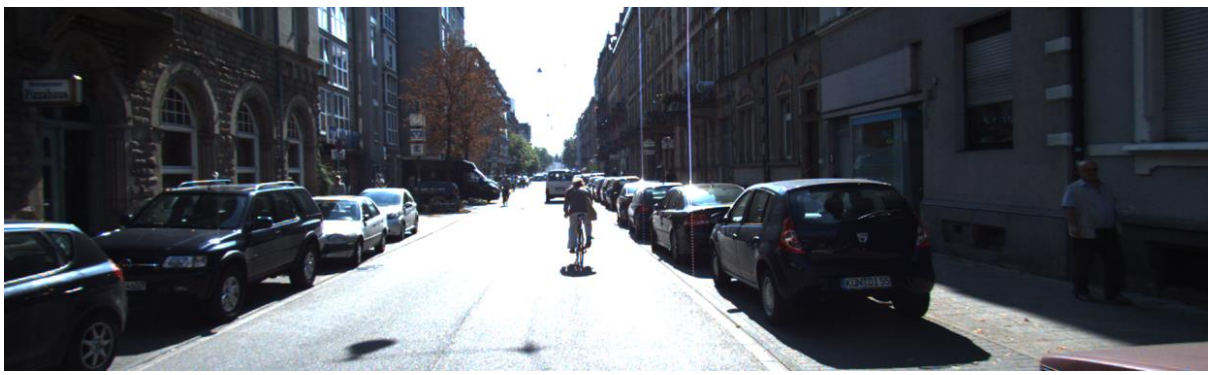

Figure 5: Corresponding image of point cloud

VOXEL DOWNSAMPLING

The point cloud has more points and will take more computation time, to reduce the computation cost we have to downsample the point clouds. Voxel downsampling is such a method, which reduces the point cloud by taking the points inside the cube,known as voxel. There will be one point as a result of mean of points taken in the point cloud.

By using voxel_down_sample in open3d, we can do downsampling, in our case we are selecting the size as 5 percentage of voxel grid, because increasing the voxel size, will reduce the points significantly

```
#Step 2 : Voxel Downsampling:
print('before downsampling:' , len(pcd.points))
downsample = pcd.voxel_down_sample(voxel_size=0.05)
print('After Voxel downsampling :', len(downsample.points))
# open3d.visualization.draw_geometries([downsample])
```

```
before downsampling: 115777
After Voxel downsampling : 77118
```

Figure 6: Code for voxel downsampling

RANSAC

As a next step, we have to do segmentation by using RANSAC which is robust when there are more outliers. In our point cloud, because more number of points lines in the ground these are called as inliers and the cars, bicyclist,pedestrians are considered as outliers.

We have to select the number of points which is 3 because of plane and the distance between the plane and closest point is 0.3 and number of iterations as 250. From segment_plane method in open3d we segment the inliers(red color) and outliers(blue color).

```
#Segmentation : RANSAC
plane_model, inliers = downsample.segment_plane(distance_threshold=0.3,
                                        ransac_n=3,
                                        num_iterations=250)
inlier_cloud=downsample.select_by_index(inliers)
outlier_cloud=downsample.select_by_index(inliers,invert=True)
inlier_cloud.paint_uniform_color([1,0,0])#Inliers - Red
outlier_cloud.paint_uniform_color([0,0,1])# Outliers - Blue
open3d.visualization.draw_geometries([inlier_cloud,outlier_cloud])
```

Figure 7: RANSAC implementation code

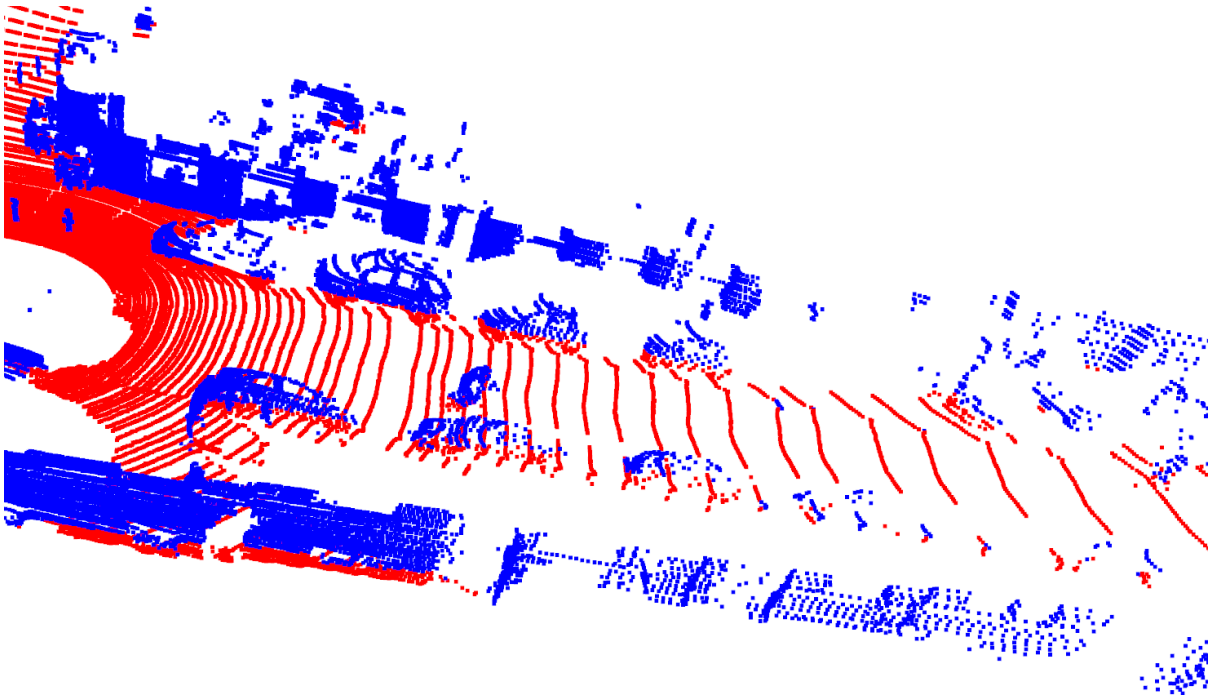

Figure 8: Segmentation of point cloud

DBSCAN

Once, we segmented the point cloud, we have to form the clusters within point cloud. K-Means its not a suitable option because of sensitive to scale, outliers and noisy data. Hence we select the DBSCAN which needs epsilon and minimum of points as parameters. The algorithm forms the cluster, when the points is greater than or equal to minimum number of points passed as parameter.

DBSCAN clustering is implemented in open3d with cluster_dbscan method and we pass 0.5 as epsilon value and 5 points as minimum points

```
# Clustering - DBScan
with open3d.utility.VerbosityContextManager(open3d.utility.VerbosityLevel.Debug) as cm:
    labels = np.array(outlier_cloud.cluster_dbscan(eps=0.5,min_points=5,print_progress=True))
max_label = labels.max()
print(f"point cloud has {max_label + 1} clusters")
colors = plt.get_cmap("tab20")(labels/(max_label if max_label > 0 else 1))
colors[labels < 0] = 0
outlier_cloud.colors = open3d.utility.Vector3dVector(colors[:,:3])
open3d.visualization.draw_geometries([inlier_cloud,outlier_cloud])
```

```
[Open3D DEBUG] Precompute neighbors.
[Open3D DEBUG] Done Precompute neighbors.
[Open3D DEBUG] Compute Clusters
[Open3D DEBUG] Done Compute Clusters: 252
point cloud has 252 clusters
```

Figure 9: code for DBSCAN clustering

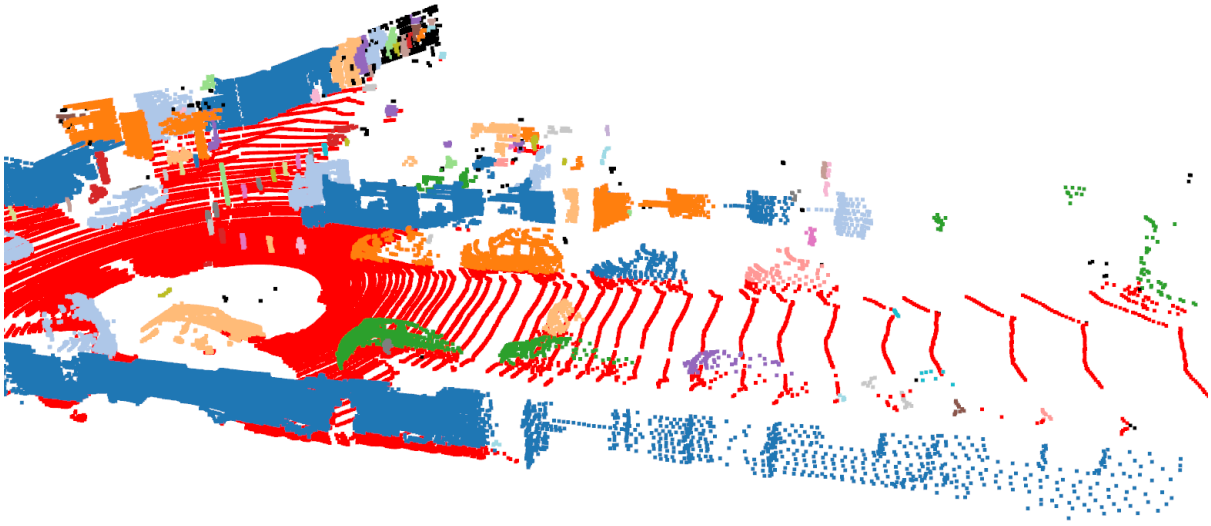

Figure 10: clusters resulted from DBSCAN

3D BOUNDING BOX

The clusters generated from DBSCAN returns indexes when calling the cluster_dbscan method.We, have to converts this into list and perfrom bounding boxes on each clusters, and minimum and max points were specified to select for bounding box formation . Minimum points is given as 500 and maximum points is 20 and get_axis_aligned_bounding_box() produces the bounding box.

In the below code, we check for points within each cluster using the index and if it lies between the specified range, the bounding box is created then

```
Max_points = 500
Min_points =  20
clusters = []
for i in range(0,len(indexes)):
    nb_pts = len(outlier_cloud.select_by_index(indexes[i]).points)
    if(nb_pts > Min_points and nb_pts < Max_points):
        sub_cloud = outlier_cloud.select_by_index(indexes[i])
        obb = sub_cloud.get_axis_aligned_bounding_box()
        obb.color=(0,0,1)
        obbs.append(obb)
```

Figure 11: code for 3d bounding box implementation

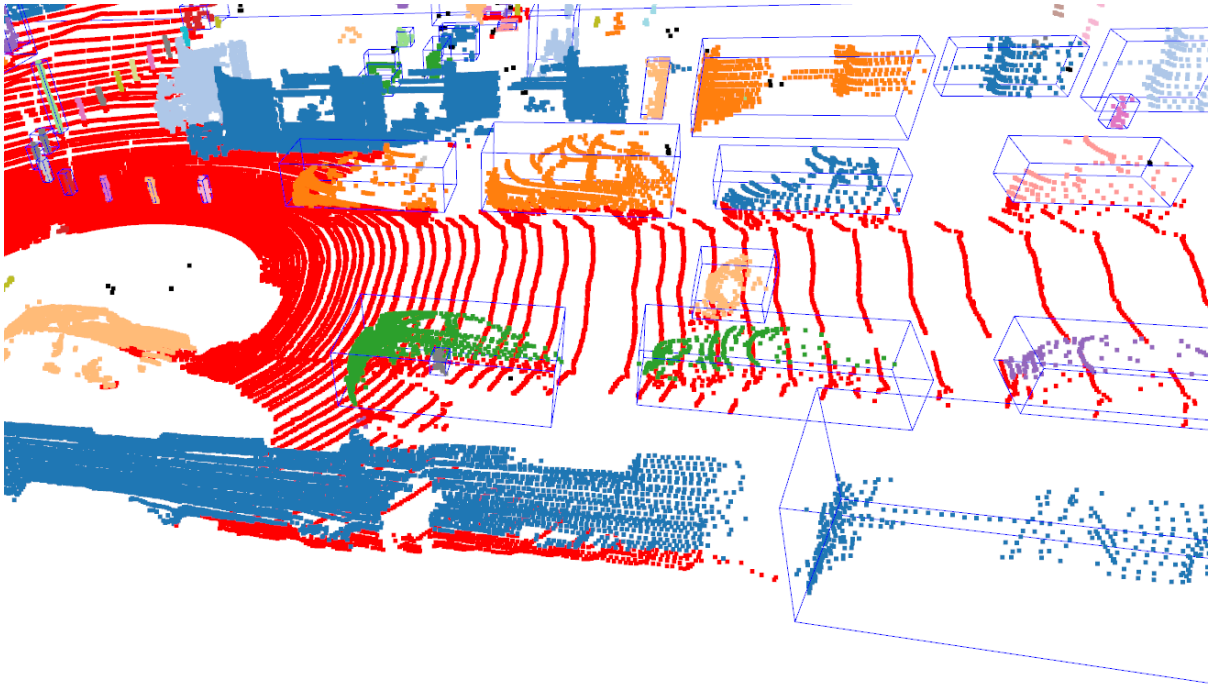

Figure 12: Visualization of 3D bounding box

VISUALIZATION

For each point, the visualization of 3d point clouds are shown over in below figures

Figure 13: Equivalent 1st test image of point cloud for 3D object detection

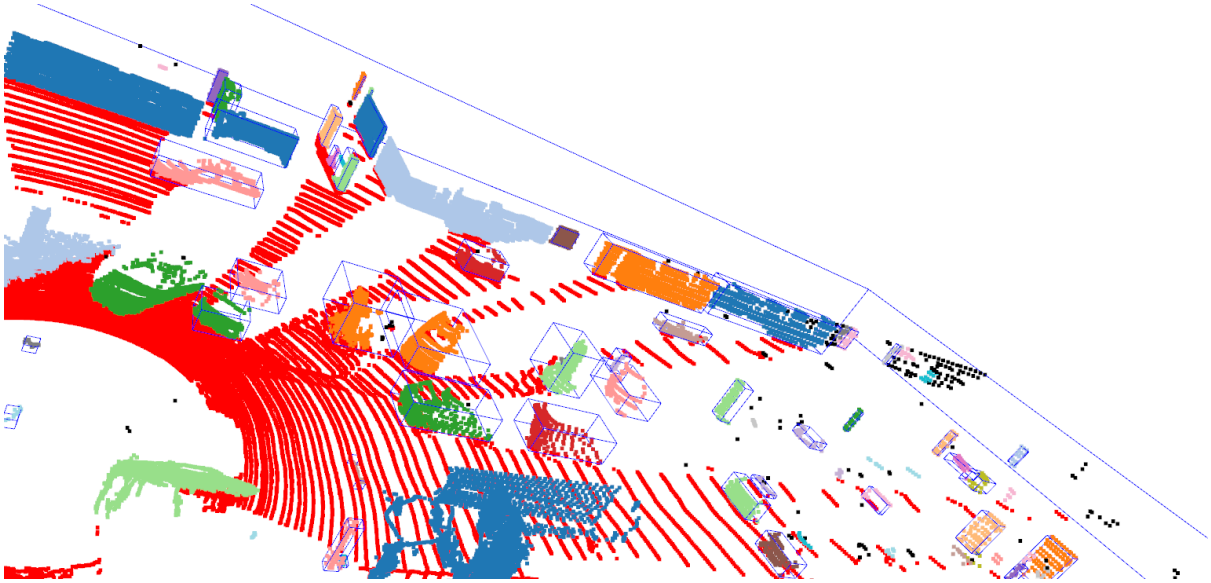

Figure 14: 3D bounding visualization of 1st test point cloud

Figure 15: Equivalent 2nd test image of point cloud for 3D object detection

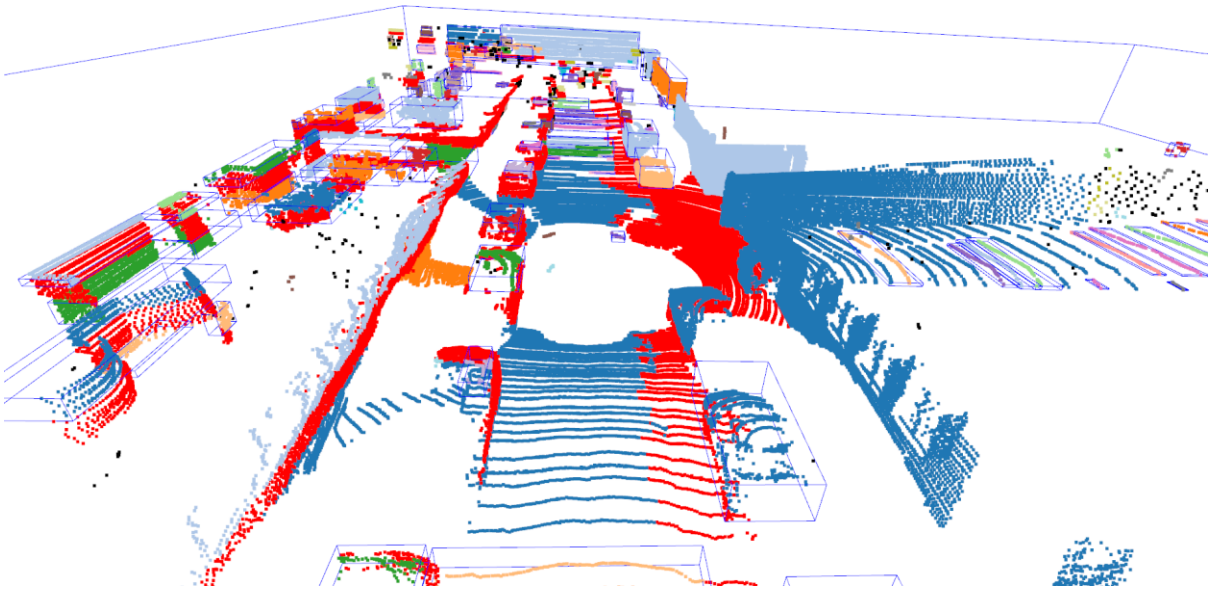

Figure 16: 3D bounding box visualization of 2nd point cloud

CONCLUSION

Using unsupervised learning, we can able to perform 3d object detection, however this method produces more false positives. Nevertheless, this method can be a quick solution for developing 3d object detection algorithm and served as baseline, and these algorithms doesn't need the GPU resources.

As an alternate solution, we can use deep learning-based method to provide better accuracy and increase the average precision value. There are architectures like point pillar, VoxelNet, complex yolo, point net and many others that improvise the accuracy and inference speed of point clouds.

REFERENCES

1. Mao, Jiageng, Shi, Shaoshuai, Wang, Xiaogang, and Hongsheng Li. "3D Object Detection for Autonomous Driving: A Review and New Outlooks." *arXiv*, (2022). Accessed November 15, 2022. https://doi.org/10.48550/arXiv.2206.09474.
2. Ghasemieh, Alireza, and Rasha Kashef. "3D object detection for autonomous driving: Methods, models, sensors, data, and challenges." *Transportation Engineering 8*, (2022): 100115. Accessed November 15, 2022. https://doi.org/10.1016/j.treng.2022.100115.
3. KITTI dataset https://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d
4. Jeremy Cohen " think Autonomous AI course"