This is a template submission for the midterm course : 3D Object Detection (Midterm).

3D Object detection

We have used the Waymo dataset real-world data and used 3d point cloud for lidar based object detection.

- Configuring the ranges channel to 8 bit and view the range /intensity image (ID_S1_EX1)
- Use the Open3D library to display the lidar point cloud on a 3d viewer and identifying 6 images from point cloud.(ID_S1_EX2)
- Create Birds Eye View perspective (BEV) of the point cloud,assign lidar intensity values to BEV,normalize the heightmap of each BEV (ID_S2_EX1,ID_S2_EX2,ID_S2_EX3)
- In addition to YOLO, use the repository [https://github.com/maudzung/SFA3D.git]and add parameters ,instantiate fpn resnet model(ID_S3_EX1)
- Convert BEV coordinates into pixel coordinates and convert model output to bounding box format (ID_S3_EX2)
- Compute intersection over union, assign detected objects to label if IOU exceeds threshold (ID_S4_EX1)
- Compute false positives and false negatives, precision and recall(ID_S4_EX2,ID_S4_EX3)

The project can be run by running

python loop_over_dataset.py

All training/inference is done on udacity workspace.

Step-1: Compute Lidar point cloud from Range Image

In this we are first previewing the range image and convert range and intensity channels to 8 bit format. After that, we use the openCV library to stack the range and intensity channel vertically to visualize the image.

- Convert "range" channel to 8 bit
- Convert "intensity" channel to 8 bit
- Crop range image to +/- 90 degrees left and right of forward facing x axis
- Stack up range and intensity channels vertically in openCV

The changes are made in 'loop_over_dataset.py'

```
49   ## Set parameters and perform initializations
50
51   ## Select Waymo Open Dataset file and frame numbers
52   # data_filename = 'training_segment-1005081002024129653_5313_150_5333_150_with_camera_labels.tfrecord' # Sequence 1
53   # data_filename = 'training_segment-10072231702153043603_5725_000_5745_000_with_camera_labels.tfrecord' # Sequence 2
54   data_filename = 'training_segment-10963653239323173269_1924_000_1944_000_with_camera_labels.tfrecord' # Sequence 3
55   show_only_frames = [1,10] # show only frames in interval for debugging
56
57   ## Prepare Waymo Open Dataset file for loading
58   data_fullpath = os.path.join(os.path.dirname(os.path.realpath(__file__)), 'dataset', data_filename) # adjustable path
     called from another working directory
59   model = "darknet"
60   sequence = "3"
61   results_fullpath = os.path.join(os.path.dirname(os.path.realpath(__file__)), 'results/' + model + '/results_sequence_'
     model)
```

```
80  np.random.seed(10) # make random values predictable
81
82  ## Selective execution and visualization
83  exe_data=[]  #option 'pcl_from rangeimage','load_image'
84  exec_detection = [] # options are 'bev_from_pcl', 'detect_objects', 'validate_object_labels', 'measure_detection_performance'; options not in
    the list will be loaded from file
85  exec_tracking = [] # options are 'perform_tracking'
86  exec_visualization = ['show_range_image'] # options are 'show_range_image', 'show_bev', 'show_pcl', 'show_labels_in_image',
    'show_objects_and_labels_in_bev', 'show_objects_in_bev_labels_in_camera', 'show_tracks', 'show_detection_performance', 'make_tracking_movie'
87  exec_list = make_exec_list(exec_detection, exec_tracking, exec_visualization)
88  vis_pause_time = 0 # set pause time between frames in ms (0 = stop between frames until key is pressed)
89
90
```

The changes are made in "objdet_pcl.py"


```
74   # visualize range image
75 ▾ def show_range_image(frame, lidar_name):
76
77       ####### ID_S1_EX1 START #######
78       #######
79       print("student task ID_S1_EX1")
80
81       # step 1 : extract lidar data and range image for the roof-mounted lidar
82       lidar = [obj for obj in frame.lasers if obj.name == lidar_name][0]
83
84       # step 2 : extract the range and the intensity channel from the range image
85       ri = dataset_pb2.MatrixFloat()
86       ri.ParseFromString(zlib.decompress(lidar.ri_return1.range_image_compressed))
87       ri = np.array(ri.data).reshape(ri.shape.dims)
88
89       # step 3 : set values <0 to zero
90       ri[ri<0]=0.0
91
92       # step 4 : map the range channel onto an 8-bit scale and make sure that the full range of values is appropriately considered
93       ri_range = ri[:,:,0]
94       ri_range = ri_range * 255 / (np.amax(ri_range) - np.amin(ri_range))
95       img_range = ri_range.astype(np.uint8)
96
97       # step 5 : map the intensity channel onto an 8-bit scale and normalize with the difference between the 1- and 99-percentile to mi...
    outliers
98       ri_intensity = ri[:,:,1]
99       percentile_1, percentile_99 = percentile(ri_intensity,1), percentile(ri_intensity,99)
100      ri_intensity = 255 * np.clip(ri_intensity,percentile_1,percentile_99)/percentile_99
101      img_intensity = ri_intensity.astype(np.uint8)
102
103      # step 6 : stack the range and intensity image vertically using np.vstack and convert the result to an unsigned 8-bit integer
104      img_range_intensity = np.vstack((img_range,img_intensity))
105      img_range_intensity = img_range_intensity.astype(np.uint8)
```

The range image sample:
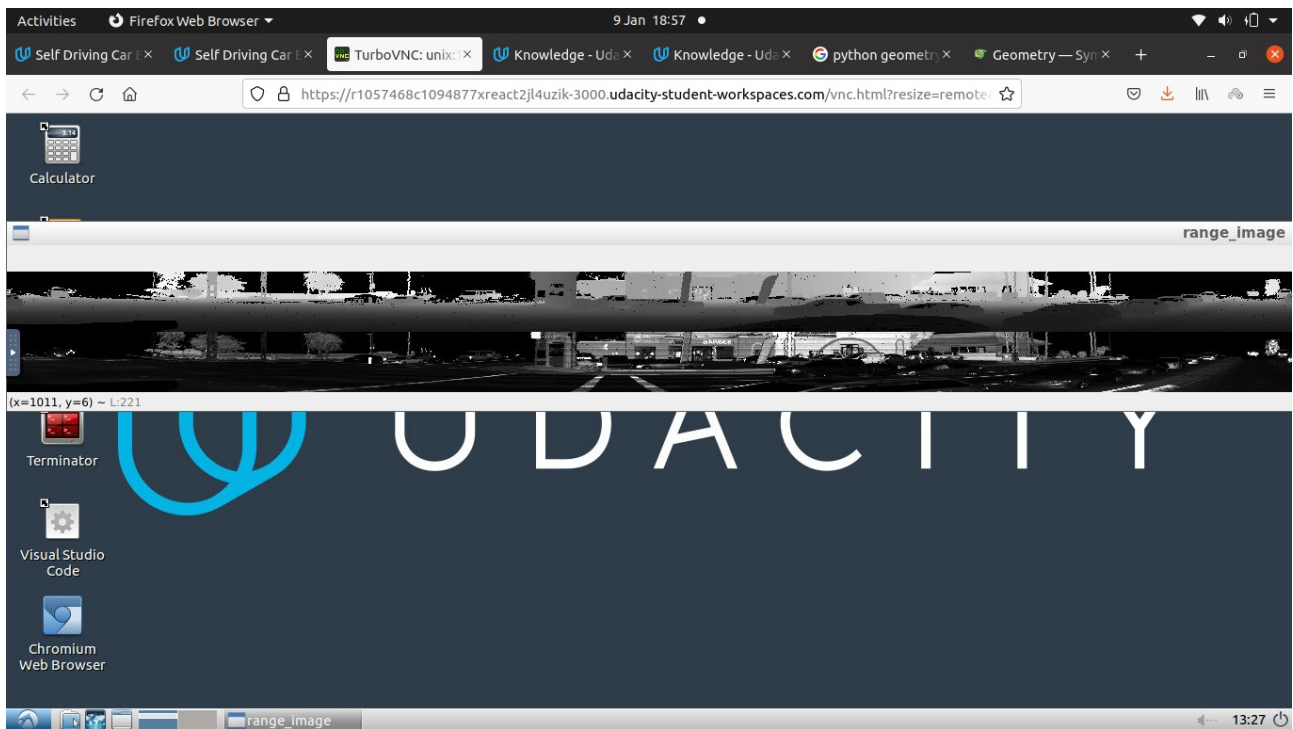
For the next part, we use the Open3D library to display the lidar point cloud on a 3D viewer and identify 10 images from point cloud

- Visualize the point cloud in Open3D
- 10 examples from point cloud with varying degrees of visibility

The changes are made in 'loop_over_dataset.py'

The changes are made in "objdet_pcl.py"

```
36  # visualize lidar point-cloud
37 ▾ def show_pcl(pcl):
38
39      ####### ID_S1_EX2 START #######
40      #######
41      print("student task ID_S1_EX2")
42
43      # step 1 : initialize open3d with key callback and create window
44      viz = o3d.visualization.VisualizerWithKeyCallback()
45      viz.create_window(window_name='Open3D', width=1280, height=1080, left=50, top=50, visible=True)
46
47      global ptr
48      ptr= True
49 ▾    def Window(viz):
50          global ptr
51          print('right arrow pressed')
52          ptr= False
53          return
54      viz.register_key_callback(262,Window)
55
56      # step 2 : create instance of open3d point-cloud class
57      pcd = o3d.geometry.PointCloud()
58
59      # step 3 : set points in pcd instance by converting the point-cloud into 3d vectors (using open3d function Vector3dVector)
60      pcd.points = o3d.utility.Vector3dVector(pcl[:,:3])
61
62      # step 4 : for the first frame, add the pcd instance to visualization using add_geometry; for all other frames, use update_geomet
63      viz.add_geometry(pcd)
64
65      # step 5 : visualize point cloud and keep window open until right-arrow is pressed (key-code 262)
66 ▾    while ptr:
67          viz.poll_events()
68          viz.update_renderer()
```
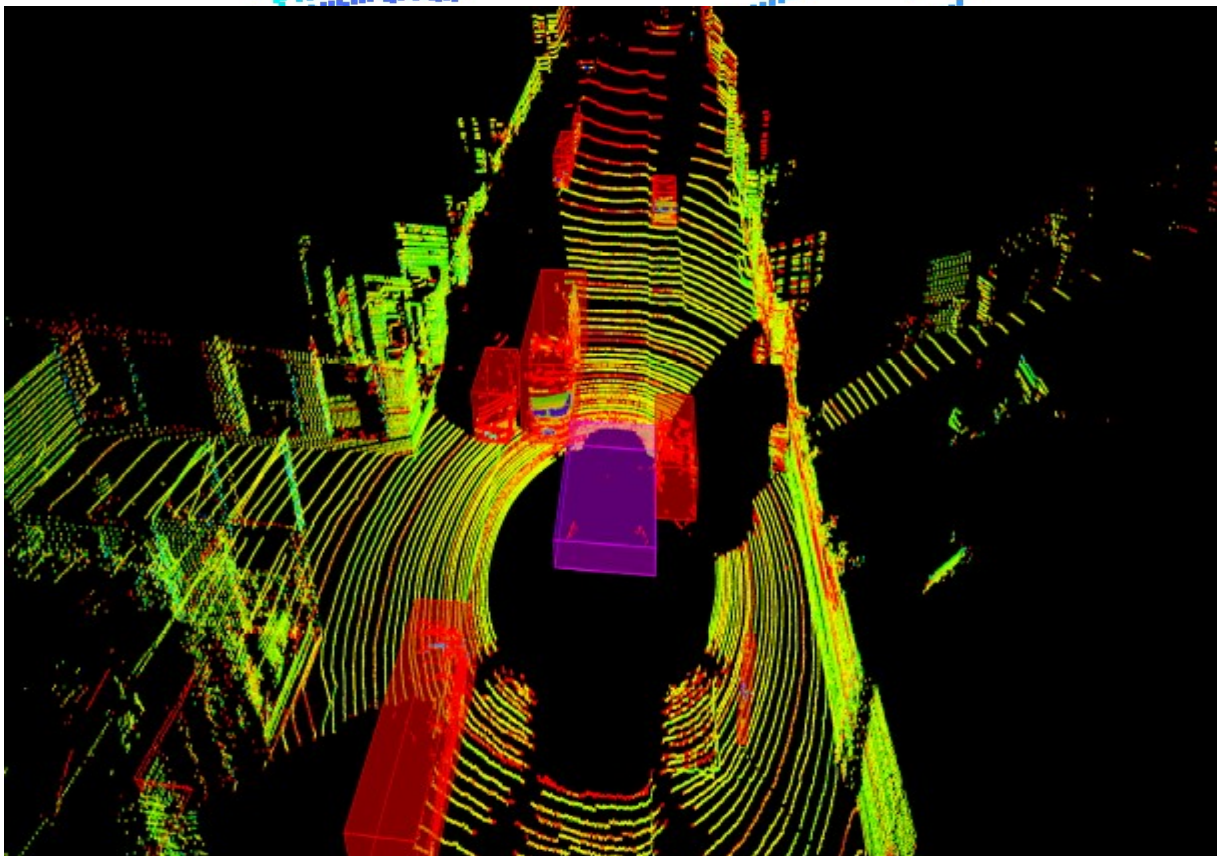
Point cloud images

Open3D

Bonnet

Wind Screen

Wheel

Open3D

Wind screen

Back of car

wheel
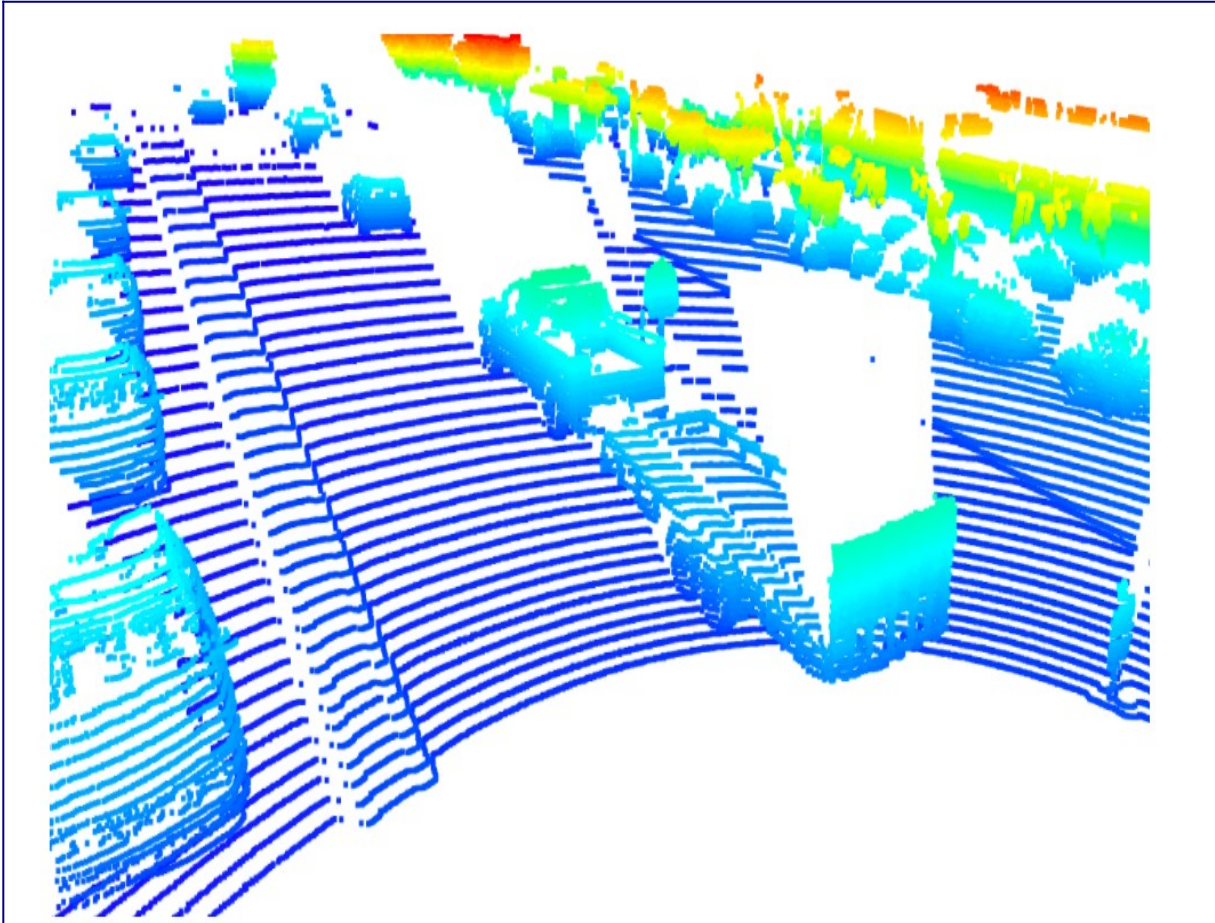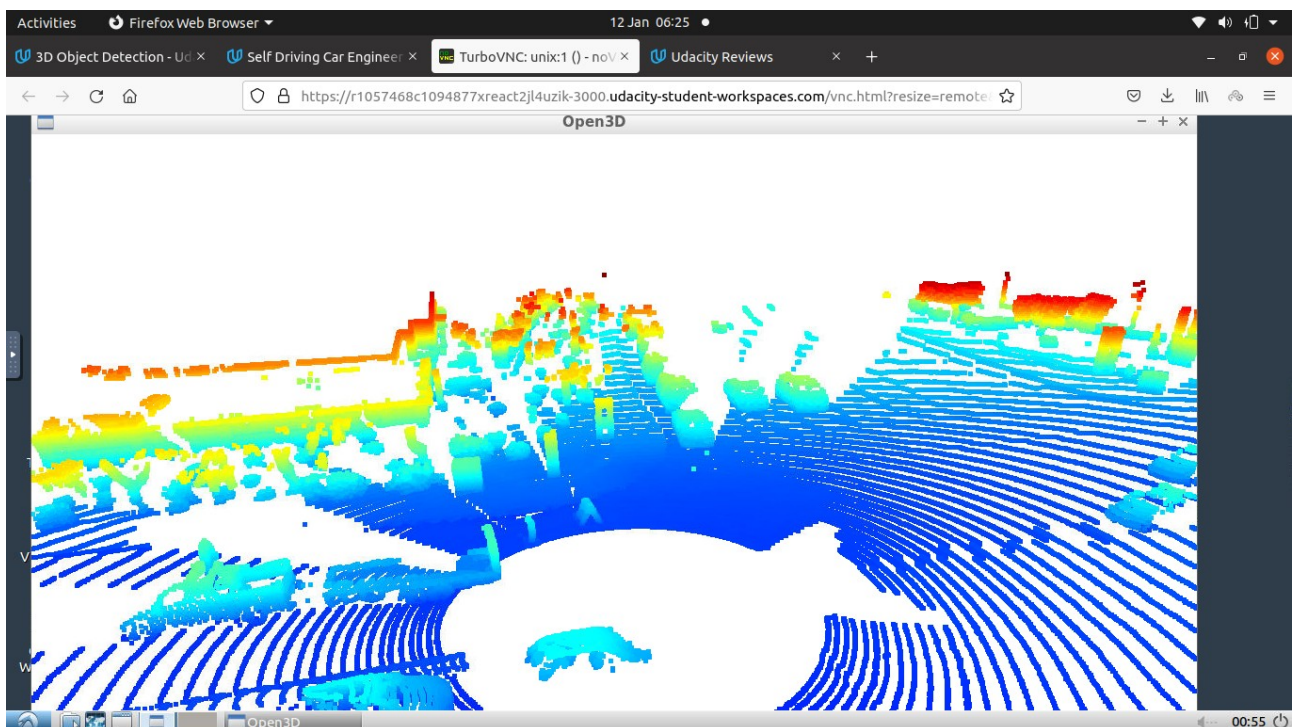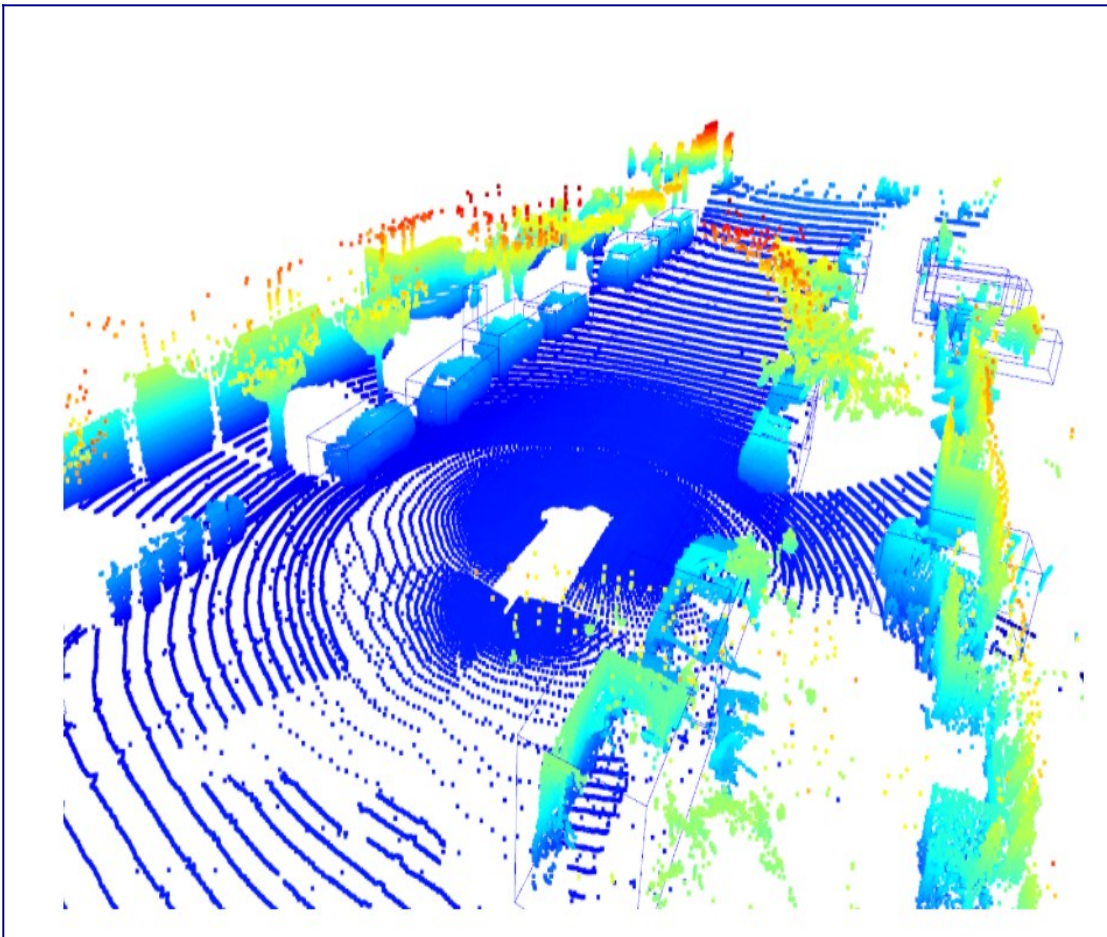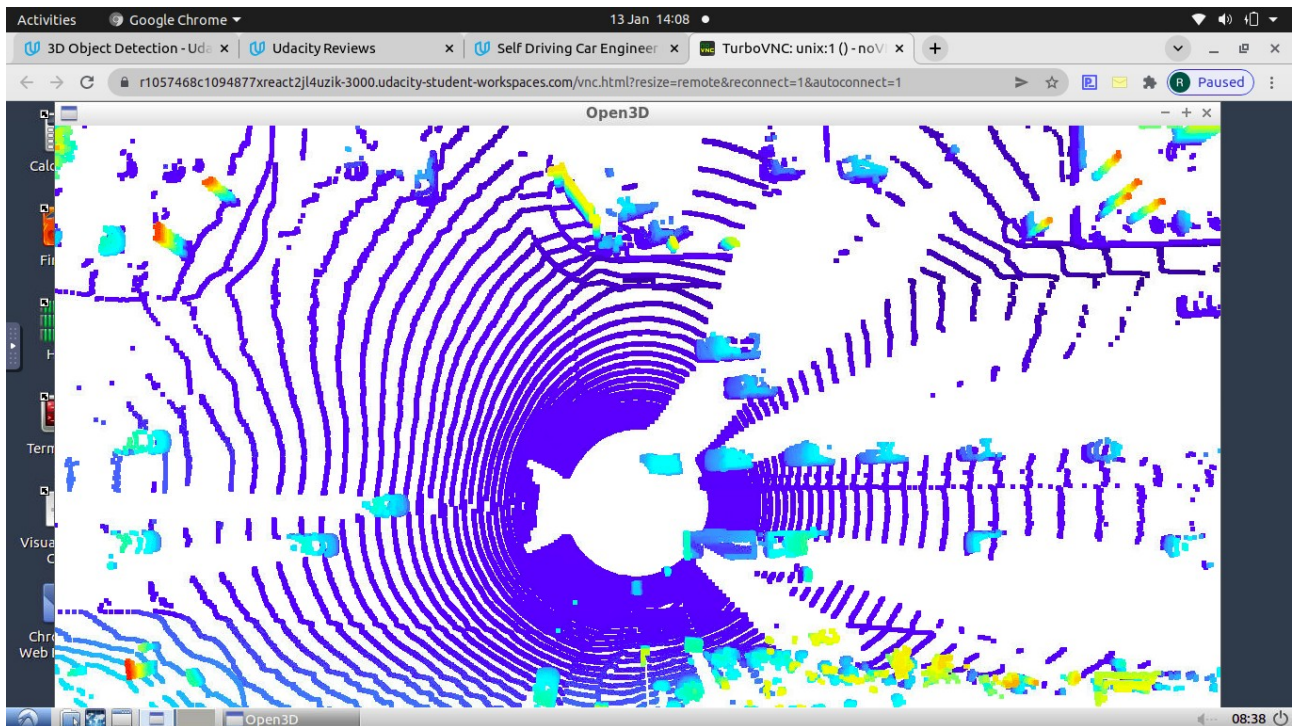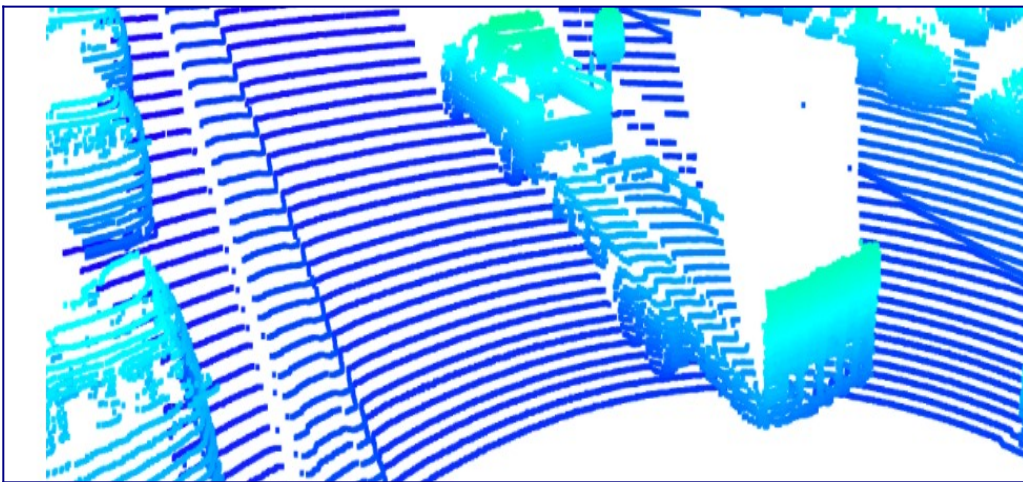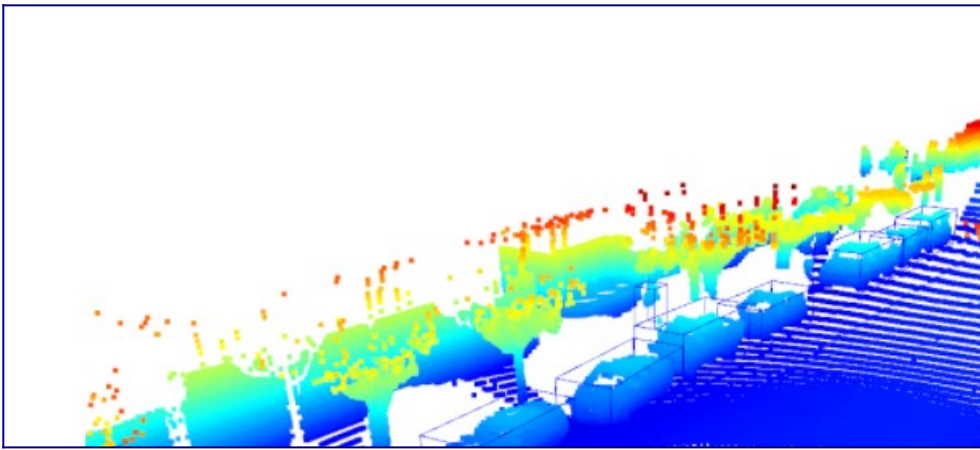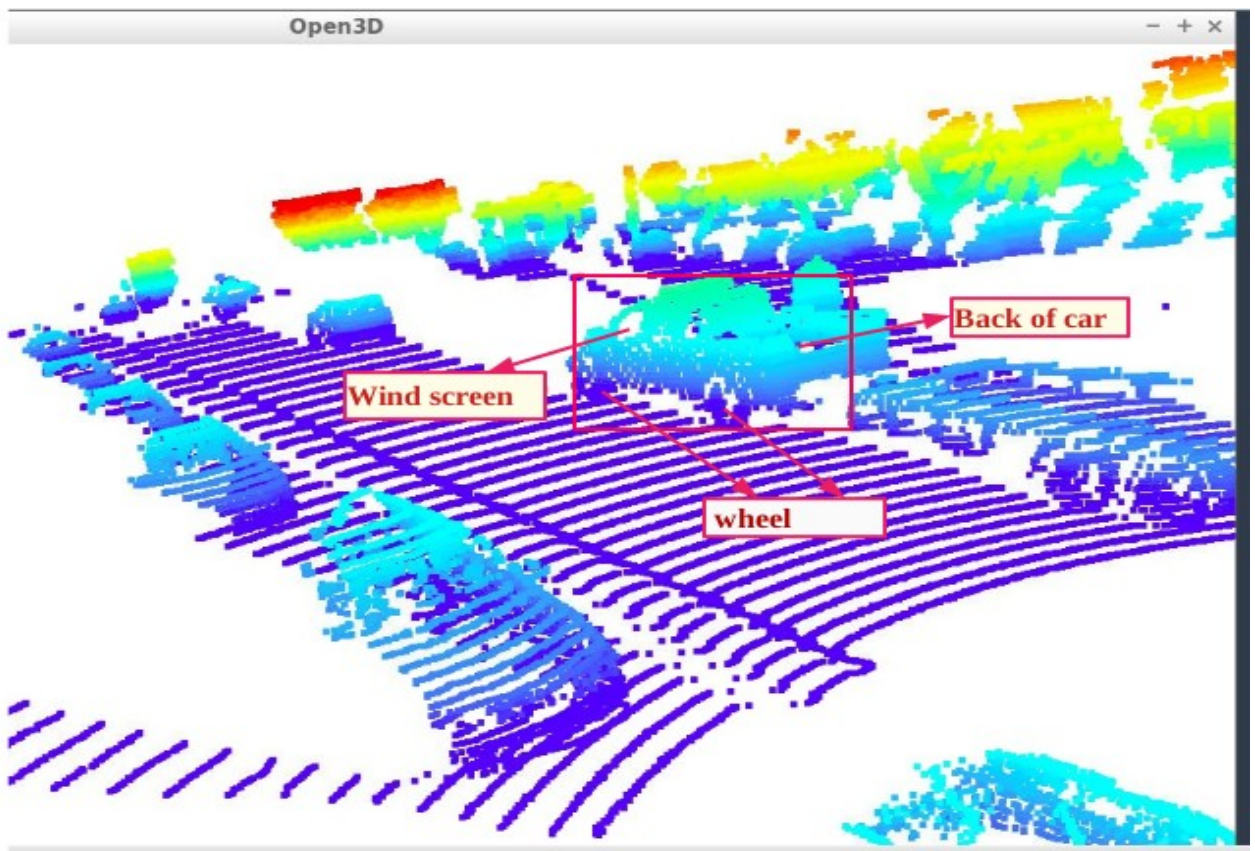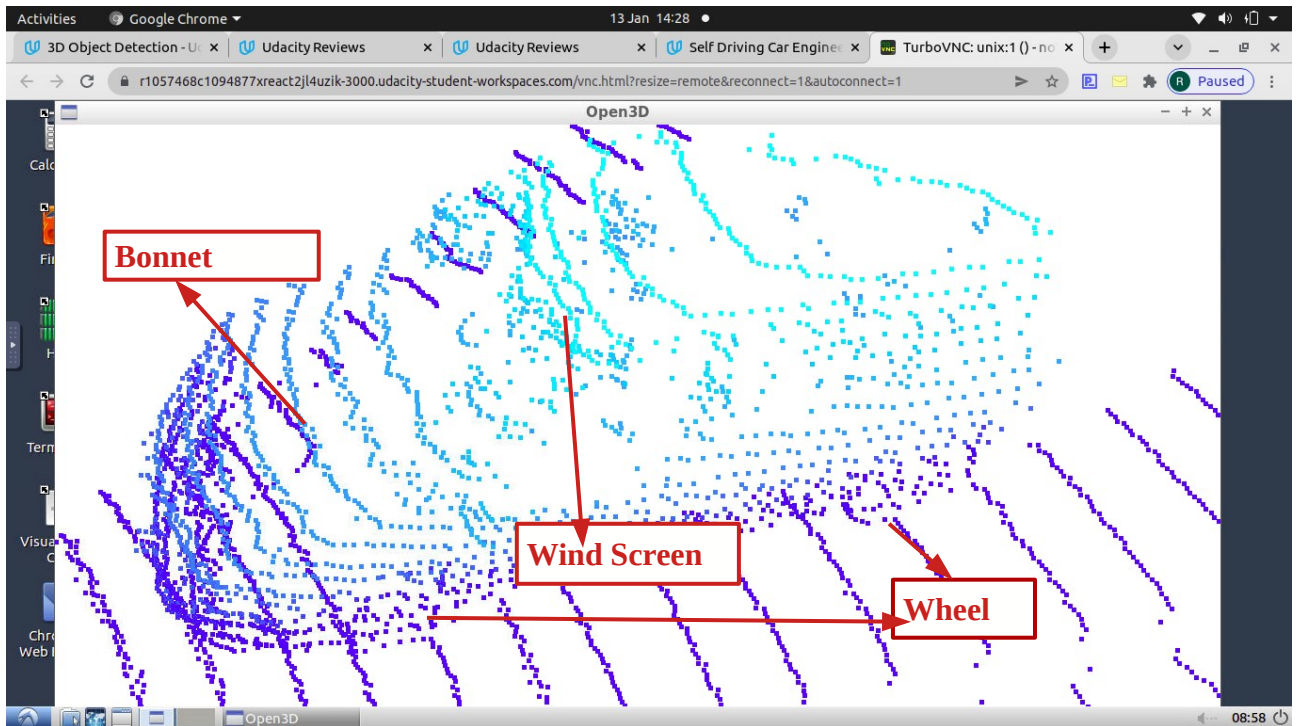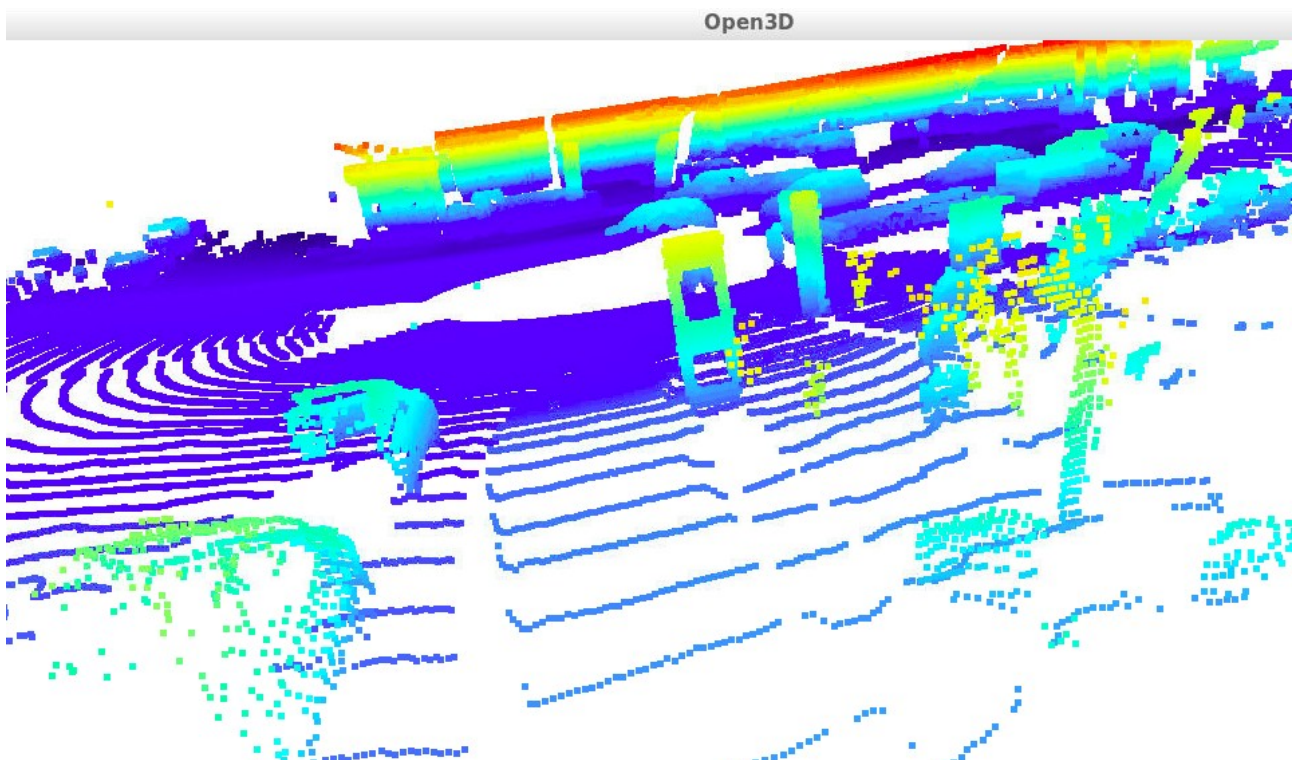
Stable features include the tail lights, the rear bumper majorly. In some cases the additional features include the headover lights, car front lights, rear window shields. These are identified through the intensity channels . The chassis of the car is the most prominent identifiable feature from the lidar

perspective. The images are analysed with different settings and the rear lights are the major stable components, also the bounding boxes are correctly assigned to the cars (used from Step-3).

Step-2: Creaate BEV from Lidar PCL

In this case, we are:

- Converting the coordinates to pixel values
- Assigning lidar intensity values to the birds eye view BEV mapping
- Using sorted and pruned point cloud lidar from the previous task
- Normalizing the height map in the BEV
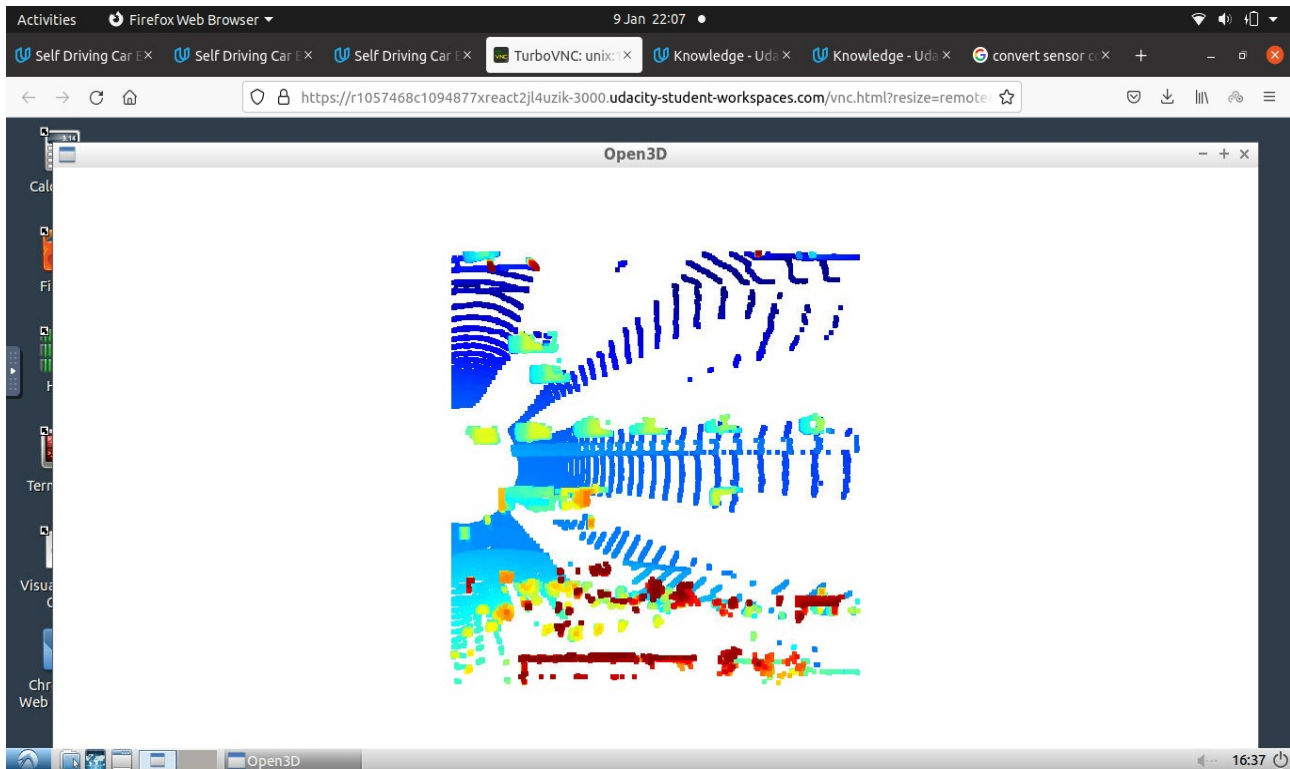- Compute and map the intensity values

The changes are in the 'loop_over_dataset.py'

```
82   ## Selective execution and visualization
83   exe_data=['pcl_from rangeimage']  #option 'pcl_from rangeimage','load_image'
84   exec_detection = ['bev_from_pcl'] # options are 'bev_from_pcl', 'detect_objects', 'validate_object_labels', 'mea
     loaded from file
85   exec_tracking = [] # options are 'perform_tracking'
86   exec_visualization = [] # options are 'show_range_image', 'show_bev', 'show_pcl', 'show_labels_in_image', 'show_
     'show_objects_in_bev_labels_in_camera', 'show_tracks', 'show_detection_performance', 'make_tracking_movie'
87   exec_list = make_exec_list(exec_detection, exec_tracking, exec_visualization)
88   vis_pause_time = 0 # set pause time between frames in ms (0 = stop between frames until key is pressed)
89
```

The changes are also in the "objdet_pcl.py"

```
                                    3D Object Detection Workspace
     README.md          X     loop_over_datase...    X     objdet_pcl.py           X
121
122    # create birds-eye view of lidar data
123  ▾ def bev_from_pcl(lidar_pcl, configs):
124
125        # remove lidar points outside detection area and with too low reflectivity
126  ▾     mask = np.where((lidar_pcl[:, 0] >= configs.lim_x[0]) & (lidar_pcl[:, 0] <= configs.lim_x[1]) &
127                        (lidar_pcl[:, 1] >= configs.lim_y[0]) & (lidar_pcl[:, 1] <= configs.lim_y[1]) &
128                        (lidar_pcl[:, 2] >= configs.lim_z[0]) & (lidar_pcl[:, 2] <= configs.lim_z[1]))
129        lidar_pcl = lidar_pcl[mask]
130
131        # shift level of ground plane to avoid flipping from 0 to 255 for neighboring pixels
132        lidar_pcl[:, 2] = lidar_pcl[:, 2] - configs.lim_z[0]
133
134        # convert sensor coordinates to bev-map coordinates (center is bottom-middle)
135        ####### ID_S2_EX1 START #######
136        #######
137        print("student task ID_S2_EX1")
138
139
140        ## step 1 :  compute bev-map discretization by dividing x-range by the bev-image height (see configs)
141        bev_discret = (configs.lim_x[1] - configs.lim_x[0]) / configs.bev_height
142
143
144        ## step 2 : create a copy of the lidar pcl and transform all metrix x-coordinates into bev-image coordinates
145        copy_lidar_pcl = np.copy(lidar_pcl)
146        copy_lidar_pcl[:, 0] = np.int_(np.floor(copy_lidar_pcl[:, 0] / bev_discret))
147
148        # step 3 : perform the same operation as in step 2 for the y-coordinates but make sure that no negative bev-coordinates occur
149        copy_lidar_pcl[:, 1] = np.int_(np.floor(copy_lidar_pcl[:, 1] / bev_discret) + (configs.bev_width + 1) / 2)
150        copy_lidar_pcl[:, 1] = np.abs(copy_lidar_pcl[:,1])
151
152        # step 4 : visualize point-cloud using the function show_pcl from a previous task
```
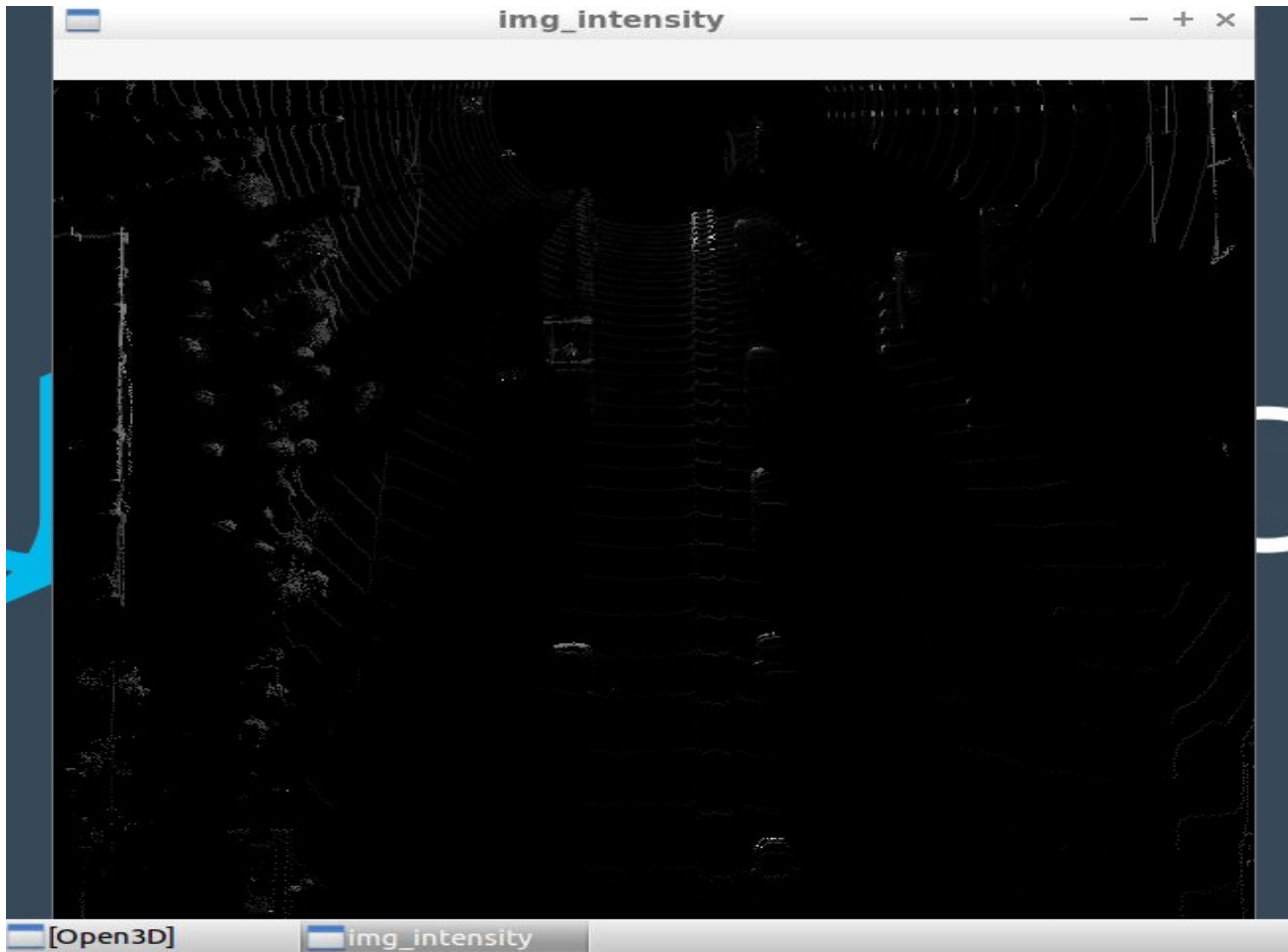
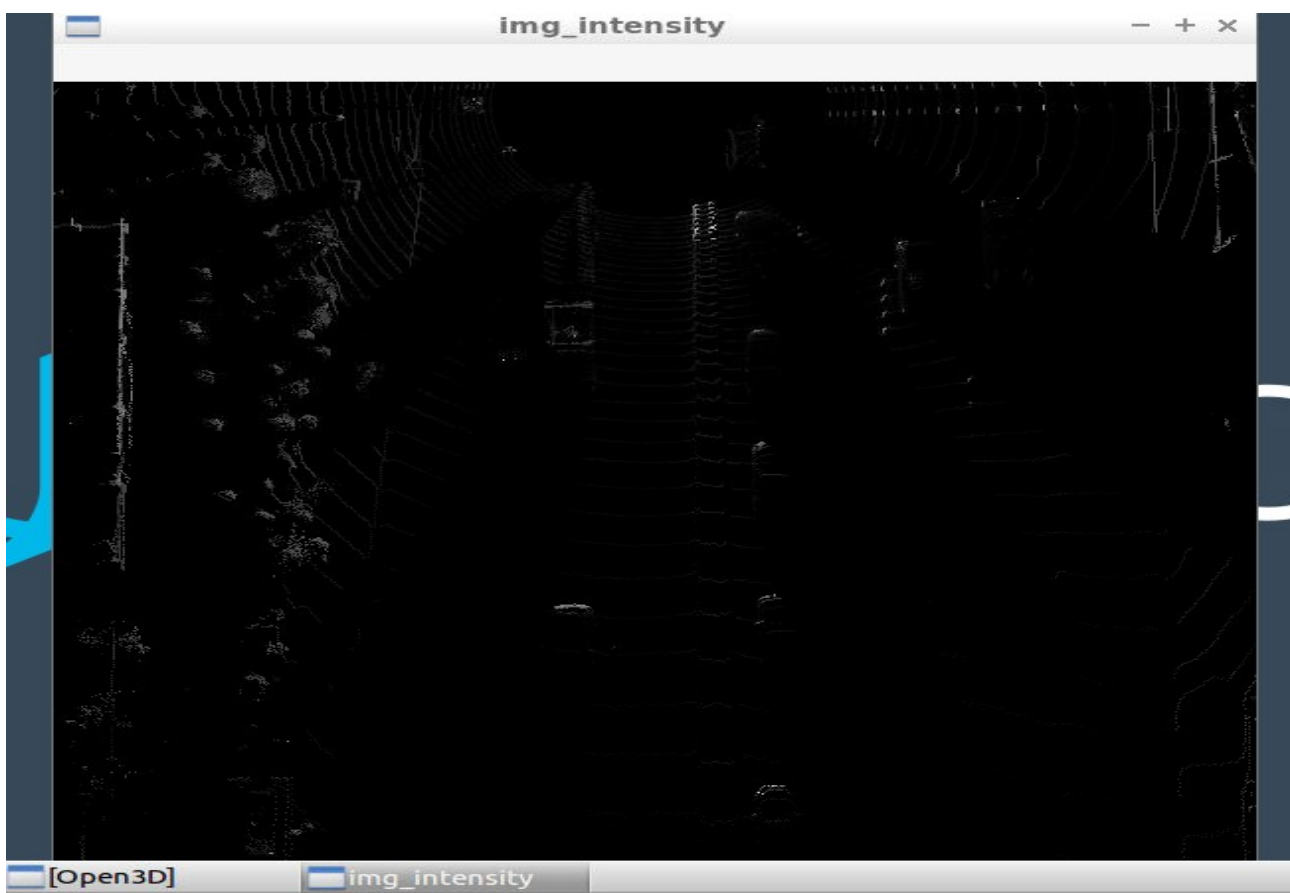A sample preview of the BEV:

A preview of the intensity layer:

The 'lidar_pcl_top' is used in this case, shown in the Figure:

The corresponding intensity channel:

The corresponding normalized height channel:
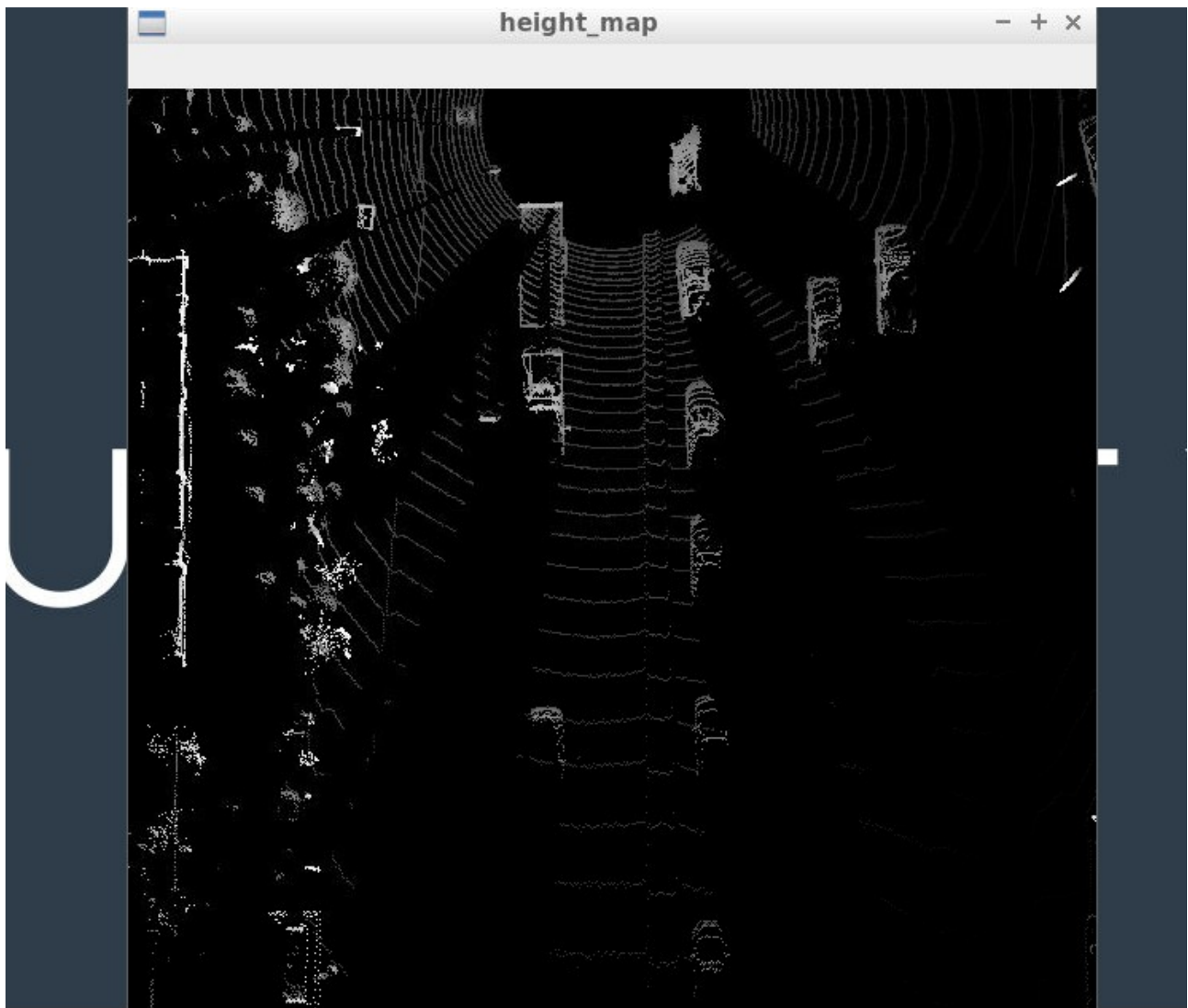
Step-3: Model Based Object Detection in BEV Image

Here, particularly the test.py file and extracting the relevant configurations from 'parse_test_configs()' and added them in the 'load_configs_model' config structure by looking into this repo [https://github.com/maudzung/SFA3D.git].

- Instantiating the fpn resnet model from the cloned repository configs
- Extracting 3d bounding boxes from the responses
- Transforming the pixel to vehicle coordinates
- Model output tuned to the bounding box format [class-id, x, y, z, h, w, l, yaw]

The changes are in "loop_over_dataset.py"

```
81
82  ## Selective execution and visualization
83  exe_data=['pcl_from rangeimage','load_image']  #option 'pcl_from rangeimage','load_image'
84  exec_detection = ['bev_from_pcl','detect_objects'] # options are 'bev_from_pcl', 'detect_objects', 'validate_object_labels', 'measure_det
    the list will be loaded from file
85  exec_tracking = [] # options are 'perform_tracking'
86  exec_visualization = ['show_objects_in_bev_labels_in_camera',] # options are 'show_range_image', 'show_bev', 'show_pcl', 'show_labels_in_
    'show_objects_and_labels_in_bev', 'show_objects_in_bev_labels_in_camera', 'show_tracks', 'show_detection_performance', 'make_tracking_mov
87  exec_list = make_exec_list(exec_detection, exec_tracking, exec_visualization)
88  vis_pause_time = 0 # set pause time between frames in ms (0 = stop between frames until key is pressed)
89
```

The changes for the detection are inside the "objdet_detect.py" file:

```
README.md          ×    loop_over_datase...    ×    objdet_detect.py    ×

181  # detect trained objects in birds-eye view
182 ▾ def detect_objects(input_bev_maps, model, configs):
183
184      # deactivate autograd engine during test to reduce memory usage and speed up computations
185 ▾    with torch.no_grad():
186
187          # perform inference
188          outputs = model(input_bev_maps)
189
190          # decode model output into target object format
191 ▾        if 'darknet' in configs.arch:
192
193              # perform post-processing
194              output_post = post_processing_v2(outputs, conf_thresh=configs.conf_thresh, nms_thresh=configs.nms_thresh)
195              detections = []
196 ▾            for sample_i in range(len(output_post)):
197 ▾                if output_post[sample_i] is None:
198                      continue
199                  detection = output_post[sample_i]
200 ▾                for obj in detection:
201                      x, y, w, l, im, re, _, _, _ = obj
202                      yaw = np.arctan2(im, re)
203                      detections.append([1, x, y, 0.0, 1.50, w, l, yaw])
204
205 ▾          elif 'fpn_resnet' in configs.arch:|
206              # decode output and perform post-processing
207
208              ####### ID_S3_EX1-5 START #######
209              #######
210              print("student task ID_S3_EX1-5")
211              outputs['hm_cen'] = _sigmoid(outputs['hm_cen'])
212              outputs['cen_offset'] = _sigmoid(outputs['cen_offset'])
213              # detections size (batch_size, K, 10)
214 ▾            detections = decode(outputs['hm_cen'], outputs['cen_offset'], outputs['direction'], outputs['z_coor'],
215                              outputs['dim'], K=40) #K=configs.k
216              detections = detections.cpu().numpy().astype(np.float32)
217              # print(detections)
218              detections = post_processing(detections, configs)
219              detections = detections[0][1]

    BASH
```

As the model input is a three-channel BEV map, the detected objects will be returned with coordinates and properties in the BEV coordinate space. Thus, before the detections can move along in the processing pipeline, they need to be converted into metric coordinates in vehicle space.

A sample preview of the bounding box images:
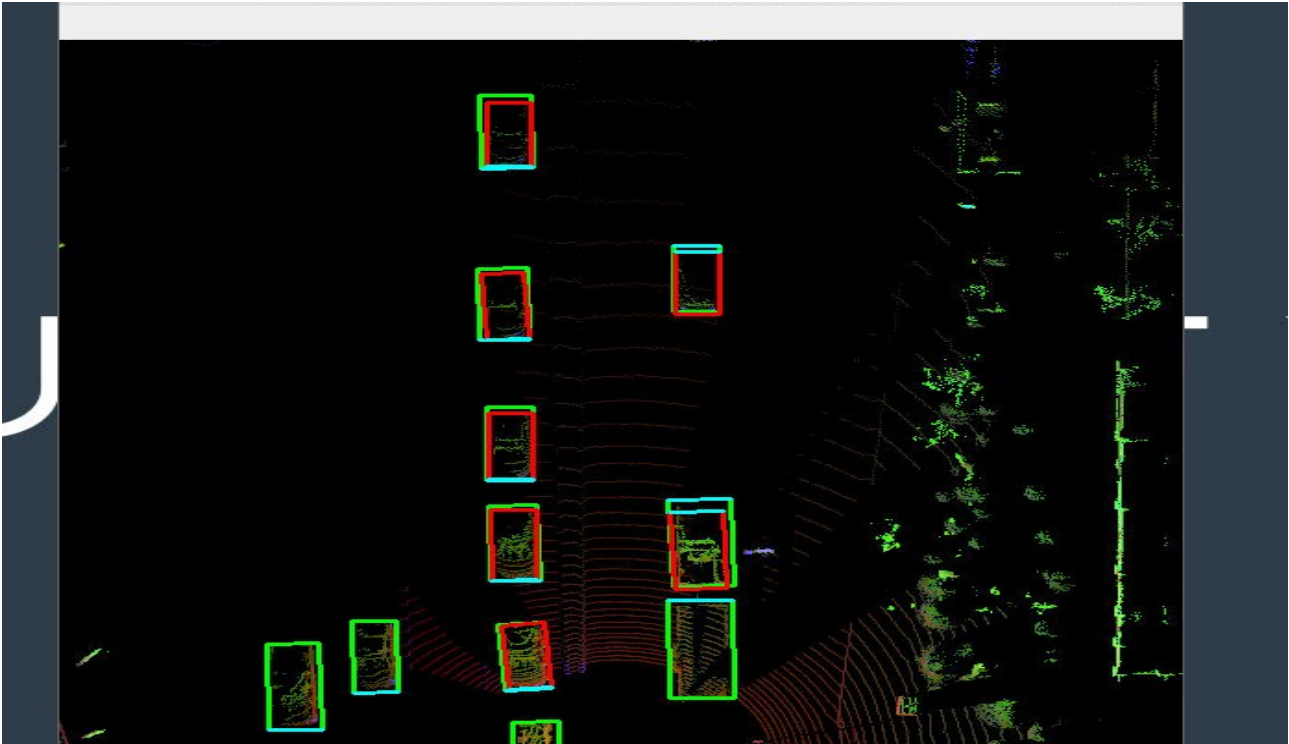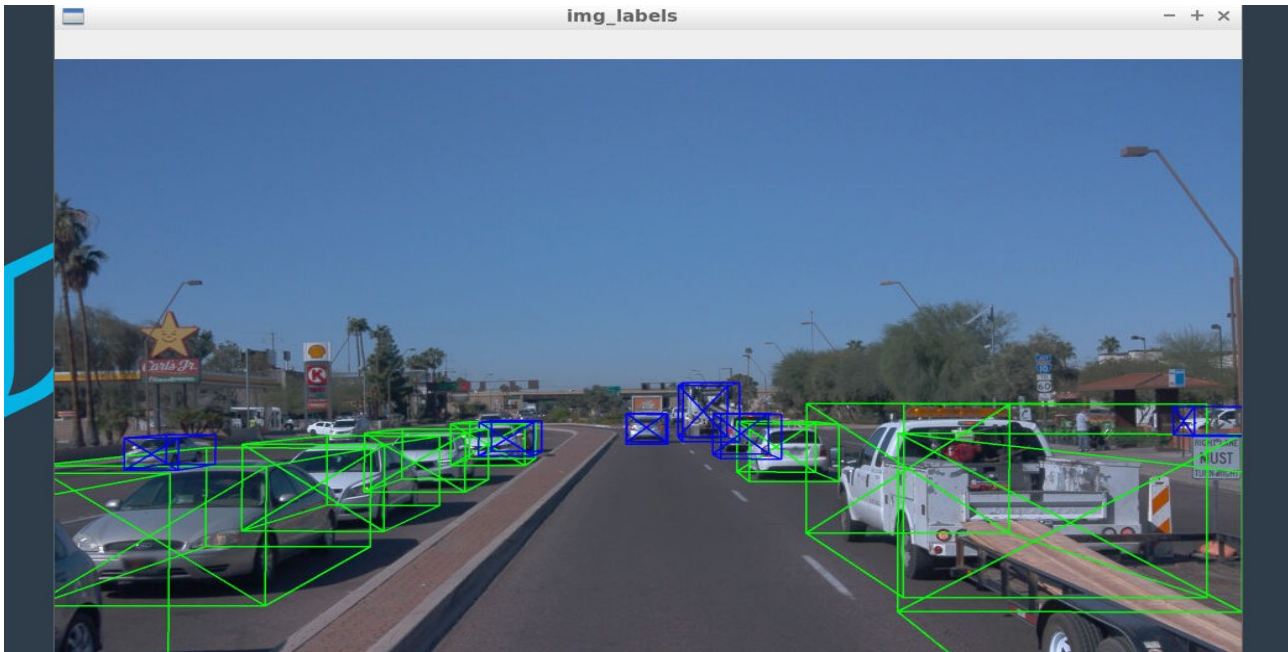
Step-4: Performance detection for 3D Object Detection

In this step, the performance is computed by getting the IOU between labels and detections to get the false positive and false negative values.The task is to compute the geometric overlap between the bounding boxes of labels and the detected objects:

- Assigning a detected object to a label if IOU exceeds threshold
- Computing the degree of geometric overlap
- For multiple matches objects/detections pair with maximum IOU are kept
- Computing the false negative and false positive values
- Computing precision and recall over the false positive and false negative values

The changes in the code are:


The changes for "objdet_eval.py" where the precision and recall are calculated as functions of false positives and negatives:

```
README.md        ×    loop_over_datase...   ×    objdet_eval.py        ×

87    |             center_devs.append(best_match[1:])
88
89
90        ####### ID_S4_EX2 START #######
91        #######
92        print("student task ID_S4_EX2")
93
94        # compute positives and negatives for precision/recall
95
96        ## step 1 : compute the total number of positives present in the scene
97        all_positives = labels_valid.sum()
98
99        ## step 2 : compute the number of false negatives
100       true_positives= len(ious)
101       false_negatives = all_positives-true_positives
102
103       ## step 3 : compute the number of false positives
104       false_positives = len(detections)-all_positives
105
106       #######
107       ####### ID_S4_EX2 END #######
108
109       pos_negs = [all_positives, true_positives, false_negatives, false_positives]
110       det_performance = [ious, center_devs, pos_negs]
111
112       return det_performance
113
```

```
README.md        ×    loop_over_datase...   ×    objdet_eval.py        ×

50
51             ## step 1 : extract the four corners of the current label bounding-box
52             box=label.box
53             box_1=tools.compute_box_corners(box.center_x, box.center_y, box.width, box.length, box.heading)
54
55             ## step 2 : loop over all detected objects
56 ▾           for objects in detections:
57
58                 ## step 3 : extract the four corners of the current detection
59                 _id, x, y, z, _h, w, l, yaw= objects
60                 box_2=tools.compute_box_corners(x, y, w, l, yaw)
61
62                 ## step 4 : computer the center distance between label and detection bounding-box in x, y, and z
63                 x_dist=np.array(box.center_x-x).item()
64                 y_dist=np.array(box.center_y-y).item()
65                 z_dist=np.array(box.center_z-z).item()
66
67                 ## step 5 : compute the intersection over union (IOU) between label and detection bounding-box
68                 rect1=Polygon(box_1)
69                 rect2=Polygon(box_2)
70                 intersection=rect1.intersection(rect2).area
71                 union=rect1.union(rect2).area
72                 iou=intersection/union
73
74
75                 ## step 6 : if IOU exceeds min_iou threshold, store [iou,dist_x, dist_y, dist_z] in matches_lab_det and increase the TP
76
77 ▾               if iou>min_iou:
78                     matches_lab_det.append([iou, dist_x, dist_y, dist_z])
79
80             #######
                 ####### ID_S4_EX1 END #######
```
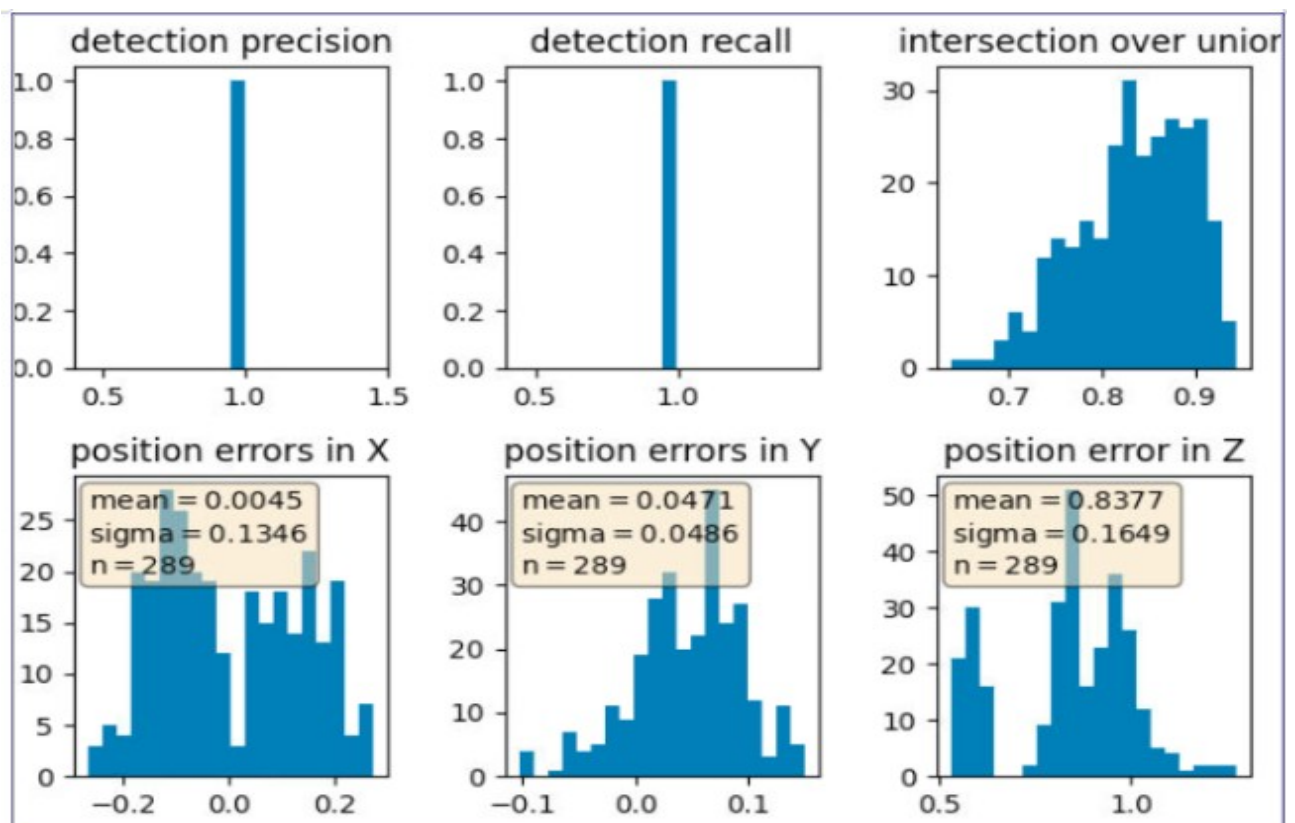
The precision recall curve is plotted showing similar results of precision =0.986 and recall=0.81372
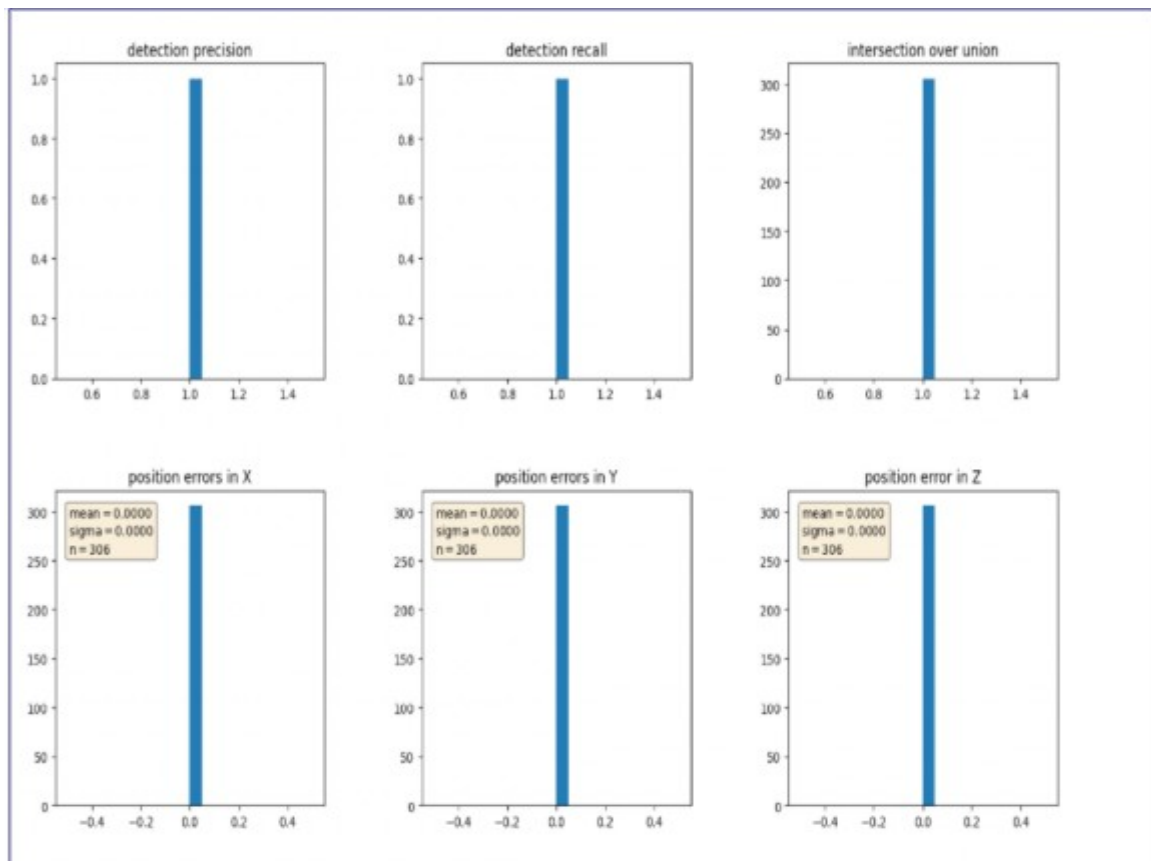
In the next step, we set the

configs_det.use_labels_as_objects=True

which results in precision and recall values as 1.This is shown in the following image:

Summary of Lidar based 3D Object Detection

From the project, it is understandable that for a stabilized tracking, lidar should be used . The conversion of range data to point cloud through spatial volumes, or points (or CNN networks) are important for further analysis. The usage of resnet/darknet and YOLO to convert these high dimensional point cloud representations to object detections through bounding boxes is essential for 3D object detection. Evaluating the performance with help of maximal IOU mapping ,mAP, and representing the precision/recall of the bounding boxes are essential to understand the effectiveness of Lidar based detection.