Section 1: Compute Lidar Point-Cloud from Range Image

1.1Visualize range image channels:

- Convert range image “range” channel to 8bit
- Convert range image “intensity” channel to 8bit
- Crop range image to +/- 90 deg. left and right of the forward-facing x-axis
- Stack cropped range and intensity image vertically and visualize the result using OpenCV

Changes in "loop_over_dataset.py"

```
## Select Waymo Open Dataset file and frame numbers
data_filename = 'training_segment-1005081002024129653_5313_150_5333_150_with_camera_labels.tfrecord' # Sequence 1
# data_filename = 'training_segment-10072231702153043603_5725_000_5745_000_with_camera_labels.tfrecord' # Sequence 2
# data_filename = 'training_segment-10963653239323173269_1924_000_1944_000_with_camera_labels.tfrecord' # Sequence 3
show_only_frames = [0, 1] # show only frames in interval for debugging
```

```
## Selective execution and visualization
exec_data = []
exec_detection = [] # options are 'bev_from_pcl', 'detect_objects', 'validate_object_labels', 'measure_detection_pe
exec_tracking = [] # options are 'perform_tracking'
exec_visualization = ['show_range_image'] # options are 'show_range_image', 'show_bev', 'show_pcl', 'show_labels_in
exec_list = make_exec_list(exec_detection, exec_tracking, exec_visualization)
vis_pause_time = 0 # set pause time between frames in ms (0 = stop between frames until key is pressed)
```

The implantation of "objdet_pcl.py"

```
# visualize range image
def show_range_image(frame, lidar_name):

    ####### ID_S1_EX1 START #######
    #######
    print("student task ID_S1_EX1")

    # step 1 : extract lidar data and range image for the roof-mounted lidar
    # lidar_name = dataset_pb2.LaserName.TOP
    lidar = [obj for obj in frame.lasers if obj.name == lidar_name][0]

    # step 2 : extract the range and the intensity channel from the range image
    # ri = []
    if len(lidar.ri_return1.range_image_compressed) > 0: # use first response
        ri = dataset_pb2.MatrixFloat()
        ri.ParseFromString(zlib.decompress(lidar.ri_return1.range_image_compressed))
        ri = np.array(ri.data).reshape(ri.shape.dims)

    # # step 3 : set values <0 to zero
    ri[ri<0]=0.0

    # # step 4 : map the range channel onto an 8-bit scale and make sure that the full range of values is appropriately considered
    ri_intensity = ri[:,:,1]
    ri_intensity = np.amax(ri_intensity)/2 * ri_intensity * 255 / (np.amax(ri_intensity) - np.amin(ri_intensity))
    img_intensity = ri_intensity.astype(np.uint8)
```

```python
# # step 5 : map the intensity channel onto an 8-bit scale and normalize with the difference between the 1- and 99-percentile to mitigate the influence of outliers
deg90 = int(img_intensity.shape[1] / 4)
ri_center = int(img_intensity.shape[1]/2)
img_intensity = img_intensity[:,ri_center-deg90:ri_center+deg90]

p1, p99 = np.percentile(img_intensity, (1, 99))
intensity_image_norm = cv2.normalize(img_intensity, None, 0, 255, cv2.NORM_MINMAX)
intensity_image_norm = cv2.convertScaleAbs(intensity_image_norm, alpha=(255.0/(p99-p1)), beta=(-255.0*p1/(p99-p1)))

# # step 6 : stack the range and intensity image vertically using np.vstack and convert the result to an unsigned 8-bit integer
stacked_image = np.vstack((intensity_image_norm, img_intensity))
stacked_image = stacked_image.astype(np.uint8)
cv2.imshow('range_image', stacked_image)
cv2.waitKey(0)

# img_range_intensity = [] # remove after implementing all steps

#######
####### ID_S1_EX1 END #######
# return img_intensity
return stacked_image
```

The range image sample:

1.2Visualize lidar point-cloud

Changes in "loop_over_dataset.py"

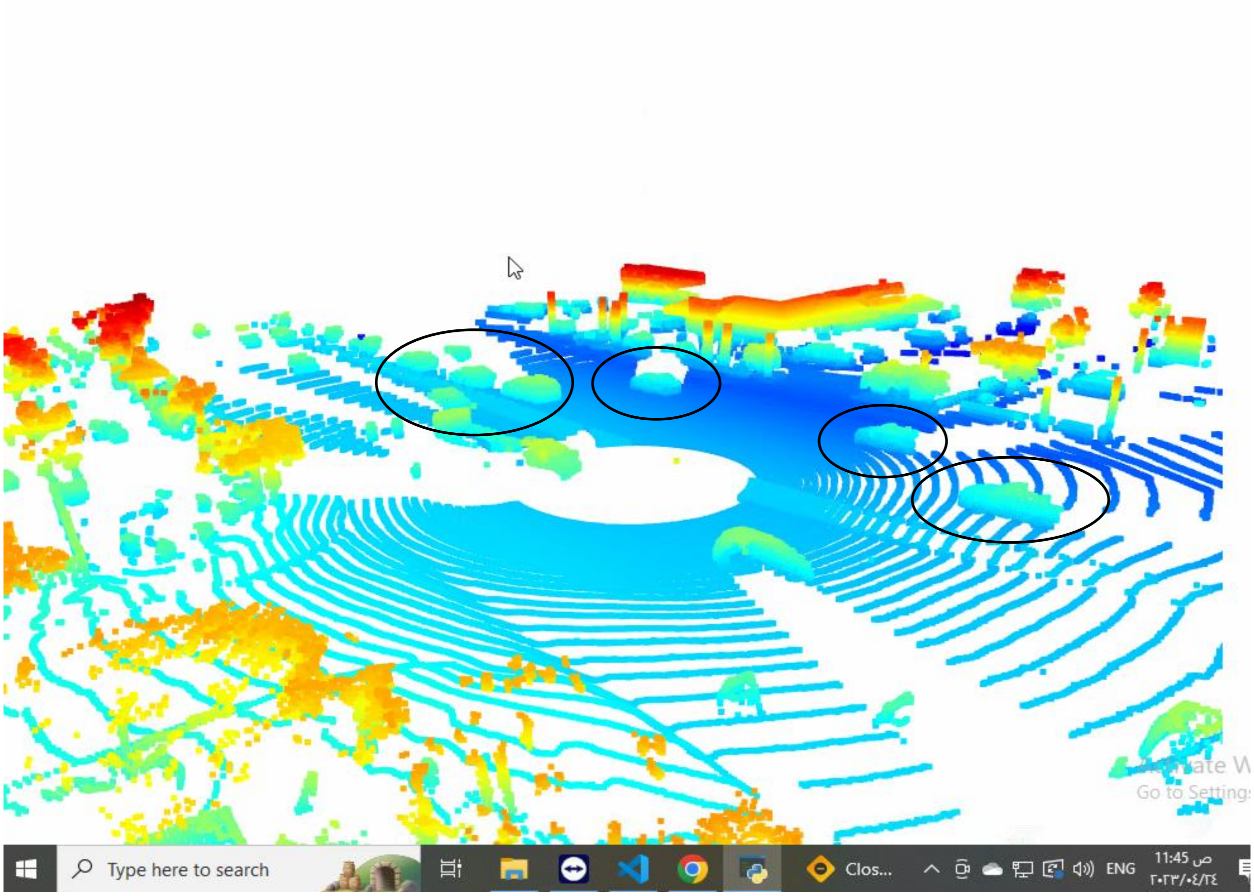
```python
## Select Waymo Open Dataset file and frame numbers
# data_filename = 'training_segment-1005081002024129653_5313_150_5333_150_with_camera_labels.tfrecord' # Sequence 1
# data_filename = 'training_segment-10072231702153043603_5725_000_5745_000_with_camera_labels.tfrecord' # Sequence 2
data_filename = 'training_segment-10963653239323173269_1924_000_1944_000_with_camera_labels.tfrecord' # Sequence 3
show_only_frames = [0, 200] # show only frames in interval for debugging
```

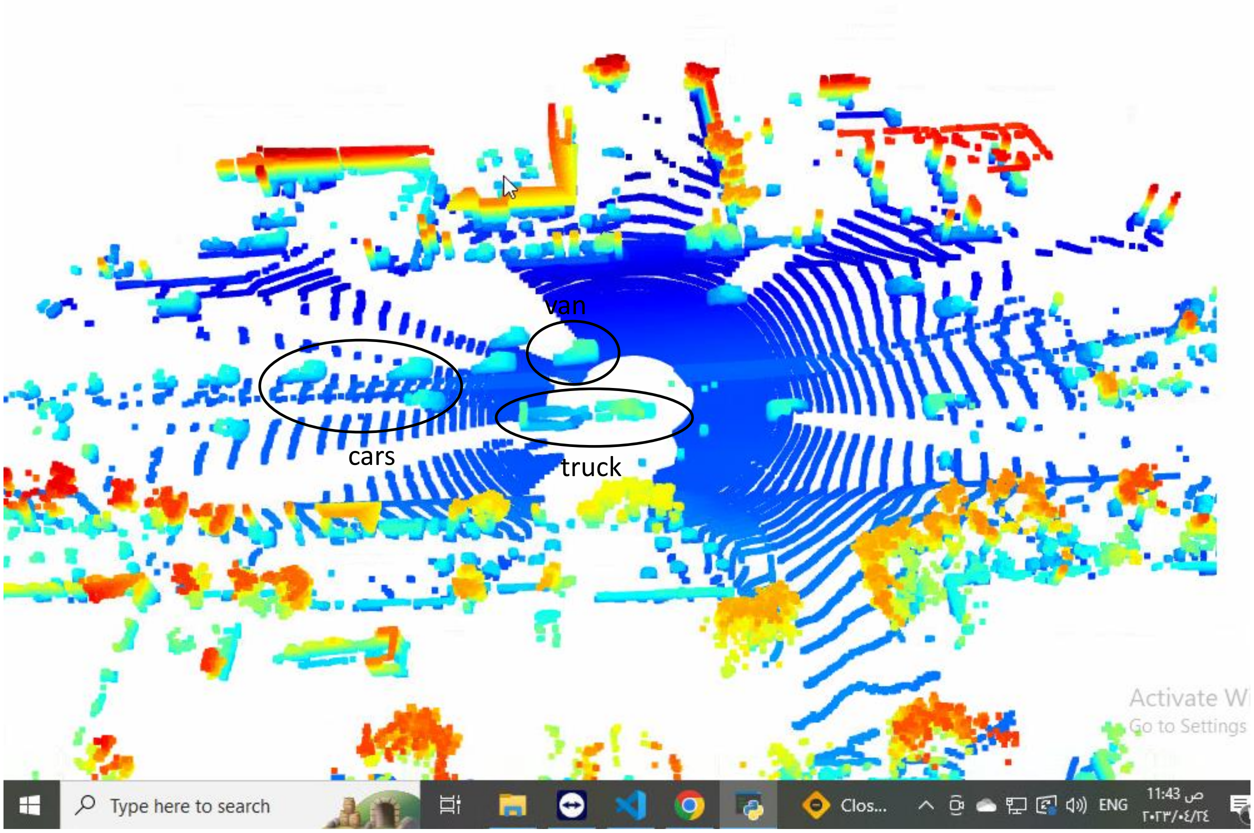
```python
## Selective execution and visualization
exec_data = []
exec_detection = [] # options are 'bev_from_pcl', 'detect_objects', 'validate_object_labels', 'measure_detection_perfo
exec_tracking = [] # options are 'perform_tracking'
exec_visualization = ['show_pcl'] # options are 'show_range_image', 'show_bev', 'show_pcl', 'show_labels_in_image', 's
exec_list = make_exec_list(exec_detection, exec_tracking, exec_visualization)
vis_pause_time = 0 # set pause time between frames in ms (0 = stop between frames until key is pressed)
```

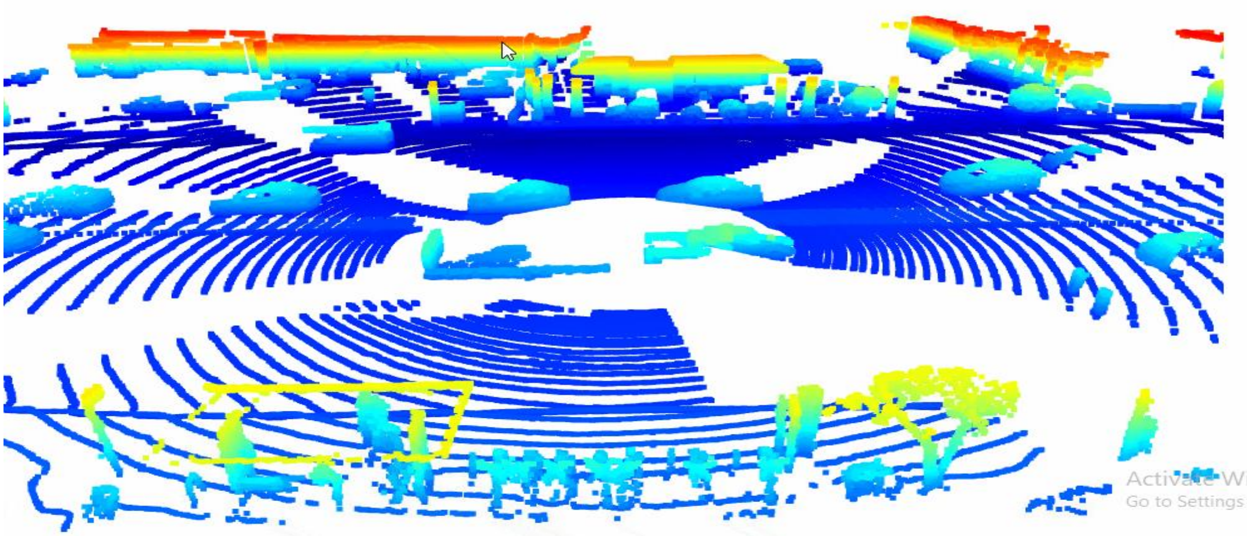
The implantation of "objdet_pcl.py"

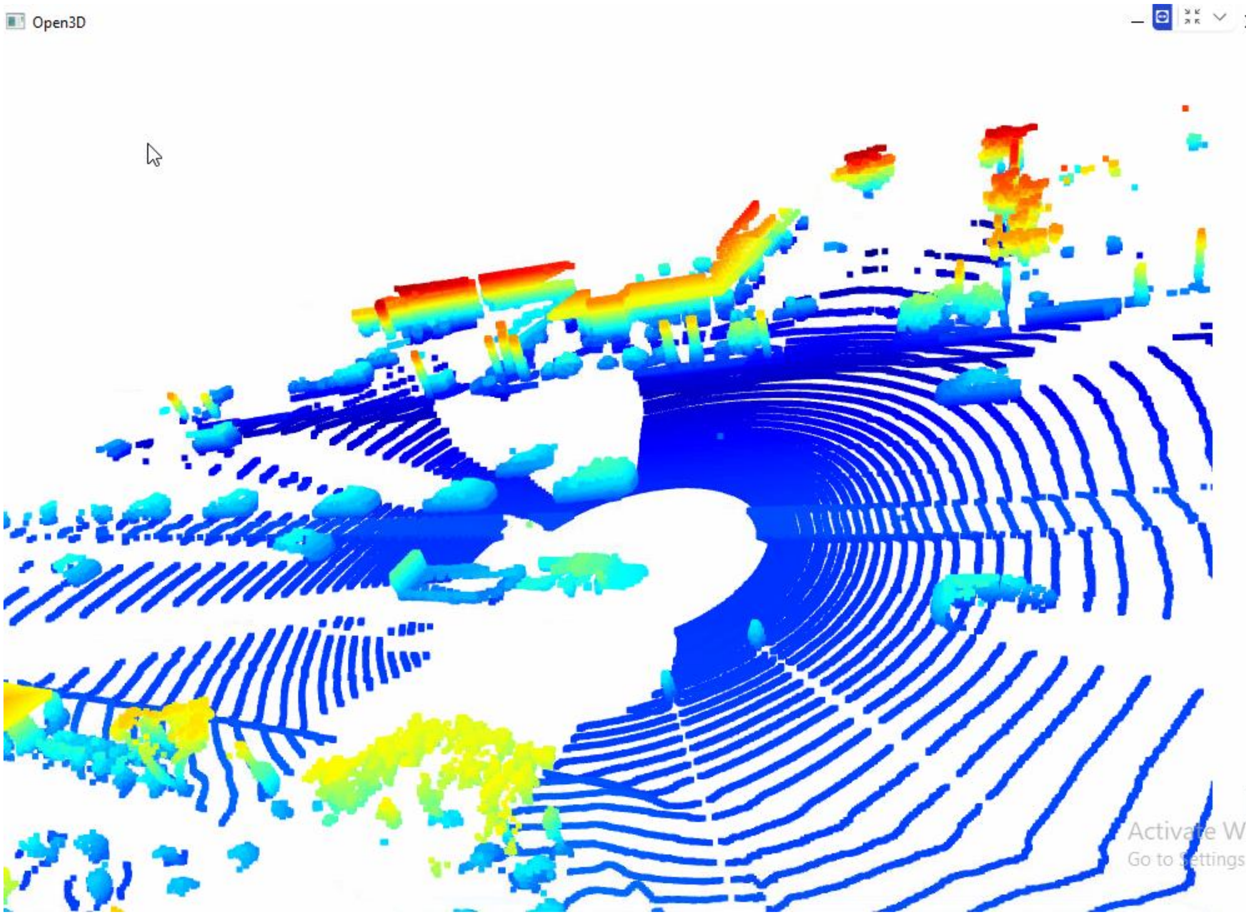
```
# visualize lidar point-cloud
def show_pcl(pcl):

    ####### ID_S1_EX2 START #######
    #######
    print("student task ID_S1_EX2")

    # step 1 : initialize open3d with key callback and create window
    vis = o3d.visualization.VisualizerWithKeyCallback()
    vis.create_window()

    # step 2 : create instance of open3d point-cloud class
    pcd = o3d.geometry.PointCloud()

    # step 3 : set points in pcd instance by converting the point-cloud into 3d vectors (using open3d function Vector3dVector)
    pcd.points = o3d.utility.Vector3dVector(pcl[:,:3])

    # step 4 : for the first frame, add the pcd instance to visualization using add_geometry; for all other frames, use update_geometry instead
    vis.add_geometry(pcd)
    vis.update_geometry(pcd)

    #step 5 : visualize point cloud and keep window open until right-arrow is pressed (key-code 262)
    def key_callback(visualizer):
        visualizer.close()
    vis.register_key_callback(262,key_callback)
    vis.run()
    ####### ID_S1_EX2 END #######
```

lidar point-cloud




van
cars
truck
Activate Wi
Go to Settings
Type here to search
Clos...
ENG
11:43 ص


Activate Wi
Go to Settings



The rear bumper is one of the more stable elements. In certain situations, the extra features include rear window shields, front automobile lights, and head-over lights. The intensity channels are used to find them. The automobile's chassis is the most prominent and identifiable element from a lidar standpoint. The backlights are the main stable elements once the photos are evaluated using various settings, and the bounding boxes (used from Step-3) are accurately allocated to the automobiles.

Section 2: Create Birds-Eye View from Lidar PCL

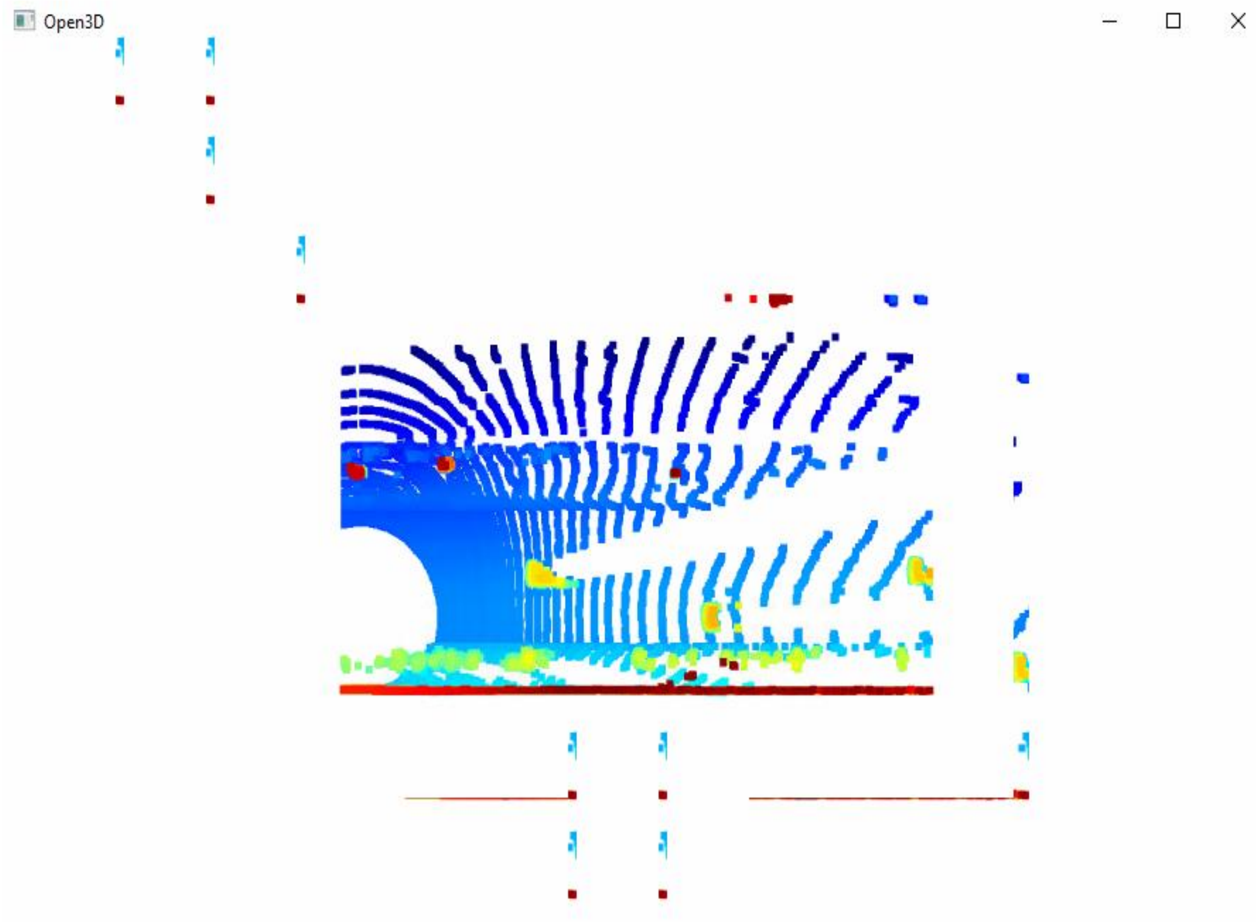
Changes in "loop_over_dataset.py"

```
## Select Waymo Open Dataset file and frame numbers
data_filename = 'training_segment-1005081002024129653_5313_150_5333_150_with_camera_labels.tfrecord' # Sequence 1
# data_filename = 'training_segment-10072231702153043603_5725_000_5745_000_with_camera_labels.tfrecord' # Sequence 2
# data_filename = 'training_segment-10963653239323173269_1924_000_1944_000_with_camera_labels.tfrecord' # Sequence 3
show_only_frames = [0, 1] # show only frames in interval for debugging
```

```
## Selective execution and visualization
exec_data = ['pcl_from_rangeimage']
exec_detection = ['bev_from_pcl'] # options are 'bev_from_pcl', 'detect_objects', 'validate_object_labels', 'measur
exec_tracking = [] # options are 'perform_tracking'
exec_visualization = [] # options are 'show_range_image', 'show_bev', 'show_pcl', 'show_labels_in_image', 'show_obj
exec_list = make_exec_list(exec_detection, exec_tracking, exec_visualization)
vis_pause_time = 0 # set pause time between frames in ms (0 = stop between frames until key is pressed)
```

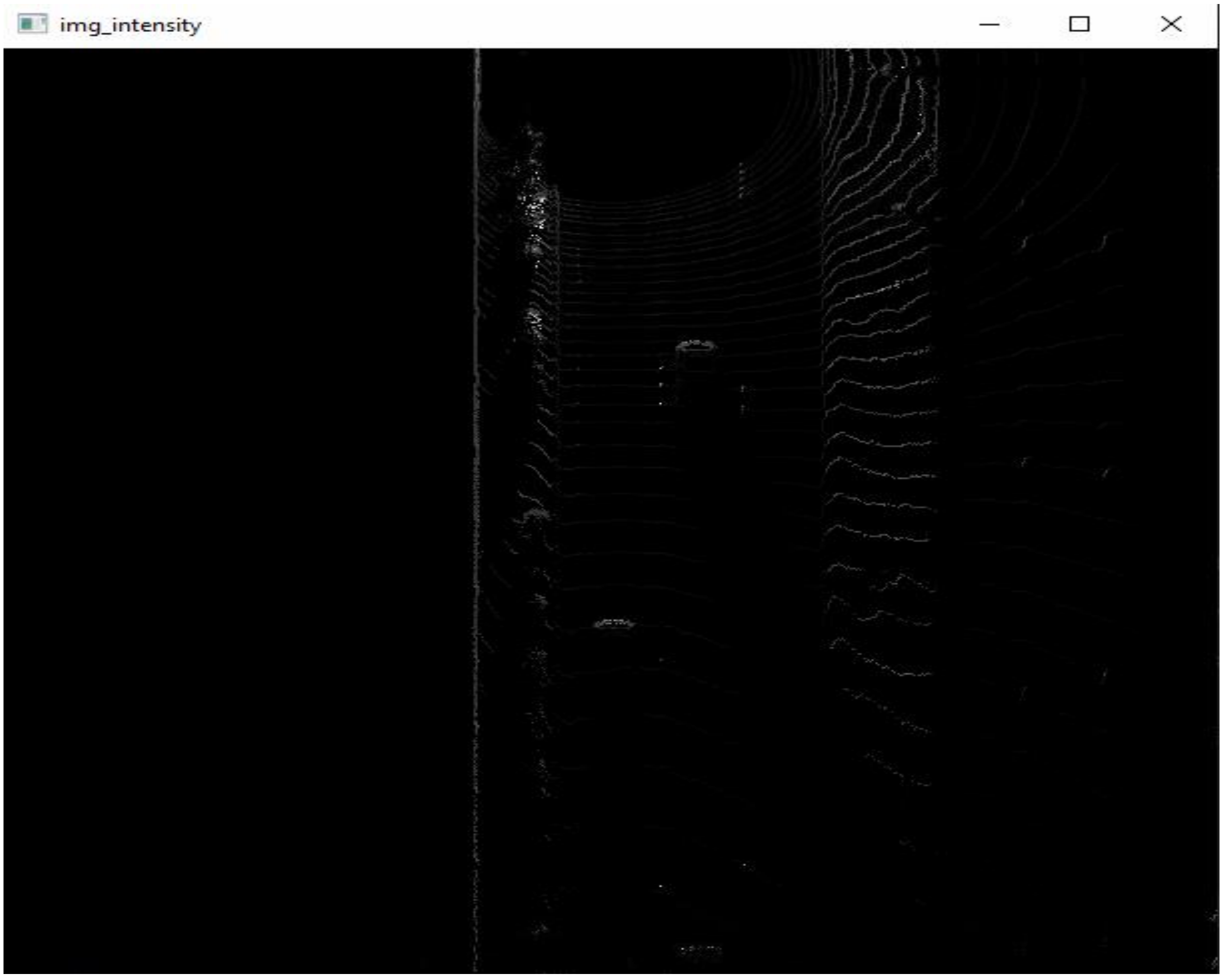
Convert sensor coordinates to BEV-map coordinates

```
# create birds-eye view of lidar data
def bev_from_pcl(lidar_pcl, configs):

    # remove lidar points outside detection area and with too low reflectivity
    mask = np.where((lidar_pcl[:, 0] >= configs.lim_x[0]) & (lidar_pcl[:, 0] <= configs.lim_x[1]) &
                    (lidar_pcl[:, 1] >= configs.lim_y[0]) & (lidar_pcl[:, 1] <= configs.lim_y[1]) &
                    (lidar_pcl[:, 2] >= configs.lim_z[0]) & (lidar_pcl[:, 2] <= configs.lim_z[1]))
    lidar_pcl = lidar_pcl[mask]

    # shift level of ground plane to avoid flipping from 0 to 255 for neighboring pixels
    lidar_pcl[:, 2] = lidar_pcl[:, 2] - configs.lim_z[0]


    # convert sensor coordinates to bev-map coordinates (center is bottom-middle)
    ####### ID_S2_EX1 START #######
    #######
    print("student task ID_S2_EX1")

    ## step 1 :  compute bev-map discretization by dividing x-range by the bev-image height (see configs)
    bev_discret = (configs.lim_x[1] - configs.lim_x[0]) / configs.bev_height
    ## step 2 : create a copy of the lidar pcl and transform all metrix x-coordinates into bev-image coordinates
    lidar_pcl_cpy = np.copy(lidar_pcl)
    lidar_pcl_cpy[:, 0] = np.int_(np.floor(lidar_pcl_cpy[:, 0] / bev_discret))
    # step 3 : perform the same operation as in step 2 for the y-coordinates but make sure that no negative bev-coordinates occur
    lidar_pcl_cpy[:, 1] = np.int_(np.floor(lidar_pcl_cpy[:, 1] / bev_discret) + (configs.bev_width + 1) / 2)
    # step 4 : visualize point-cloud using the function show_pcl from a previous task
    show_pcl(lidar_pcl_cpy)

    #######
    ####### ID_S2_EX1 END #######
```



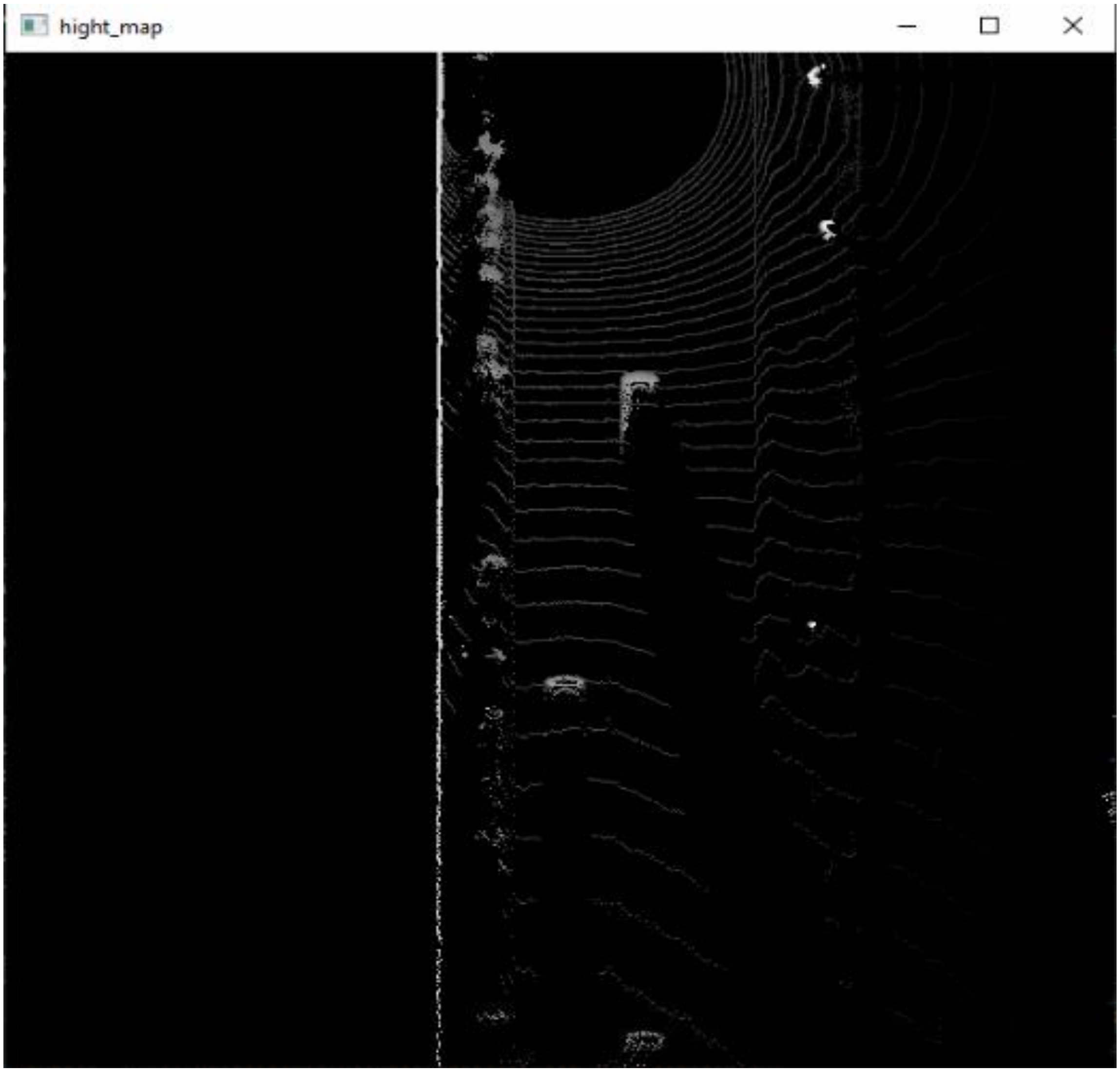
Compute intensity layer of the BEV map

```python
# Compute intensity layer of the BEV map
####### ID_S2_EX2 START #######
#######
# print("student task ID_S2_EX2")

## step 1 : create a numpy array filled with zeros which has the same dimensions as the BEV map
intensity_map = np.zeros((configs.bev_height, configs.bev_width))

# step 2 : re-arrange elements in lidar_pcl_cpy by sorting first by x, then y, then -z (use numpy.lexsort)
lidar_pcl_cpy[lidar_pcl_cpy[:,3]>1.0,3] = 1.0
idx_intensity = np.lexsort((-lidar_pcl_cpy[:, 2], lidar_pcl_cpy[:, 1], lidar_pcl_cpy[:, 0]))
lidar_pcl_top = lidar_pcl_cpy[idx_intensity]

## step 3 : extract all points with identical x and y such that only the top-most z-coordinate is kept (use numpy.unique)
##          also, store the number of points per x,y-cell in a variable named "counts" for use in the next task
lidar_pcl_int, indices, count = np.unique(lidar_pcl_cpy[:, 0:2], axis=0, return_index=True, return_counts=True)
lidar_pcl_top = lidar_pcl_cpy[indices]

## step 4 : assign the intensity value of each unique entry in lidar_top_pcl to the intensity map
##          make sure that the intensity is scaled in such a way that objects of interest (e.g. vehicles) are clearly visible
##          also, make sure that the influence of outliers is mitigated by normalizing intensity on the difference between the max. and min. value within the point c
intensity_map[np.int_(lidar_pcl_top[:, 0]),
              np.int_(lidar_pcl_top[:, 1])] = lidar_pcl_top[:, 3] / (np.amax(lidar_pcl_top[:, 3])-np.amin(lidar_pcl_top[:, 3]))
## step 5 : temporarily visualize the intensity map using OpenCV to make sure that vehicles separate well from the background
img_intensity = intensity_map * 256
img_intensity = img_intensity.astype(np.uint8)
cv2.imshow('img_intensity', img_intensity)
cv2.waitKey(0)
cv2.destroyAllWindows()
# #######
# ####### ID_S2_EX2 END #######
```



Compute height layer of the BEV map

```
# Compute height layer of the BEV map
####### ID_S2_EX3 START #######
#######
print("student task ID_S2_EX3")

## step 1 : create a numpy array filled with zeros which has the same dimensions as the BEV map
high_map = np.zeros((configs.bev_height + 1, configs.bev_width))

## step 2 : assign the height value of each unique entry in lidar_top_pcl to the height map
##          make sure that each entry is normalized on the difference between the upper and lower height defined in the config file
##          use the lidar_pcl_top data structure from the previous task to access the pixels of the height_map
high_map[np.int_(lidar_pcl_top[:, 0]), np.int_(lidar_pcl_top[:, 1])] = lidar_pcl_top[:, 2] / float(np.abs(configs.lim_z[1] - configs.lim_z[0]))

## step 3 : temporarily visualize the intensity map using OpenCV to make sure that vehicles separate well from the background
img_height = high_map * 256
img_height = img_height.astype(np.uint8)
cv2.imshow('hight_map', high_map)
cv2.waitKey(0)
cv2.destroyAllWindows()
#######
####### ID_S2_EX3 END #######
```

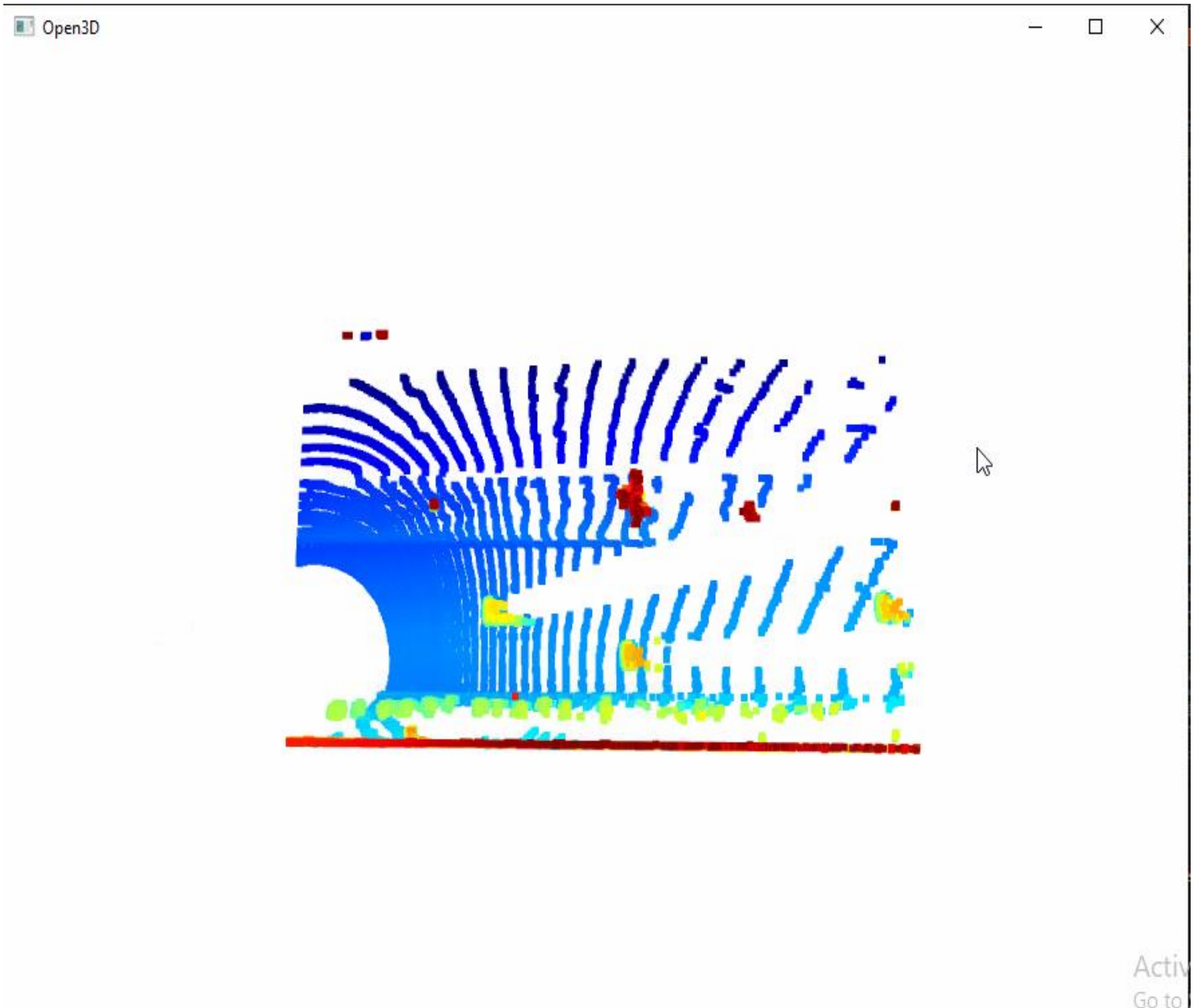


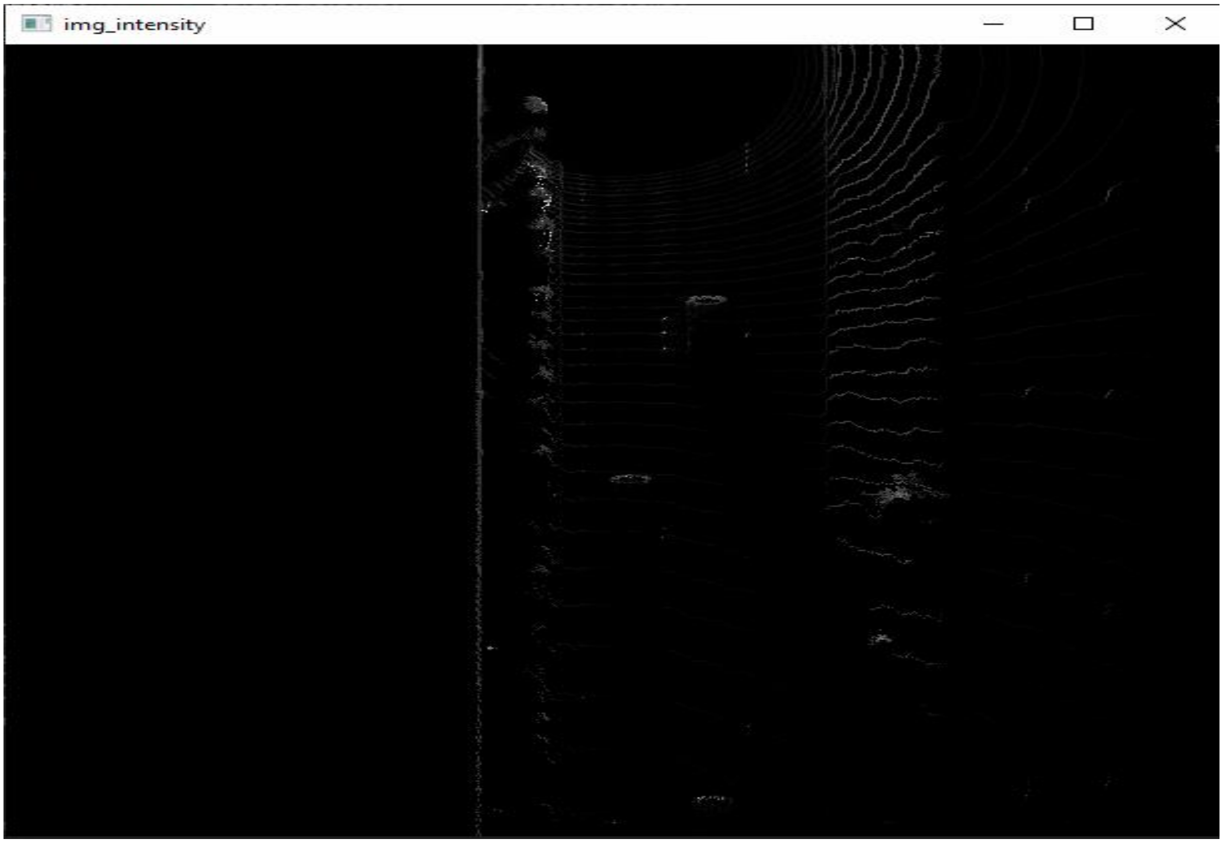
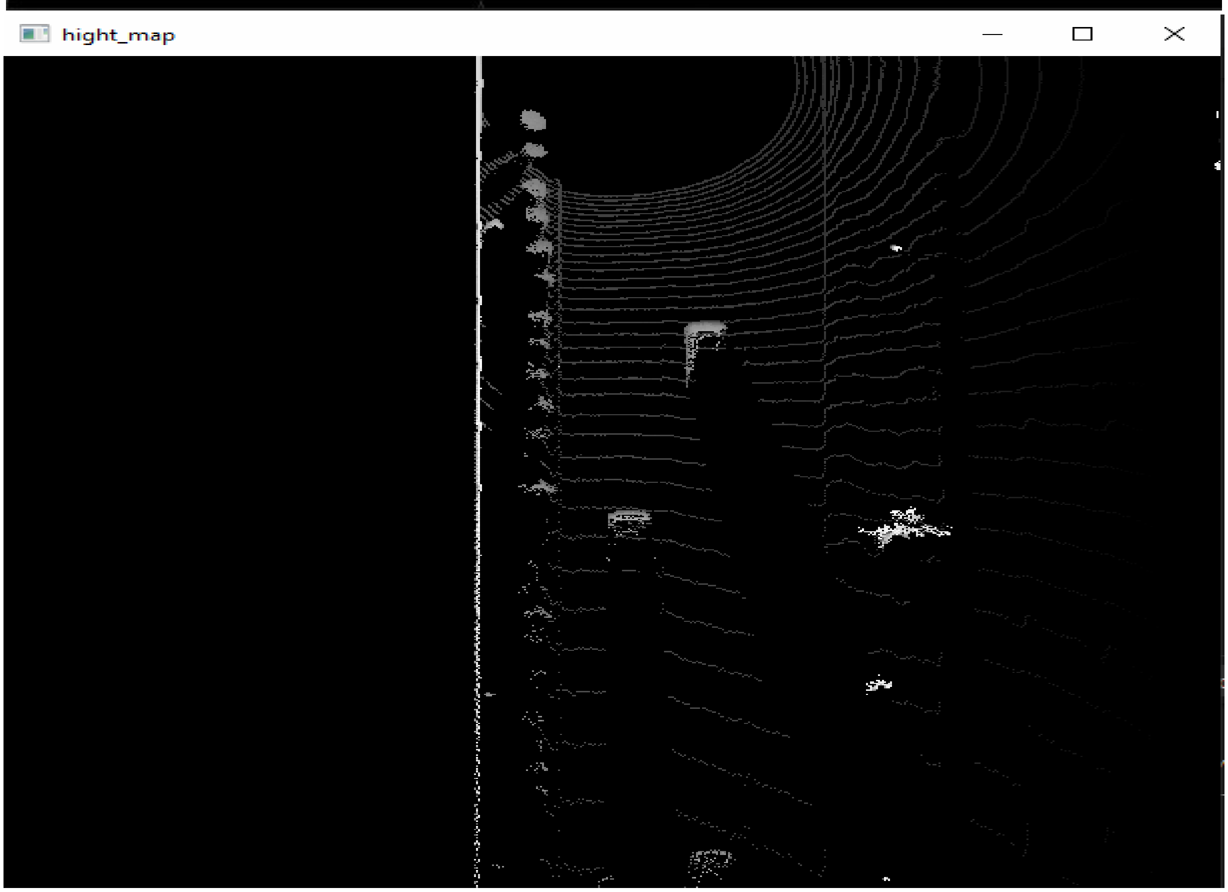
Section 3: Model-based Object Detection in BEV Image

Changes in "loop_over_dataset.py"

```
## Select Waymo Open Dataset file and frame numbers
data_filename = 'training_segment-1005081002024129653_5313_150_5333_150_with_camera_labels.tfrecord' # Sequence 1
# data_filename = 'training_segment-10072231702153043603_5725_000_5745_000_with_camera_labels.tfrecord' # Sequence 2
# data_filename = 'training_segment-10963653239323173269_1924_000_1944_000_with_camera_labels.tfrecord' # Sequence 3
show_only_frames = [0, 1] # show only frames in interval for debugging
```

```
## Selective execution and visualization
exec_data = ['pcl_from_rangeimage', 'load_image']
exec_detection = ['bev_from_pcl', 'detect_objects','measure_detection_performance'] # options are 'bev_from_pcl', 'det
exec_tracking = [] # options are 'perform_tracking'
exec_visualization = ['show_objects_in_bev_labels_in_camera'] # options are 'show_range_image', 'show_bev', 'show_pcl'
exec_list = make_exec_list(exec_detection, exec_tracking, exec_visualization)
vis_pause_time = 0 # set pause time between frames in ms (0 = stop between frames until key is pressed)
```


Open3D


img_intensity

hight_map

Extract 3D bounding boxes from model response

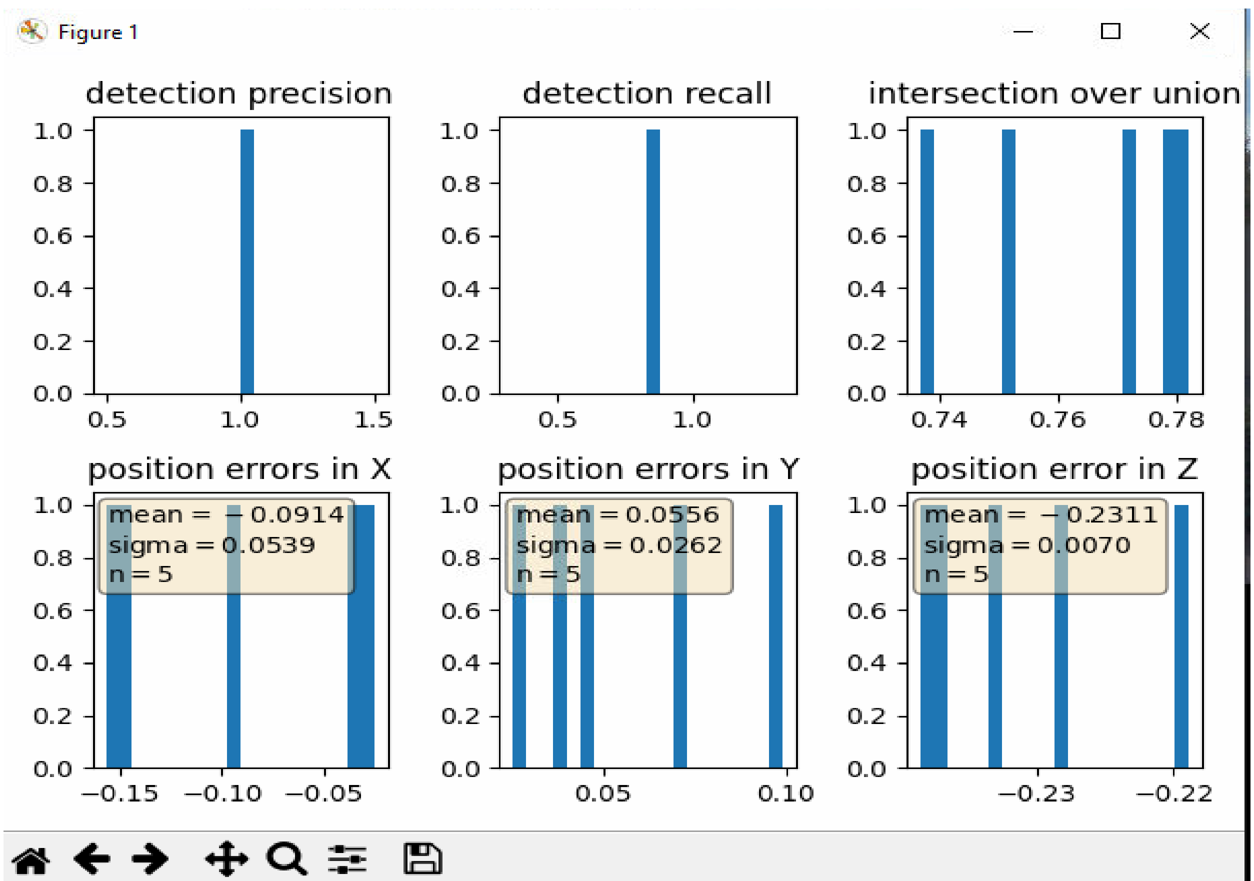
Section 4: Performance Evaluation for Object Detection

Changes in "loop_over_dataset.py"

```
## Select Waymo Open Dataset file and frame numbers
data_filename = 'training_segment-1005081002024129653_5313_150_5333_150_with_camera_labels.tfrecord' # Sequence 1
# data_filename = 'training_segment-10072231702153043603_5725_000_5745_000_with_camera_labels.tfrecord' # Sequence 2
# data_filename = 'training_segment-10963653239323173269_1924_000_1944_000_with_camera_labels.tfrecord' # Sequence 3
show_only_frames = [0, 1] # show only frames in interval for debugging
```

```
## Selective execution and visualization
exec_data = ['pcl_from_rangeimage', 'load_image']
exec_detection = ['bev_from_pcl', 'detect_objects','measure_detection_performance'] # options are 'bev_from_pcl', 'det
exec_tracking = [] # options are 'perform_tracking'
exec_visualization = ['show_objects_in_bev_labels_in_camera','show_detection_performance'] # options are 'show_range_i
exec_list = make_exec_list(exec_detection, exec_tracking, exec_visualization)
vis_pause_time = 0 # set pause time between frames in ms (0 = stop between frames until key is pressed)
```

Result from frame 50 to 51



Result from frame 0 to 200