Section 1: Compute Lidar Point-Cloud from Range Image

1- Visualize range image channels (ID_S1_EX1)

The task of this exercise is to extract two of the data channels within the range image, which are "range" and "intensity", and convert the floating-point data to an 8-bit integer value range. Once you have done so, please use the OpenCV library to stack the range and intensity image vertically and visualize it.

The attributes for code execution are:

- data_filename = 'training_segment-10050810020241296 53_5313_150_5333_150_with_camera_labels.tfrecord
- show_only_frames = [0, 1]
- exec_data = []
- exec_detection = []
- exec_tracking = []
- exec_visualization = ['show_range_image']

The function written for this task is:

```
def show_range_image(frame, lidar_name):
    ####### ID_S1_EX1 START #######
    #######
    print("student task ID_S1_EX1")

    # The code for steps 1-3 is borrowed from Exampple C1-5-1

    lidar = [obj for obj in frame.lasers if obj.name == lidar_name][0]  # 
get laser data structure from frame

    ri = dataset_pb2.MatrixFloat()

ri.ParseFromString(zlib.decompress(lidar.ri_return1.range_image_compressed)
)
    ri = np.array(ri.data).reshape(ri.shape.dims)

    # The code for step 4 is borrowed from Exampple C1-5-4
    ri[ri < 0] = 0.0
    ri_range = ri[:, :, 0]
    ri_range = ri_range * 255 / (np.amax(ri_range) - np.amin(ri_range))
    img_range = ri_range.astype(np.uint8)

    # The code of steps 5-6 borrowed from Exercise: Visualizing the 
Intensity Channel
    # map value range to 8bit
    ri_intensity = ri[:, :, 1]
    percentile_1 = percentile(ri_intensity, 1)
    percentile_99 = percentile(ri_intensity, 99)
    clipped_normalised_intensity = np.clip(ri_intensity, percentile_1, 
percentile_99) / percentile_99
    ri_intensity = 255 * clipped_normalised_intensity
    img_intensity = ri_intensity.astype(np.uint8)

    deg45 = int(img_intensity.shape[1] / 8)
    ri_center = int(img_intensity.shape[1] / 2)
    img_intensity = img_intensity[:, ri_center - deg45:ri_center + deg45]
```

```
    #######
    ####### ID_S1_EX1 END #######

    return img_intensity
```

Results for frames [0,1]

Results for frames [100,101]

2- Visualize lidar point-cloud (ID_S1_EX2)

The aim of this exercise is to use the Open3D library to display the lidar point-cloud in a 3d viewer in order to develop a feel for the nature of lidar point-clouds. It is expected to:

- Find and display 6 examples of vehicles with varying degrees of visibility in the point-cloud
- Identify vehicle features that appear as a stable feature on most vehicles (e.g. rear-bumper, tail-lights) and describe them briefly. Also, use the range image viewer from the last example to underpin your findings using the lidar intensity channel.

The attributes for code execution are:

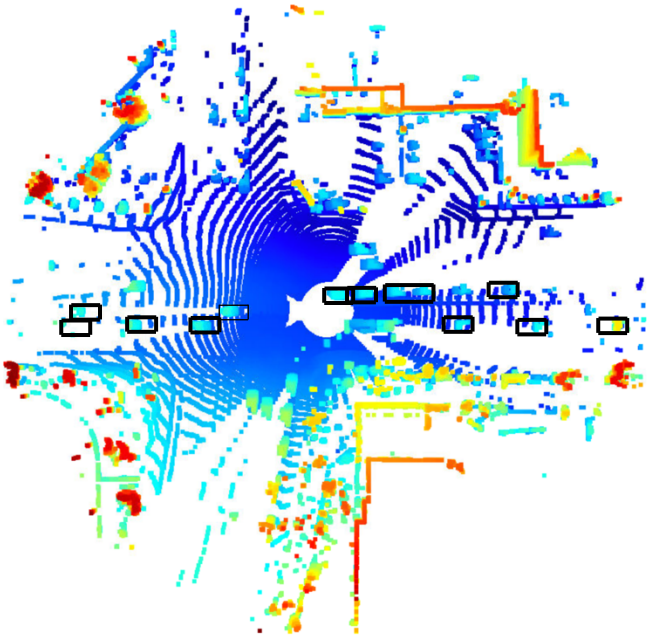
- data_filename = 'training_segment-10963653239323173269_1924_000_1944_000_with_camera_labels.tfrecord'
- show_only_frames = [0, 200]
- exec_data = []
- exec_detection = []
- exec_tracking = []
- exec_visualization = ['show_pcl

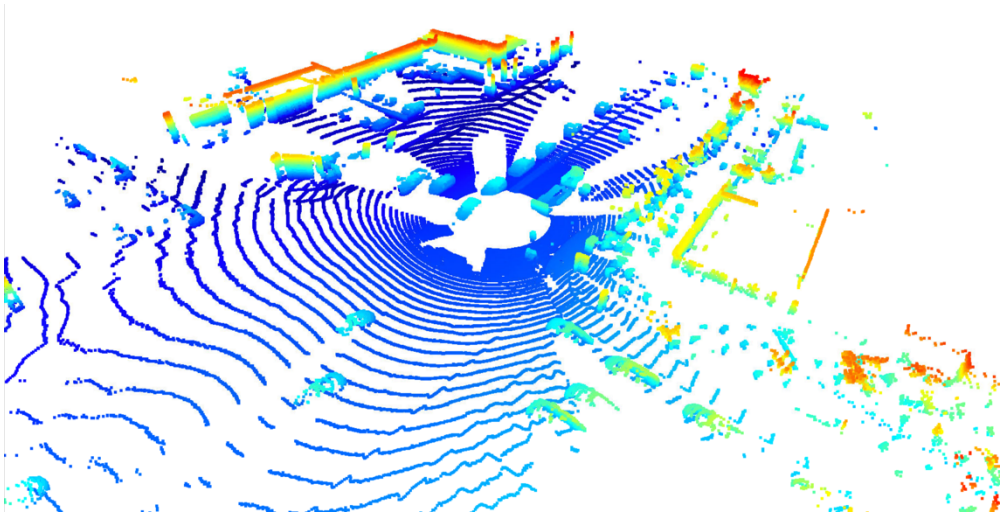
The function written for this task is:

```
def show_pcl(pcl):
    ####### ID_S1_EX2 START #######
    #######
    print("student task ID_S1_EX2")
    vis_pcl = open3d.visualization.VisualizerWithKeyCallback()  #
Visualizer with custom key callack capabilities
    vis_pcl.create_window('Open3D', 1920, 1080, 50, 50, True)  # Function
```

```
to create a window and initialize GLFW
    pcd = open3d.geometry.PointCloud()  # PointCloud class.
    # A point cloud consists of point coordinates, and optionally point
colors and point normals.

    pcd.points = open3d.utility.Vector3dVector(pcl[:, :3])

    vis_pcl.add_geometry(pcd)  # Function to add geometry to the scene and
create corresponding shaders

    global id
    id = True

    def action(vis_pcl):
        global id
        print('Right arrow is pressed')
        id = False
        return

    vis_pcl.register_key_callback(262, action)  # Function to register a
callback function for a key press event

    while id:
        vis_pcl.poll_events()
        vis_pcl.update_renderer()
    #######
    ####### ID_S1_EX2 END #######
```

Here are the results.





The identified vehicles have been highlighted by black boxes in the top picture. The bottom picture shows the environment from another angle. In this picture, the front bumper, the body shape and the location of vehicles are clearly identifiable.

Section 2: Create Birds-Eye View from Lidar PCL

1- Convert sensor coordinates to BEV-map coordinates (ID_S2_EX1)

The aim of this exercise is to perform the first step in creating a birds-eye view (BEV) perspective of the lidar point-cloud. Based on the (x,y)-coordinates in sensor space, it is expected to compute the respective coordinates within the BEV coordinate space so that in subsequent tasks, the actual BEV map can be filled with lidar data from the point-cloud.

The attributes for code execution are:

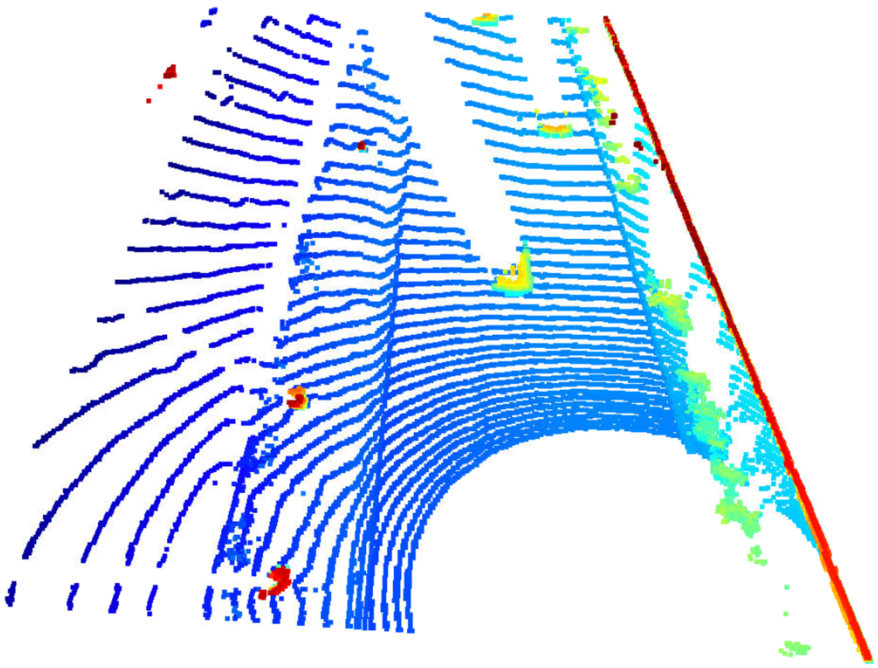
- data_filename = 'training_segment-10050810020241296 53_5313_150_5333_150_with_camera_labels.tfrecord
- show_only_frames = [0, 1]
- exec_data = ['pcl_from_rangeimage']
- exec_detection = ['bev_from_pcl']
- exec_tracking = []
- exec_visualization = []

The function written for this task is:

```
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")

    # The code is borrowed from "Exercise: Transform a Point Cloud into a
Birds-Eye View"
    bev_map_discr = (configs.lim_x[1] - configs.lim_x[0]) /
configs.bev_height
    lidar_pcl_cpy = np.copy(lidar_pcl)
    lidar_pcl_cpy[:, 0] = np.int_(np.floor(lidar_pcl_cpy[:, 0] /
bev_map_discr))

    lidar_pcl_cpy[:, 1] = np.int_(np.floor(lidar_pcl_cpy[:, 1] /
bev_map_discr) + (configs.bev_width + 1) / 2)
    lidar_pcl_cpy[:, 1] = np.abs(lidar_pcl_cpy[:, 1])
    show_pcl(lidar_pcl_cpy)

    #######
    ####### ID_S2_EX1 END #######

    # Compute intensity layer of the BEV map
```

Here is the result:



2- Compute intensity layer of the BEV map (ID_S2_EX2)

The aim of the exercise is to fill the "intensity" channel of the BEV map with data from the point-cloud. In order to do so, it is expected to identify all points with the

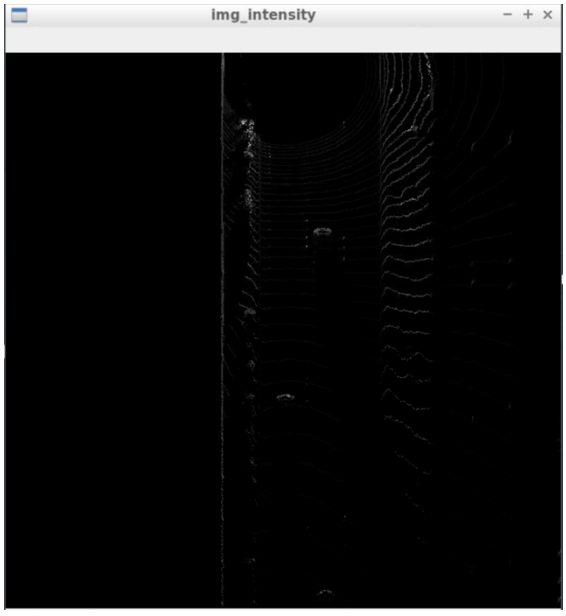
same (x,y)-coordinates within the BEV map and then assign the intensity value of the top-most lidar point to the respective BEV pixel.
The attributes for code execution are:

- data_filename = 'training_segment-10050810020241296 53_5313_150_5333_150_with_camera_labels.tfrecord
- show_only_frames = [0, 1]
- exec_data = ['pcl_from_rangeimage']
- exec_detection = ['bev_from_pcl']
- exec_tracking = []
- exec_visualization = []

The code written for this exercise is:

```
intensity_map = np.zeros((configs.bev_height, configs.bev_width))

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]

_, idx_intensity_unique, counts = np.unique(lidar_pcl_top[:, 0:2], axis=0,
return_index=True, return_counts=True)
lidar_pcl_top = lidar_pcl_top[idx_intensity_unique]

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]))

img_intensity = intensity_map * 256
img_intensity = img_intensity.astype(np.uint8)
cv2.imshow('img_intensity', img_intensity)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Here is the result:



3- Compute height layer of the BEV map (ID_S2_EX3)

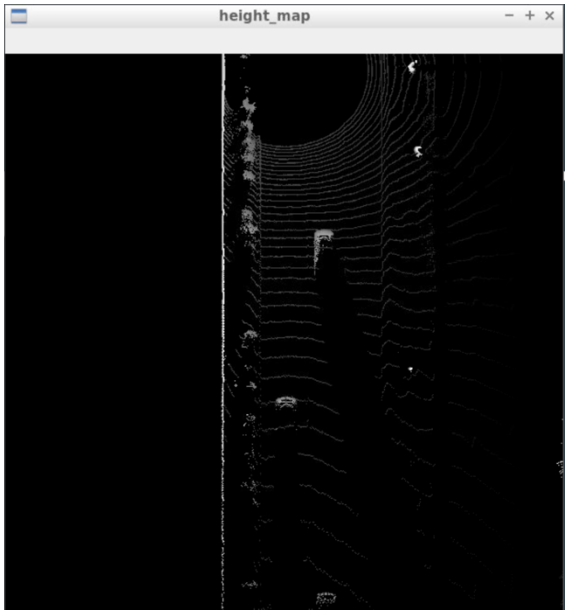
The aim of exercise is to fill the "height" channel of the BEV map with data from the point-cloud.
The attributes for code execution are:

- data_filename = 'training_segment-10050810020241296 53_5313_150_5333_150_with_camera_labels.tfrecord
- show_only_frames = [0, 1]
- exec_data = ['pcl_from_rangeimage']
- exec_detection = ['bev_from_pcl']
- exec_tracking = []
- exec_visualization = []

The code written for this exercise is:

```
height_map = np.zeros((configs.bev_height, configs.bev_width))
height_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]))
img_height = height_map * 256
img_height = img_height.astype(np.uint8)
cv2.imshow('height_map', img_height)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Here is the result:



Section 3: Model-based Object Detection in BEV Image

1- Add a second model from a GitHub repo (ID_S3_EX1)

The aim if this exercise is to decode the output and perform post-processing in detect_objects and to Visualize the results by setting the flag show_objects_in_bev_labels_in_camera
The attributes for code execution are:

- data_filename = 'training_segment-10050810020241296 53_5313_150_5333_150_with_camera_labels.tfrecord
- show_only_frames = [50, 51]
- exec_data = ['pcl_from_rangeimage', 'load_image']
- exec_detection = ['bev_from_pcl', 'detect_objects']

- exec_tracking = []
- exec_visualization = ['show_objects_in_bev_labels_in_camera']
- configs_det = det.load_configs(model_name="fpn_resnet")

The code written for this task is:

```
print("student task ID_S3_EX1-5")
outputs['hm_cen'] = _sigmoid(outputs['hm_cen'])
outputs['cen_offset'] = _sigmoid(outputs['cen_offset'])
# detections size (batch_size, K, 10)
detections = decode(outputs['hm_cen'], outputs['cen_offset'],
outputs['direction'], outputs['z_coor'],
                    outputs['dim'], K=40) #K=configs.k
detections = detections.cpu().numpy().astype(np.float32)
# print(detections)
detections = post_processing(detections, configs)
detections = detections[0][1]
print(detections)
```

Here is the result:

```
using ResNet architecture with feature pyramid
student task ID_S3_EX1-4
Loaded weights from /home/workspace/tools/objdet_models/resnet/pretrained/fpn_resnet_18_epoch_300.pth

----------------------------
processing frame #50
computing point-cloud from lidar range image
computing birds-eye view from lidar pointcloud
student task ID_S2_EX1
student task ID_S1_EX2
Right arrow is pressed
student task ID_S2_EX2
student task ID_S2_EX3
detecting objects in lidar pointcloud
student task ID_S3_EX1-5
[[9.7223872e-01 3.5127075e+02 2.1875238e+02 1.0574490e+00 1.6241086e+00
  2.0172737e+01 4.7542793e+01 1.3822034e-02]
 [6.2091374e-01 3.1184229e+02 3.5521008e+02 1.1316251e+00 1.7765539e+00
  2.0849136e+01 4.6748154e+01 8.4769009e-03]]
[[9.7223872e-01 3.5127075e+02 2.1875238e+02 1.0574490e+00 1.6241086e+00
  2.0172737e+01 4.7542793e+01 1.3822034e-02]
 [6.2091374e-01 3.1184229e+02 3.5521008e+02 1.1316251e+00 1.7765539e+00
  2.0849136e+01 4.6748154e+01 8.4769009e-03]]
student task ID_S3_EX2
loading object labels and validation from result file
loading detection performance measures from file
```

2- Extract 3D bounding boxes from model response (ID_S3_EX2)

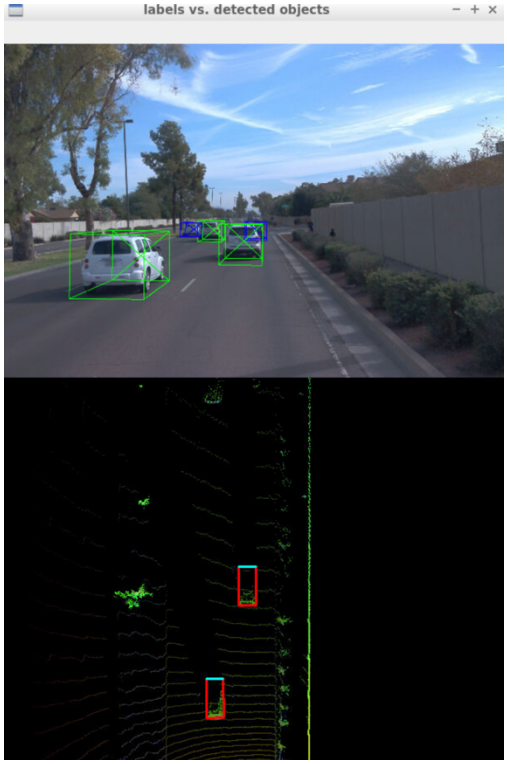
The focus of this exercise is to convert all detections coordinates from BEV coordinate space into metric coordinates in vehicle space.
The attributes for code execution in the following way:

- data_filename = 'training_segment-10059810020241296535313_150_5333_150_with_camera_labels.tfrecord
- show_only_frames = [50, 51]
- exec_data = ['pcl_from_rangeimage', 'load_image']
- exec_detection = ['bev_from_pcl', 'detect_objects']
- exec_tracking = []
- exec_visualization = ['show_objects_in_bev_labels_in_camera']
- configs_det = det.load_configs(model_name="fpn_resnet")

Here is the codes written for this exercise:

```
objects = []
for item in detections:
    _, x_bev, y_bev, z_bev, h_bev, w_bev, l_bev, yaw_bev = item
    x = y_bev / configs.bev_height * (configs.lim_x[1] - configs.lim_x[0])
    y = x_bev / configs.bev_width * (configs.lim_y[1] - configs.lim_y[0]) -
(configs.lim_y[1] - configs.lim_y[0]) / 2.0
    w = w_bev / configs.bev_width * (configs.lim_y[1] - configs.lim_y[0])
    l = l_bev / configs.bev_height * (configs.lim_x[1] - configs.lim_x[0])
    z = z_bev
    h = h_bev
    yaw = yaw_bev

    if ((x >= configs.lim_x[0]) and (x <= configs.lim_x[1])
            and (y >= configs.lim_y[0]) and (y <= configs.lim_y[1])
            and (z >= configs.lim_z[0]) and (z <= configs.lim_z[1])):
        objects.append([1, x, y, z, h, w, l, yaw])
```

Here is the result:



Section 4: Performance Evaluation for Object Detection

1- Compute intersection-over-union between labels and detections (ID_S4_EX1)

The aim of exercise is to find pairings between ground-truth labels and detections, so as to determine whether an object has been (a) missed (*false negative*), (b) successfully detected (*true positive*) or (c) has been falsely reported (*false positive*).

The attributes for code execution in the following way:

- data_filename = 'training_segment-10050810020241296 53_5313_150_5333_150_with_camera_labels.tfrecord
- show_only_frames = [50, 51]
- exec_data = ['pcl_from_rangeimage']
- exec_detection = ['bev_from_pcl', 'detect_objects', 'validate_object_labels', 'measure_detection_performance']
- exec_tracking = []
- exec_visualization = ['show_detection_performance']
- configs_det = det.load_configs(model_name="darknet")

Here is the code written for this exercise:

```
box_label = label.box
box_corners = tools.compute_box_corners(box_label.center_x,
box_label.center_y, box_label.width, box_label.length,
                                        box_label.heading)

for detveh in detections:
    _, x, y, z, _, w, l, yaw = detveh  # detveh represents the data of the
detected vehicle
    box_corners_detveh = tools.compute_box_corners(x, y, w, l, yaw)
    ## step 4 : computer the center distance between label and detection
bounding-box in x, y, and z
    distx = np.array(box_label.center_x - x).item()
    disty = np.array(box_label.center_y - y).item()
    distz = np.array(box_label.center_z - z).item()

    box1 = Polygon(box_corners)
    box2 = Polygon(box_corners_detveh)
    intersection = box1.intersection(box2).area
    union = box1.union(box2).area
    iou = intersection / union

    if iou > min_iou:
        matches_lab_det.append([iou, distx, disty, distz])
```

Here is the results:

```
student task ID_S4_EX1
student task ID_S4_EX1
student task ID_S4_EX1
ious = [0.8234376907650102, 0.8903358055776488, 0.8747769919519454]
student task ID_S4_EX2
------------------------------
processing frame #51
computing point-cloud from lidar range image
computing birds-eye view from lidar pointcloud
student task ID_S2_EX1
student task ID_S1_EX2
Right arrow is pressed
student task ID_S2_EX2
student task ID_S2_EX3
detecting objects in lidar pointcloud
student task ID_S3_EX2
validating object labels
measuring detection performance
student task ID_S4_EX1
student task ID_S4_EX1
student task ID_S4_EX1
ious = [0.8151514220492396, 0.8872181822459977, 0.8462409035977 46]
student task ID_S4_EX2
reached end of selected frames
student task ID_S4_EX3
precision = 1.0, recall = 1.0
(sdc-c2) root@4f2bd9e532e4:/home/workspace#
```

2- Compute false-negatives and false-positives (ID_S4_EX2)

Here is the code written for this exercise:

```
all_positives = labels_valid.sum()

true_positives = len(ious)
false_negatives = all_positives - true_positives
false_positives = len(detections) - true_positiv
```

Here are the results:

```
student task ID_S2_EX3
detecting objects in lidar pointcloud
student task ID_S3_EX2
validating object labels
measuring detection performance
student task ID_S4_EX1
student task ID_S4_EX1
student task ID_S4_EX1
ious = [0.8151514220492396, 0.8872181822459977, 0.846240903597746]
student task ID_S4_EX2
True positives =  3
False negatives=  0
False positives =  0
reached end of selected frames
student task ID_S4_EX3
precision = 1.0, recall = 1.0
(sdc-c2) root@4f2bd9e532e4:/home/workspace#
```

3- Compute precision and recall (ID_S4_EX3)

The aim of this exercise is to evaluate the performance of the object detection algorithm using the two standard measures "precision" and "recall" which are based on the accumulated number of positives and negatives from all frames.

The attributes for code execution in the following way:

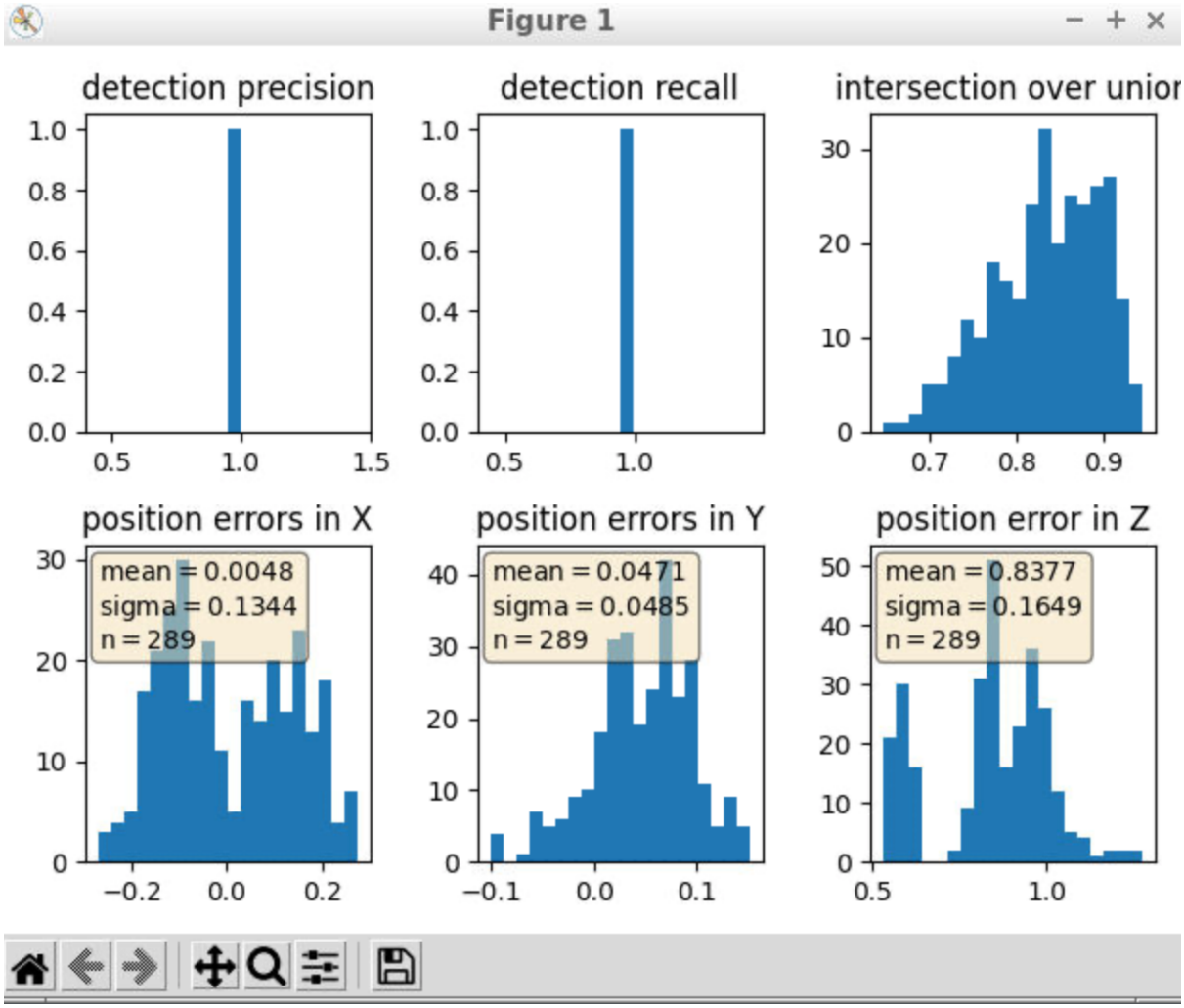
- data_filename = 'training_segment-10050810020241296 53_5313_150_5333_150_with_camera_labels.tfrecord
- show_only_frames = [50, 150]
- exec_data = ['pcl_from_rangeimage']
- exec_detection = ['bev_from_pcl', 'detect_objects', 'validate_object_labels', 'measure_detection_performance']
- exec_tracking = []
- exec_visualization = ['show_detection_performance']
- configs_det = det.load_configs(model_name="darknet")

Here is the code written for this exercise:

```
positives = sum(pos_negs_arr[:, 0])
true_positives = sum(pos_negs_arr[:, 1])
false_negatives = sum(pos_negs_arr[:, 2])
false_positives = sum(pos_negs_arr[:, 3])
precision = true_positives / float(true_positives + false_positives)
recall = true_positives / float(true_positives + false_negatives)
```

Here are the results:

```
True positives =  3
False negatives=  0
False positives =  0
------------------------------
processing frame #150
computing point-cloud from lidar range image
computing birds-eye view from lidar pointcloud
student task ID_S2_EX1
student task ID_S1_EX2
student task ID_S2_EX2
student task ID_S2_EX3
detecting objects in lidar pointcloud
student task ID_S3_EX2
validating object labels
measuring detection performance
student task ID_S4_EX1
student task ID_S4_EX1
student task ID_S4_EX1
ious = [0.8901165519591998, 0.910508505338671, 0.8975382732202531]
student task ID_S4_EX2
True positives =  3
False negatives=  0
False positives =  0
reached end of selected frames
student task ID_S4_EX3
precision = 0.9506578947368421, recall = 0.9444444444444444
```



I'm not sure why the precision and recall are lower than what is given in the project brief.