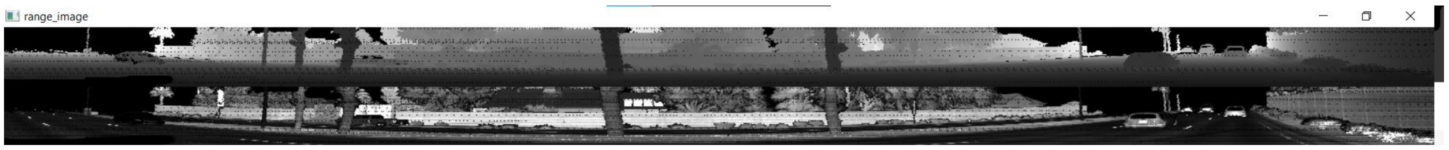
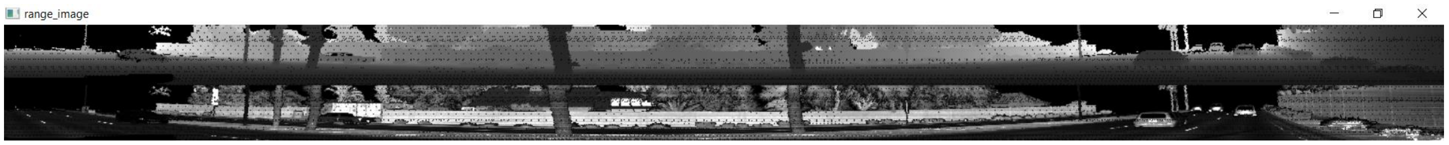
Section 1 : Compute Lidar Point-Cloud from Range Image

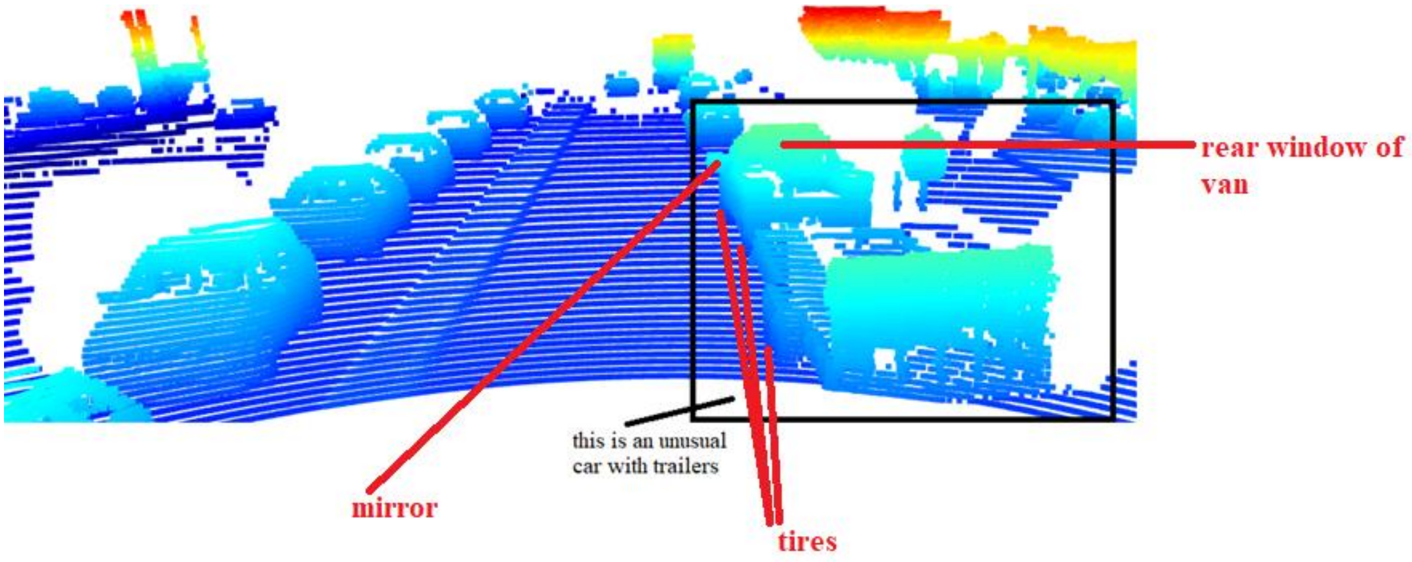
Data channel of range images in waymo dataset includes different channels. For this task, range and intensity channels are excluded and then converted to 8 bit integer values. After these operations, extracted channels are stacked and visualized via opencv library:

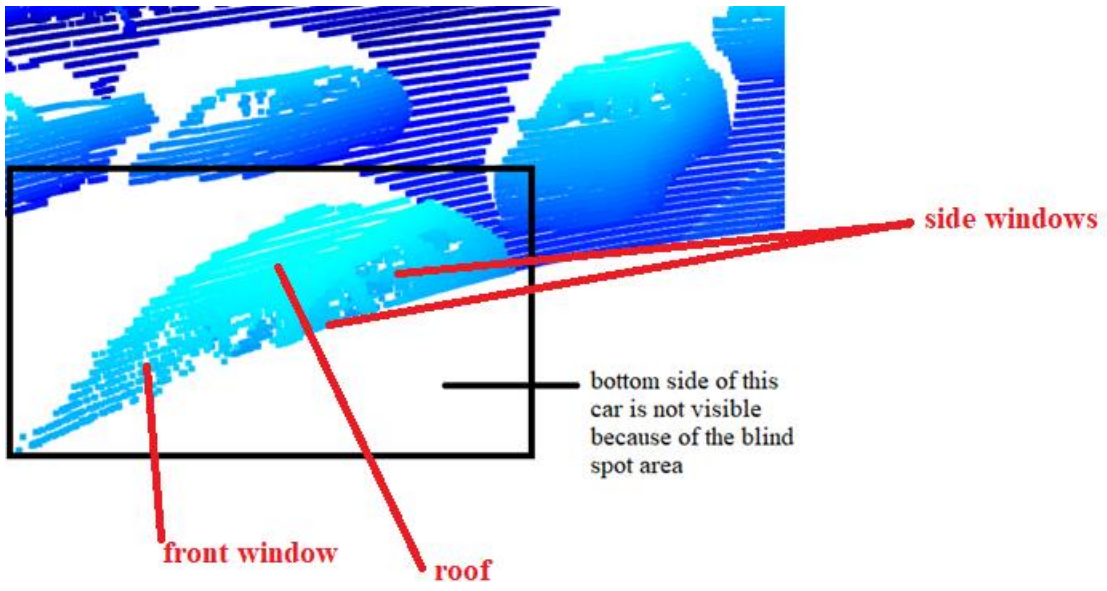


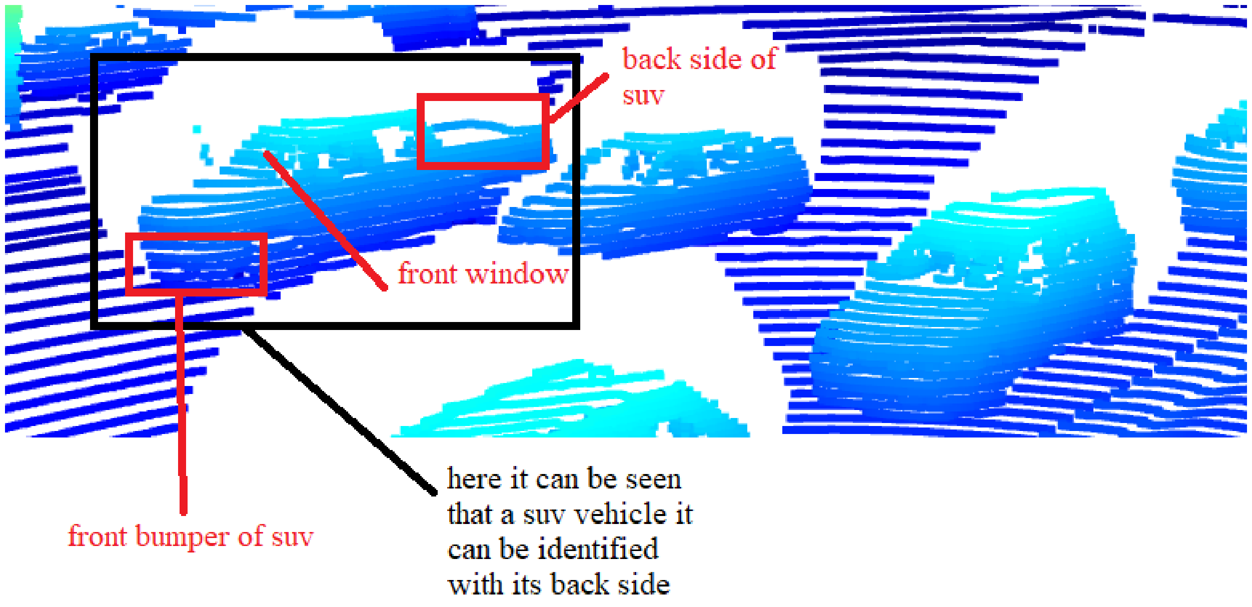


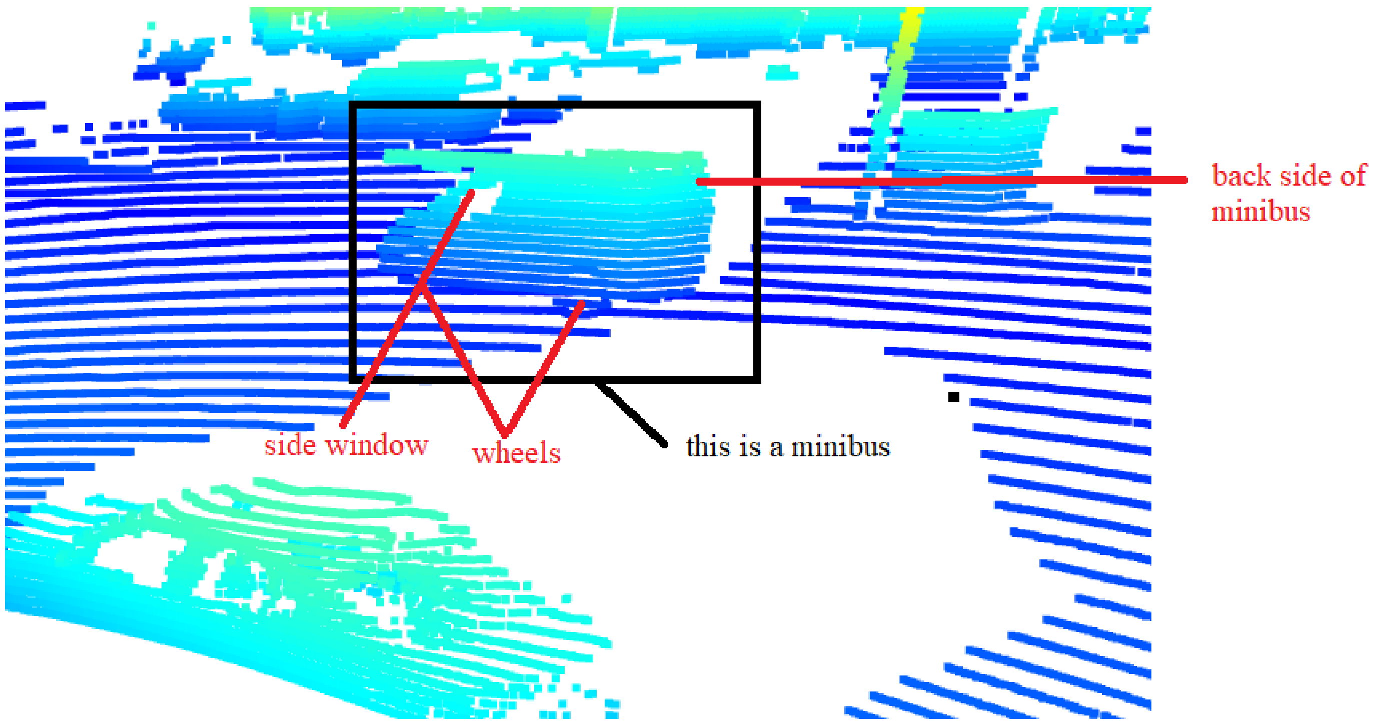
Visualize lidar point-cloud (ID_S1_EX2)

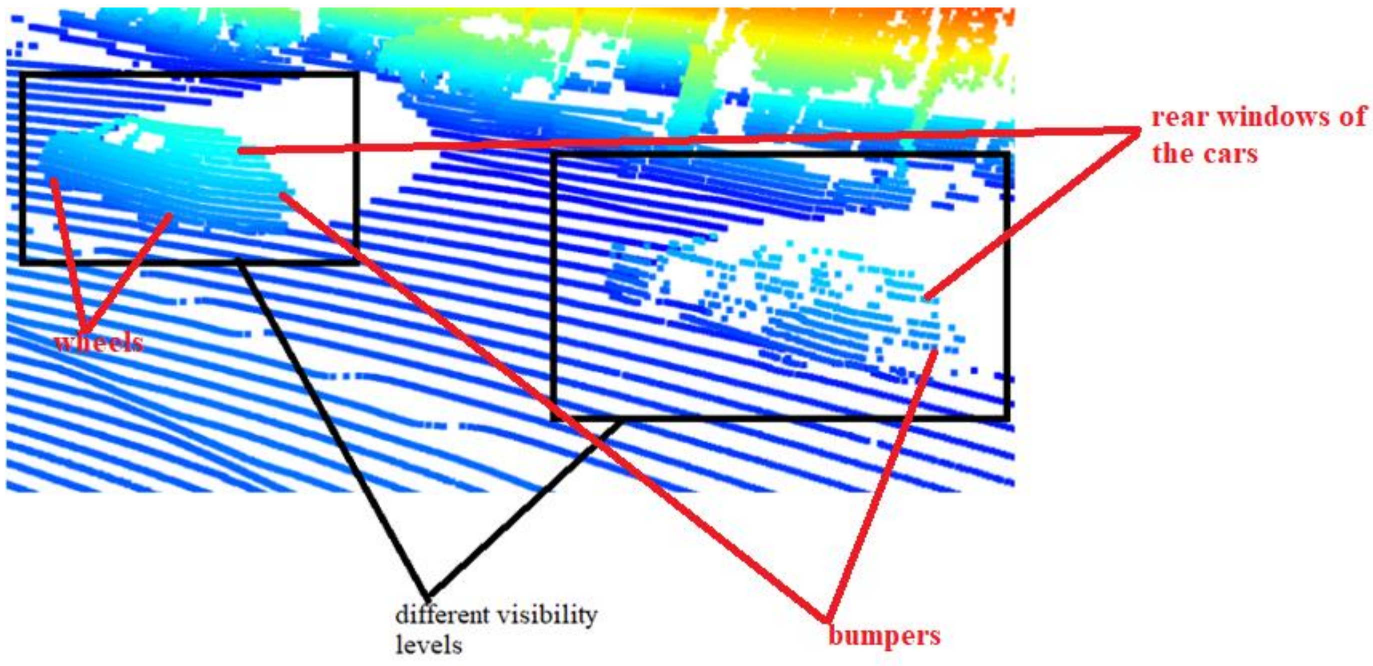
Second part of the task 1 includes operations about displaying lidar point clouds. In following figures some of the cars can be seen with different visibility levels.

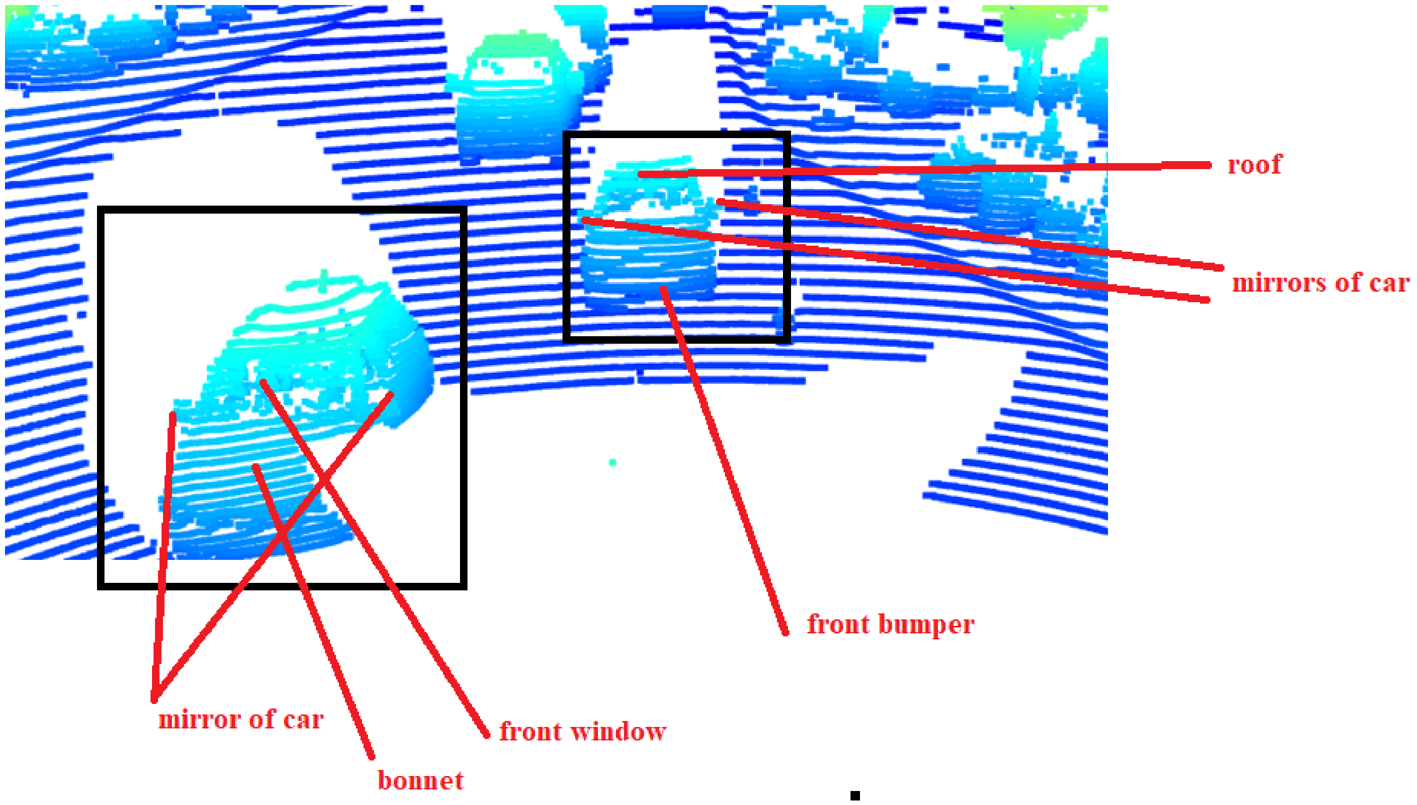


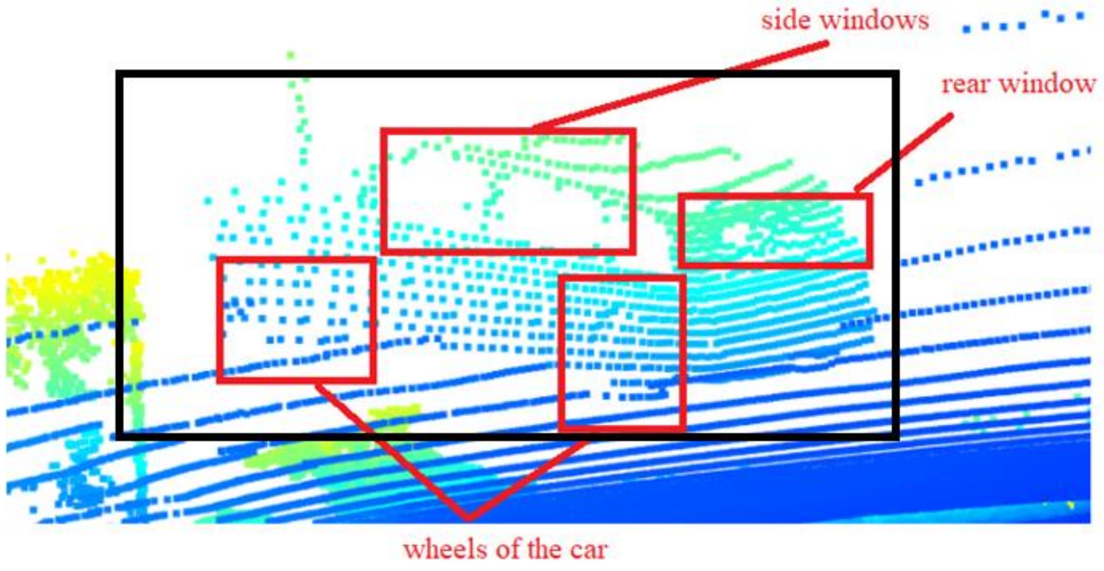

side windows
bottom side of this car is not visible because of the blind spot area
front window
roof

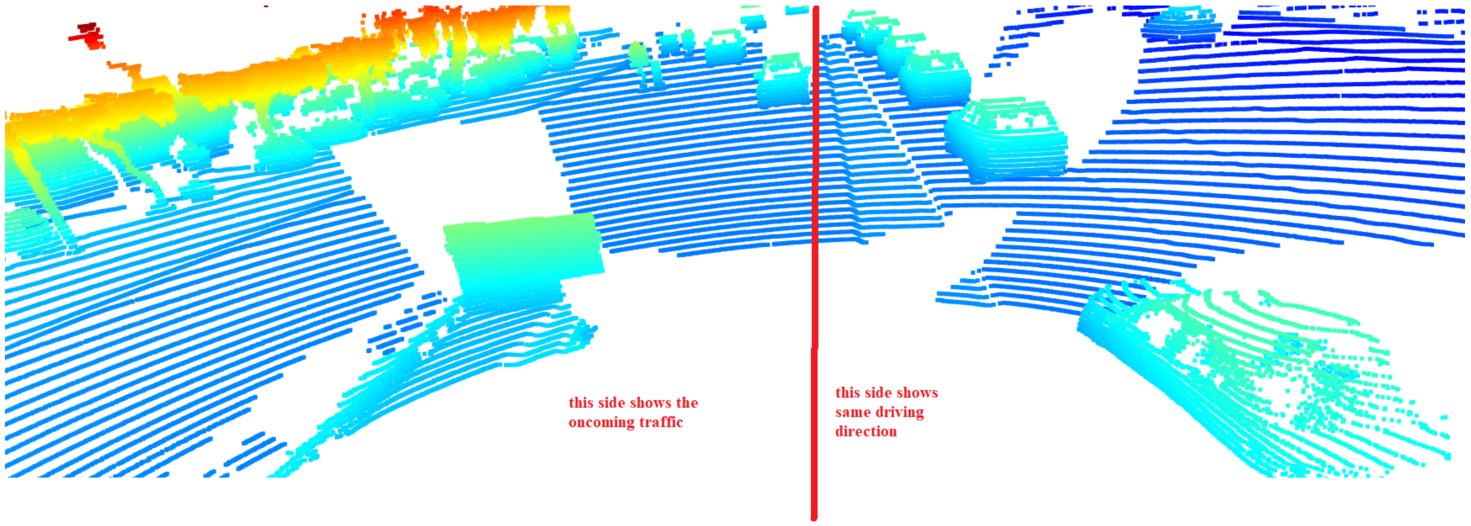

back side of suv
front window
front bumper of suv
here it can be seen that a suv vehicle it can be identified with its back side

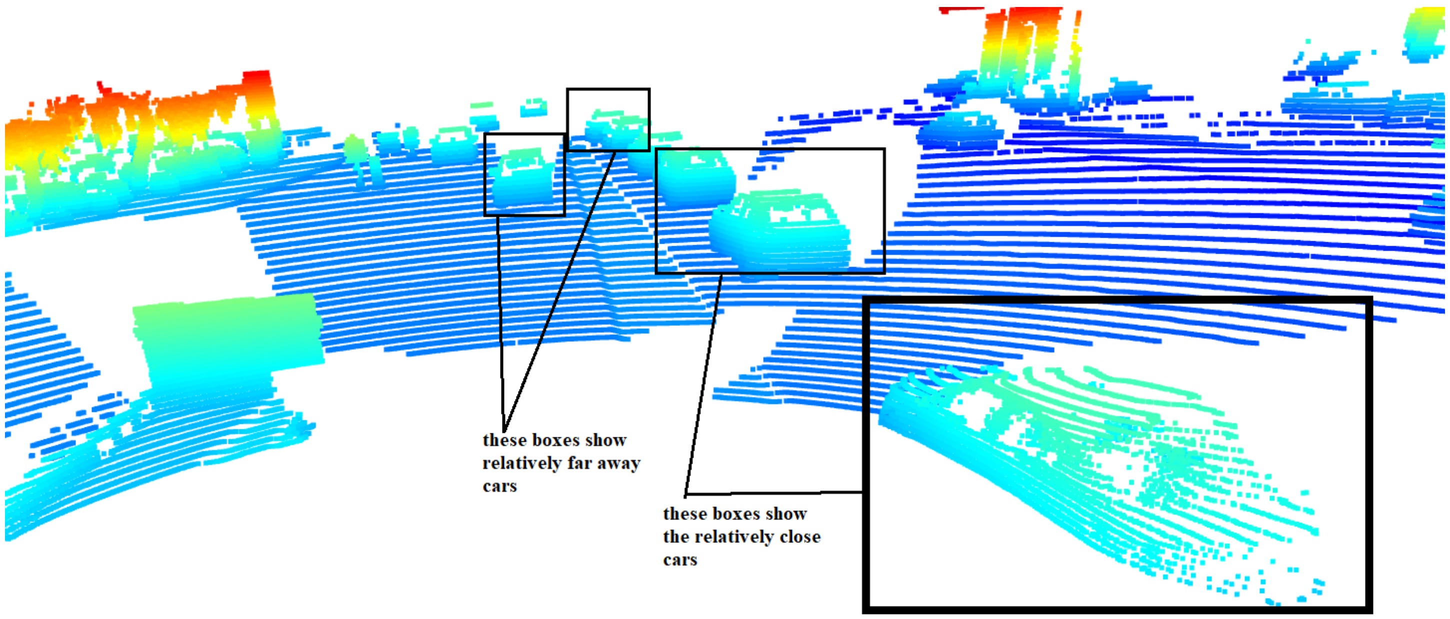

back side of minibus
side window
wheels
this is a minibus

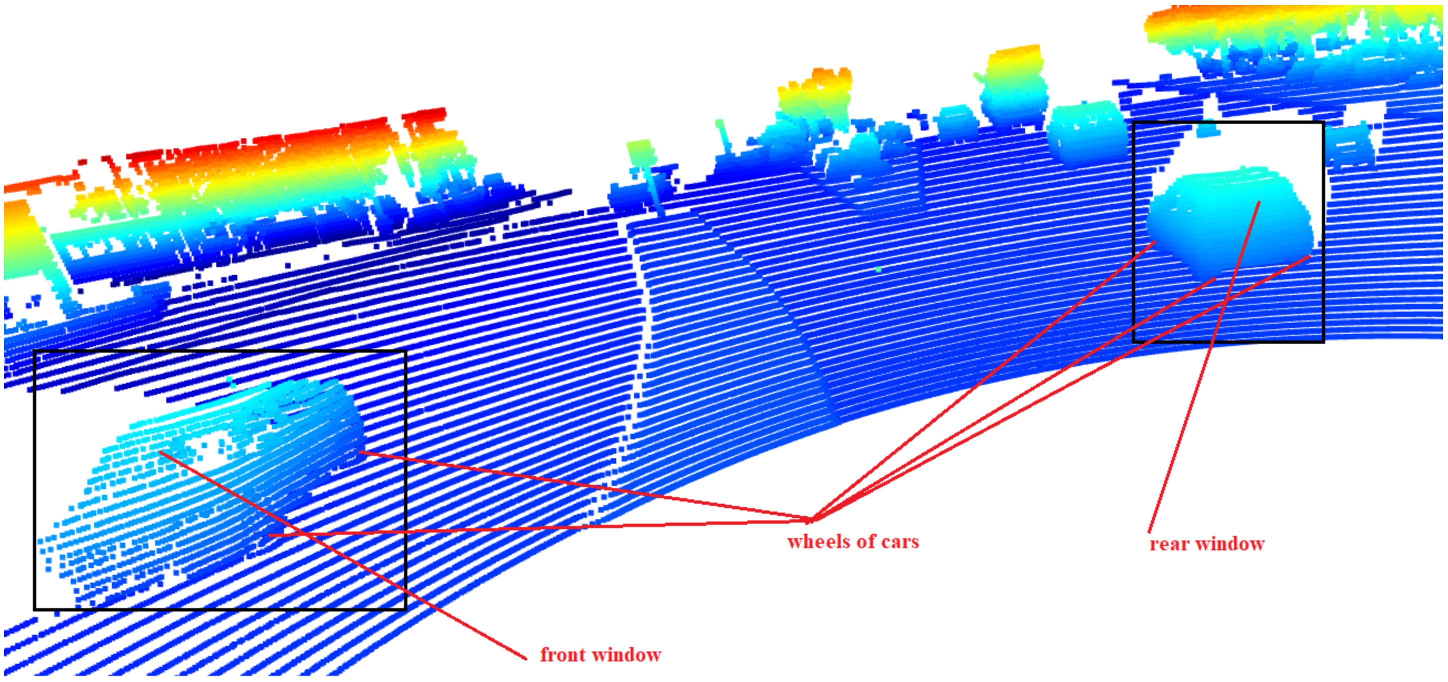

rear windows of the cars
wheels
different visibility levels
bumpers


roof
mirrors of car
front bumper
mirror of car
front window
bonnet


side windows
rear window
wheels of the car


this side shows the oncoming traffic
this side shows same driving direction


these boxes show relatively far away cars
these boxes show the relatively close cars

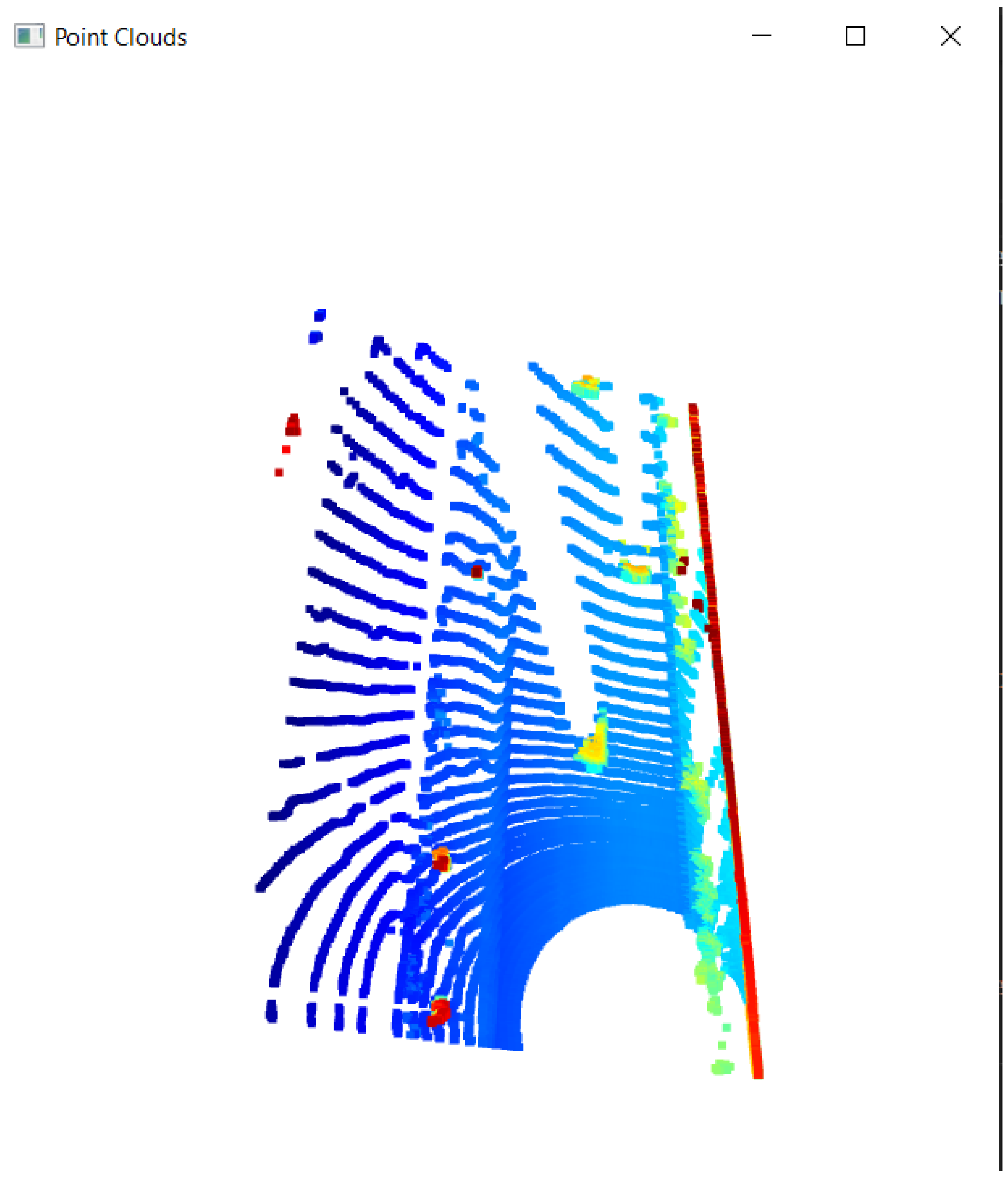


Section 2 : Create Birds-Eye View from Lidar PCL

Convert sensor coordinates to BEV-map coordinates (ID_S2_EX1)

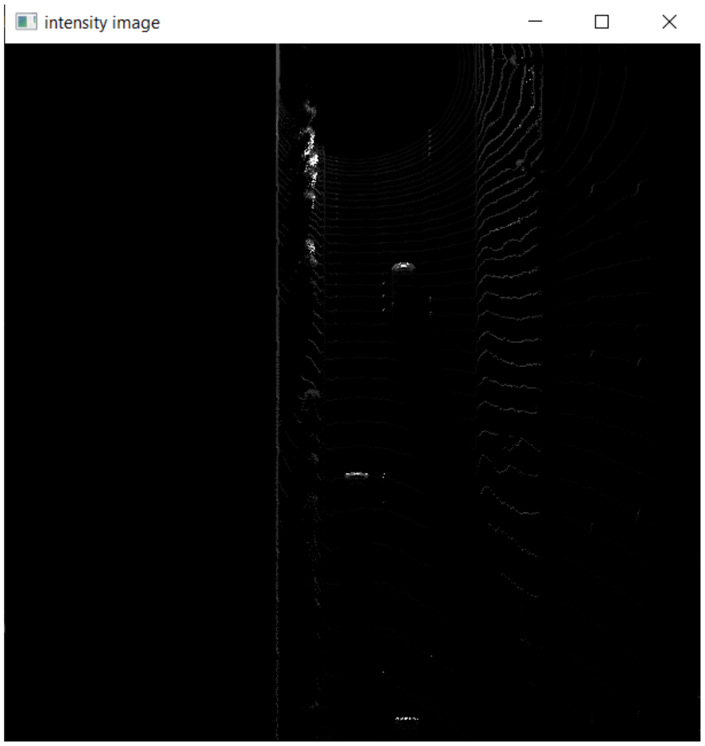
The aim of the tasks in Section 2 is generation of BEV maps from lidar point clouds and then visualize height and intensity channels.

The aim of this task is to perform the first step in creating a birds-eye view (BEV) perspective of the lidar point-cloud. Specifically, based on the (x,y)-coordinates in sensor space, the task requires computing 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.



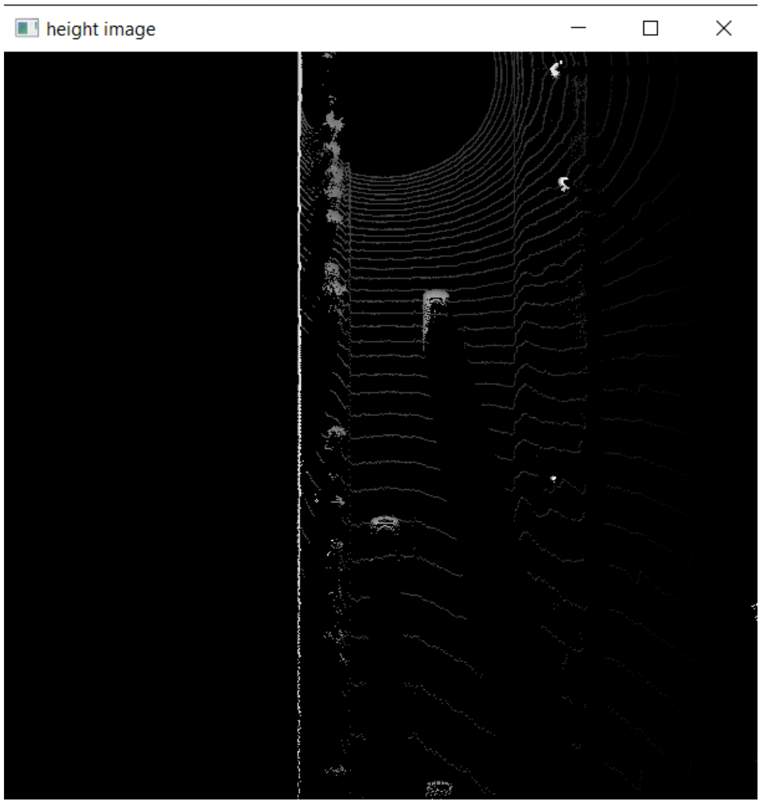
Compute intensity layer of the BEV map (ID_S2_EX2)

The aim of this task is to fill the "intensity" channel of the BEV map with data from the point-cloud. This involves assigning the intensity value of the top-most lidar point to the respective BEV pixel for all points with the same (x,y)-coordinates within the BEV map. Additionally, the resulting intensity image needs to be normalized using percentiles to reduce the impact of outlier values and enhance the visibility of objects of interest. The resulting list of points is also named "lidar_pcl_top" for later use in subsequent tasks.



Compute height layer of the BEV map (ID_S2_EX3)

The aim of this task is to cluster objects in the lidar point cloud by using a clustering algorithm. The resulting clusters will be used in later tasks for object detection and tracking. Specifically, the DBSCAN clustering algorithm is used to group nearby lidar points together into clusters based on their spatial proximity and density. The resulting clusters are then filtered based on their size and height to remove noise and small clusters.



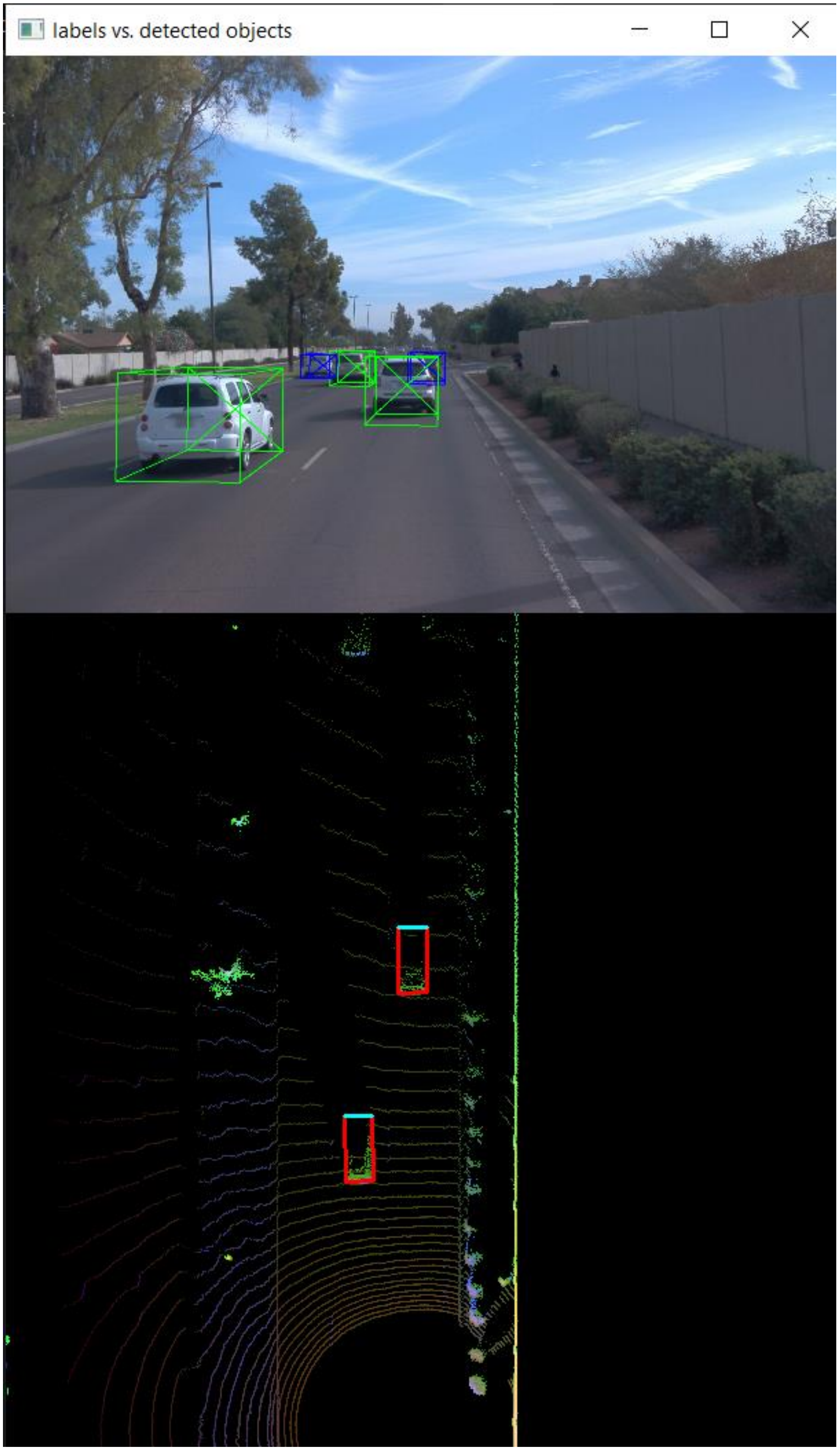
Section 3 : Model-based Object Detection in BEV Image

Add a second model from a GitHub repo (ID_S3_EX1)

The aim of this task is to demonstrate how a new model for object detection in lidar point-clouds can be integrated into an existing framework. The task involves cloning the repository for the Super Fast and Accurate 3D Object Detection based on 3D LiDAR Point Clouds, familiarizing oneself with the code involved in performing inference with a pre-trained model, extracting the relevant parameters and adding them to the configs structure, instantiating the model for fpn_resnet, decoding the output and performing post-processing in detect_objects, and finally visualizing the results by setting the appropriate flag. It should be noted that the pre-trained model and related files have already been integrated into the mid-term project, and the focus is solely on detecting vehicles.

Extract 3D bounding boxes from model response (ID_S3_EX2)

The aim of this task is to convert the detections from BEV coordinate space to metric coordinates in vehicle space, so that they have the format [1, x, y, z, h, w, l, yaw], where 1 denotes the class id for the object type vehicle. The BEV map is a 2D representation of the 3D world, and the object detection algorithm uses this map to identify objects and their properties. However, to use this information for navigation and control of the vehicle, the detections need to be converted into metric coordinates in vehicle space. This involves transforming the coordinates and properties of the detections from the BEV coordinate system to the metric coordinate system used in vehicle space. The resulting format of [1, x, y, z, h, w, l, yaw] is commonly used in object detection tasks in the autonomous driving domain, where 1 denotes the class of the object (in this case, vehicle), and the remaining values represent the position and orientation of the object in metric coordinates in vehicle space. This information can be used for various purposes, such as collision avoidance and path planning.


labels vs. detected objects

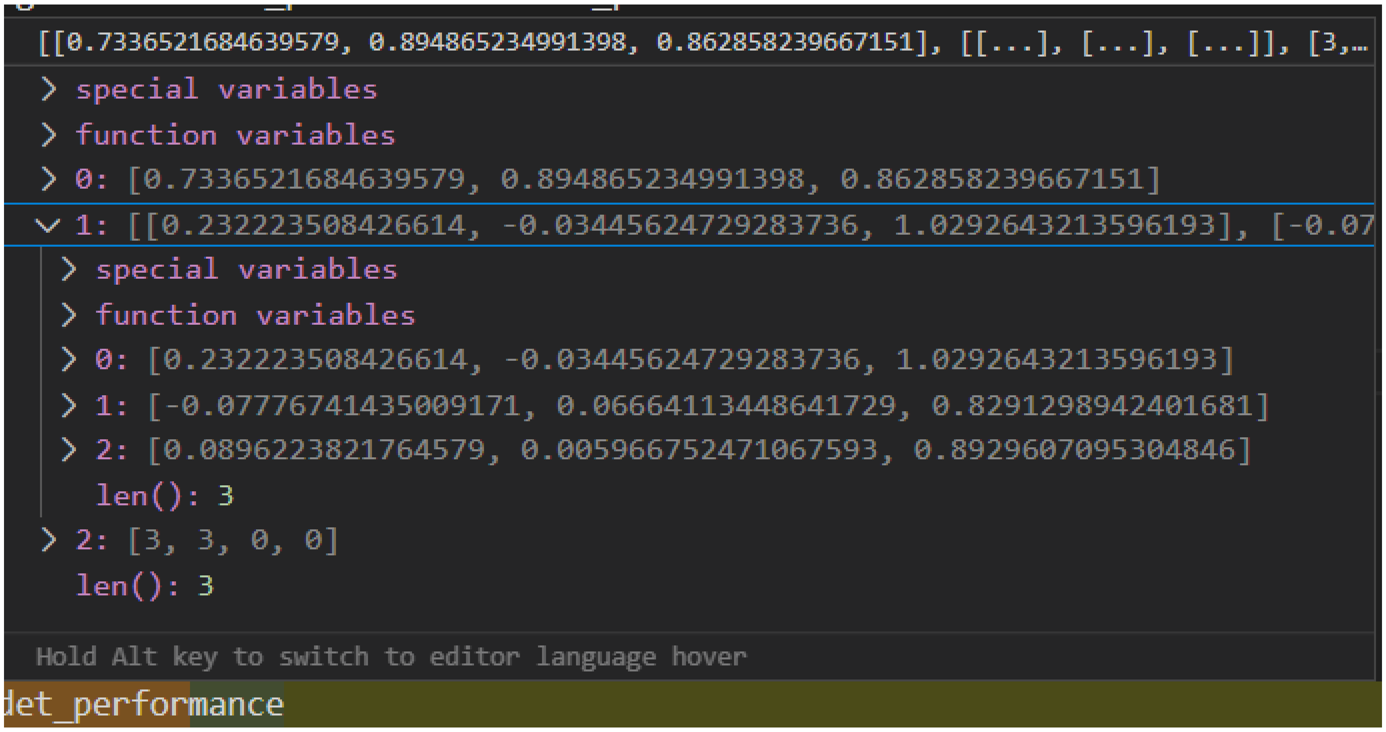
Section 4 : Performance Evaluation for Object Detection

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

The aim of this task is to evaluate the performance of an object detection algorithm by computing the intersection over union (IoU) metric between the ground-truth labels and the detections. The IoU metric measures the spatial overlap between the two bounding boxes and is commonly used in object detection tasks to evaluate the accuracy of the algorithm. By computing the IoU between the ground-truth labels and detections, we can determine whether the algorithm has correctly identified the object, missed the object, or reported a false positive. This information can be used to improve the algorithm and to assess its performance.

Compute false-negatives and false-positives (ID_S4_EX2)

The aim of this task is to evaluate the performance of an object detection algorithm by determining the number of false positives and false negatives for each frame and computing an overall performance measure based on the results. False positives refer to cases where the algorithm reports an object detection that is not present in the ground-truth labels, while false negatives refer to cases where the algorithm fails to detect an object that is present in the ground-truth labels. By computing the number of false positives and false negatives, we can determine the accuracy of the algorithm and identify areas for improvement. The overall performance measure can then be used to compare the algorithm's performance across different frames or datasets.



```
[[0.7336521684639579, 0.894865234991398, 0.862858239667151], [[...], [...], [...]], [3,…
> special variables
> function variables
> 0: [0.7336521684639579, 0.894865234991398, 0.862858239667151]
v 1: [[0.232223508426614, -0.03445624729283736, 1.0292643213596193], [-0.07
  > special variables
  > function variables
  > 0: [0.232223508426614, -0.03445624729283736, 1.0292643213596193]
  > 1: [-0.07776741435009171, 0.06664113448641729, 0.8291298942401681]
  > 2: [0.0896223821764579, 0.005966752471067593, 0.8929607095304846]
    len(): 3
> 2: [3, 3, 0, 0]
  len(): 3

Hold Alt key to switch to editor language hover
det_performance
```

Compute precision and recall (ID_S4_EX3)

The aim of this task is to evaluate the performance of an object detection algorithm using the precision and recall measures based on the accumulated number of positives and negatives from all frames. Precision and recall are two commonly used metrics to evaluate the accuracy of object detection algorithms. Precision measures the fraction of true positive detections among all the detections made by the algorithm. Recall measures the fraction of true positive detections among all the objects present in the ground-truth labels. By computing the precision and recall for the accumulated number of positives and negatives from all frames, we can evaluate the overall accuracy of the object detection algorithm. This information can be used to compare the performance of different algorithms or to assess the performance of the same algorithm on different datasets.

Detection performances of algorithm can be seen in following figure:

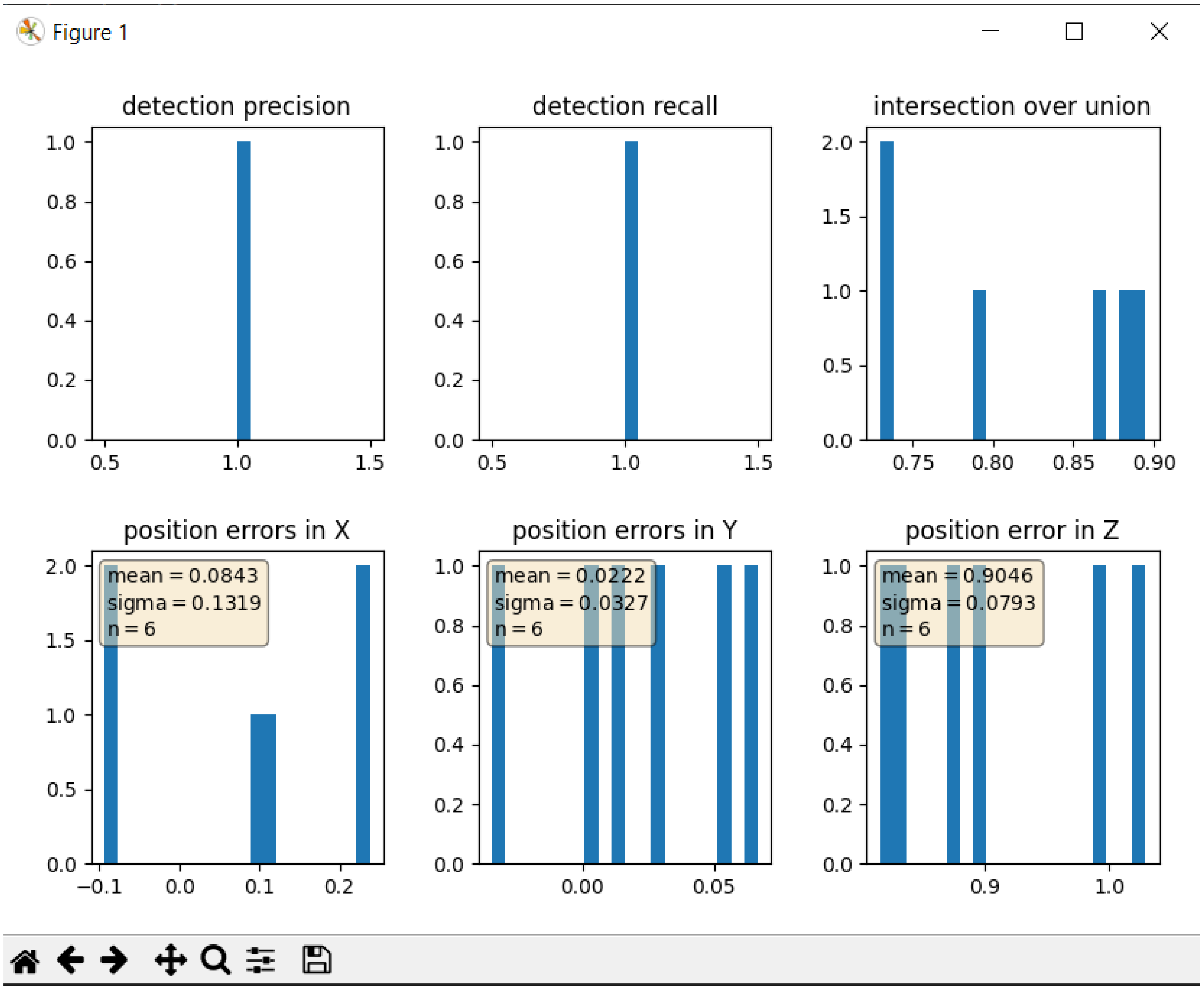


Figure 1
detection precision
detection recall
intersection over union
position errors in X
mean = 0.0843
sigma = 0.1319
n = 6
position errors in Y
mean = 0.0222
sigma = 0.0327
n = 6
position error in Z
mean = 0.9046
sigma = 0.0793
n = 6