Redback Report

Filtering

Down Sampling

For this section, I chose a `voxel_size` of 0.1. This seemed to be the best size as it optimally grouped points together to give shapes in the dataset a more distinctive outline. Too high a voxel value and the datapoints were to sparse. Too low of a value and there are too many points that add unnecessary noise to our pcd and make it hard to perform good filtering with.

Figure 1 voxel_size = 0.3

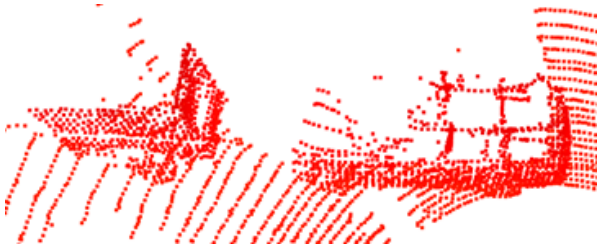

Figure 2 voxel_size = 0.1

Figure 3 voxel_size = 0.000001

Cropping

After Down Sampling we want to crop our pcd to get rid of points that add unnecessary noise. The “remove_statistical_outlier” removes points that are further away from their

neighbours compared to the average for the point cloud. By varying our 2 input parameters:

- `nb_neighbors`, which specifies how many neighbours are taken into account in order to calculate the average distance for a given point.
- `std_ratio`, which allows setting the threshold level based on the standard deviation of the average distances across the point cloud.

In this case I had to apply semi-aggressive outlier filtering in order to get rid of the points sitting under the road plane as well as the points sitting far away from our origin. I was careful not to make it too aggressive as it would start cropping out the points representing objects like cars and trucks, thus distorting their shape and making our later segmentation and clustering less effective. The values I settled for were `nb_neighbors` = 50, `std_ratio` = 0.1

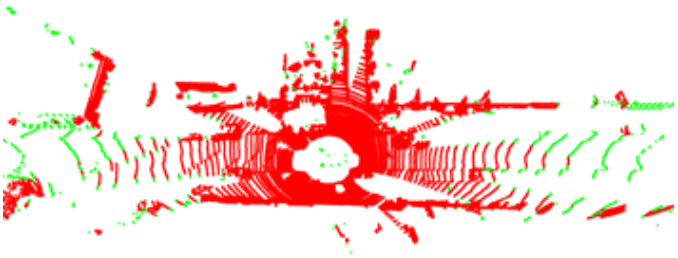

Figure 4, nb_neighbors=50, std_ratio=1

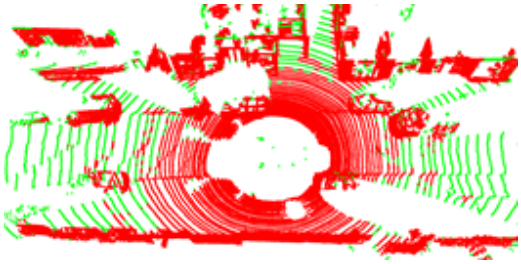

Figure 5, nb_neighbors=50, std_ratio=0.1

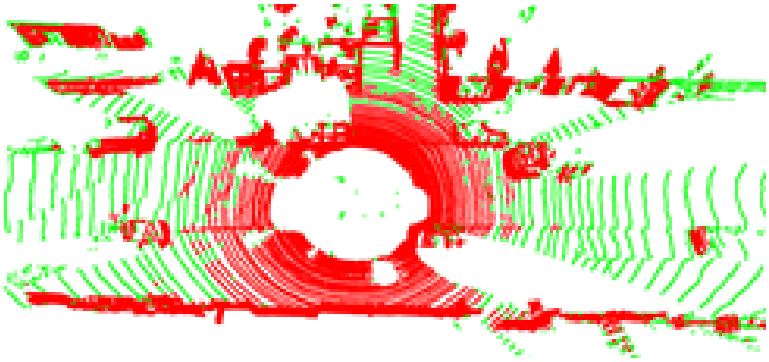

Figure 6, nb_neighbors=50, std_ratio=0.01

Segmentation

For Segmentation we are trying to balance time and reliability of our code. Using the RANSAC algorithm we can alter 3 different parameters namely `distance_threshold`, `n`, and `num_iterations`. We ideally want to have our `distance_threshold` value equal to or greater than the value of our voxel size. This value allows us to safely capture more points in our ground plane but not too high as to also include points from non-ground plane objects like cars. Thus we can reduce our `num_iterations` which greatly increases the speed of our program. In the end I settled for values `distance_threshold` = 0.2, `n` = 30, `num_iterations` = 800

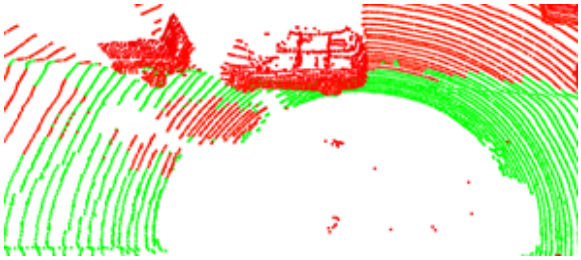

Figure 7, RANSAC, distance_threshold = 0.08, n = 10, num_iterations = 2000

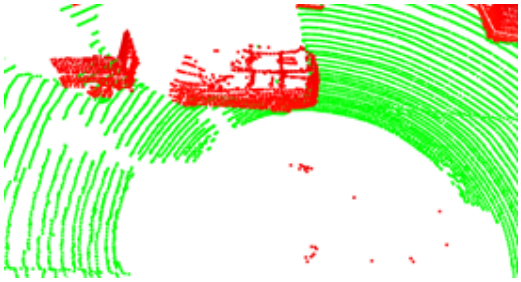

Figure 8, RANSAC, distance_threshold = 0.2, n = 30, num_iterations = 800

Clustering

With our pcd cleaned up it's time to cluster our points together to isolate distinct objects. For this I used a clustering algorithm, DBSCAN, that is a density based clustering algorithm. The algorithm requires two parameters:

- `eps` defines the distance to neighbours in a cluster
- `min_points` defines the minimum number of points required to form a cluster

As we can see, DBSCAN clusters these points together to isolate objects in our data. Another neat things is that it will also label any isolated points as noise values such as some ground values that weren't 100% removed in our segmentation part (These are the black coloured points). Our `eps` value is the main driver of how many points are included in our cluster; too low and we start eating into our cluster car objects; too high

and we can't get rid of some missed outliers. Thus the values I chose for this was `eps` = 0.5

When it comes to our `min_points`, we can assume that our segmentation filters out all of the ground plane BUT in the case that there are some ground values left over we know that the clusters of these points will be quite small in size so having a larger `min_points` value will ensure they are removed. Now having a large value may eat into the car clusters themselves but since the cluster of these objects are much bigger, it does very little to impact its overall shape so we are justified. Thus the value I chose for this was `min_points` = 20

Change in `eps`

eps=0.3, *min_points*=20, *print_progress*=True

eps=0.5, *min_points*=20, *print_progress*=True

Change in `min_points`

eps=0.5, *min_points*=5, *print_progress*=True

eps=0.5, *min_points*=40, *print_progress*=True