Plateforms and Algorithms for Autonomous Driving-Module 1

1.Euclidean clustering object detection

Asif Khan Pattan

Masters in Artificial Intelligence

University of Bologna

August 28, 2023

Objective

The objective of the assignment is to find and segment the individual object point clusters lying on the plane

1. Down sample the point cloud
2. Segment the ground plane
3. Create a KD-tree representation for the input point cloud dataset
4. Compute Euclidean distance to build the cluster list
5. The algorithm terminates when all points have been processed and are now part of the list of point clusters

Solution

TODO 1: Downsample the dataset

Voxel Filter is used to downsample the given point cloud. A pcl::VoxelGrid filter is created with a **leaf size of 10cm**, the input data is passed, and the output is computed and stored in cloud_filtered.

PointCloud before filtering: 120854 data points (x y z).

PointCloud after filtering: 51083 data points (x y z).

TODO 3: Segmentation and apply RANSAC

Segmentation of ground/road from the vehicles, pedestrians and other objects can be segemented. Random sample consensus (RANSAC) is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers The algorithm groups the data into inliers and outliers

Inliers are the points that represent the road

Outliers are the points that represent the objects lying on the road.

seg.setDistanceThreshold (0.2) is an important parameter which decides if a point in the point cloud belongs to a cluster.

A value of 0.2 represents 20cms. So points that are in 20cms range to one another forms a cluster.

0.2 is used for dataset 1 and a value of 0.6 is set for dataset 2

TODO 4: iterate over the filtered cloud, segment and remove the planar inliers

The code first checks if the size of the filtered point cloud is greater than 30% of its original size. If it is, the code enters a while loop. Inside the loop, the seg object, which is an instance of pcl::SACSegmentation<pcl::PointXYZ>, is used to segment the largest planar component from the remaining point cloud. The seg.setInputCloud (cloud_filtered) method sets the input cloud for the segmentation object, and the seg.segment (*inliers, *coefficients) method performs the segmentation and stores the inliers (points that belong to the planar surface) in the inliers object.

If no inliers are found, the code prints a message to the console and exits the loop. Otherwise, an instance of pcl::ExtractIndices<pcl::PointXYZ> is created and used to extract the planar inliers from the input cloud. The extract.setInputCloud (cloud_filtered) method sets the input cloud for the extraction object, and the extract.setIndices (inliers) method sets the indices of the points to be extracted. The extract.setNegative (false) method specifies that only the points with indices specified by setIndices should be extracted.

The extract.filter (*cloud_plane) method extracts the points associated with the planar surface and stores them in the cloud_plane object. The code then prints a message to the console indicating how many points were extracted.

Next, the extract.setNegative (true) method is called to specify that all points except those with indices specified by setIndices should be extracted. The extract.filter (*cloud_f) method

extracts these points and stores them in the cloud_f object. The filtered point cloud is then updated to contain only these points by assigning *cloud_filtered = *cloud_f.

The while loop continues until either no more inliers are found or until less than 30% of the original points remain in the filtered point cloud. At this point, all planar surfaces have been separated from the point cloud.

TODO 5: Create the KDTree and the vector of PointIndices

This code creates a KdTree object and sets it as the search method for a EuclideanClusterExtraction object, which is then configured to perform Euclidean clustering on the filtered point cloud.

Minimum Cluster size is set to 20 points and Maximum cluster size is set to 50000 points.

TODO 6: Set the spatial tolerance for new cluster candidates (pay attention to the tolerance!!!)

Cluster tolerance is set to 0.4 for dataset 1 and 0.6 for dataset 2

This means that points that are within a distance of 0.4/0.6 units from each other will be considered part of the same cluster.

Self defined "Eluclidean clustering function" is used instead of PCL functions.

The function uses the Euclidean clustering approach(based on given instructions) to build clusters of points in the point cloud. The clusters are built by iterating over every point in the point cloud and checking if it has already been visited. If the point has not been visited, the proximity function is called to find the cluster of points that are within the specified distance tolerance of the current point.

TODO 7: render the cluster and plane without rendering the original cloud

cloud_cluster which is the clusters are rendered instead of the whole unsampled point cloud for visualisation.

Results

The code is executed on both the datasets i.e., dataset_1 and dataset_2. As seen in the following figures the code was successful in capturing the individual objects into single cluster and then render box around the objects due to proper setting of **setDistanceThreshold** and **clusterTolerance**.

Dataset 1 rendering has got an average FPS of 130 to 335 while Dataset 2 rendering has got an average FPS of 75 considering the downsampling with leaf size of 0.1, number of clusters in the frame etc..

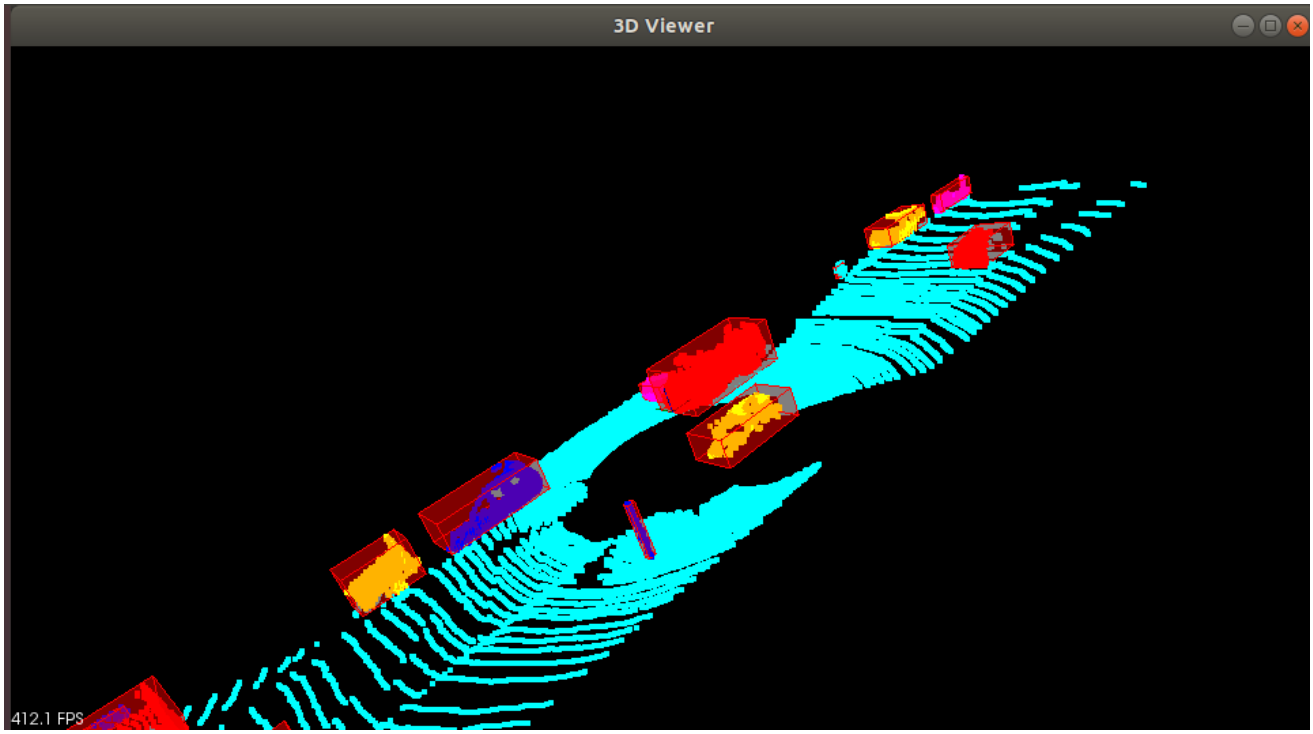

Figure 1: Dataset 1 Clustering object detection

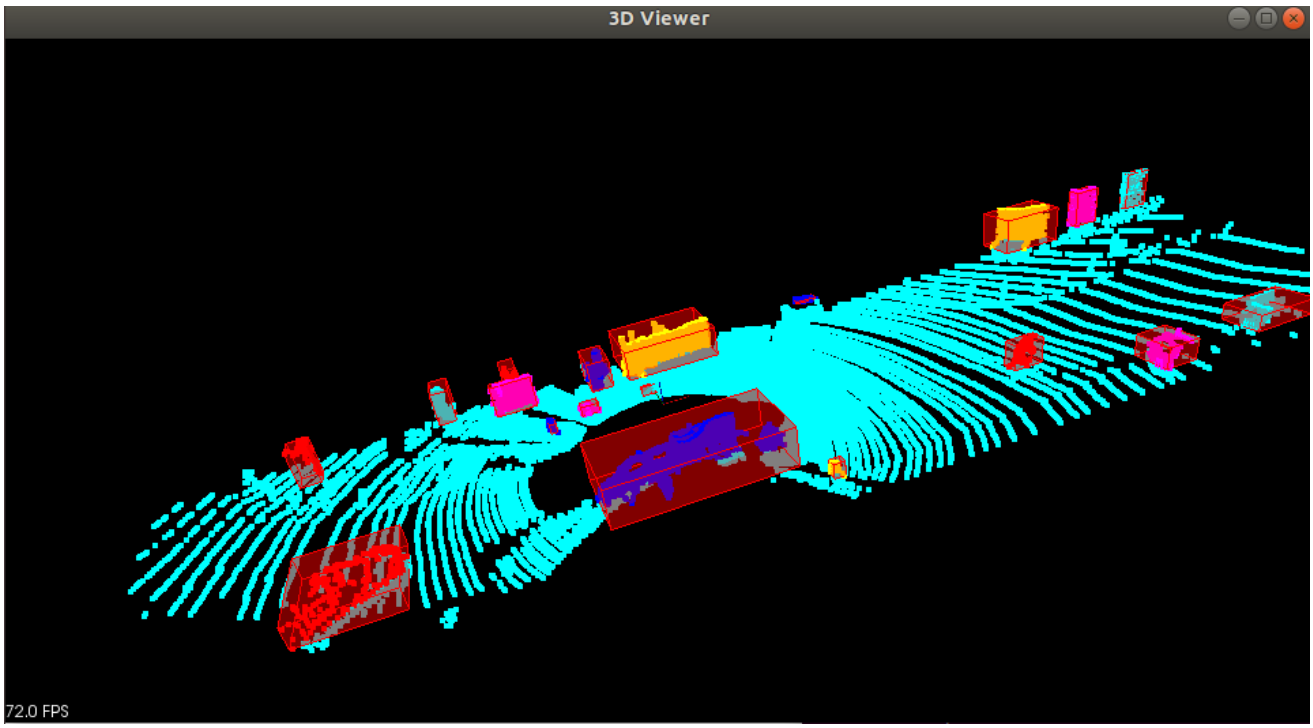

Figure 2: Dataset 2 Clustering object detection