Object Detection in Point Cloud: Poles

Geospatial Vision & Visualization

Jordan Haskel, Levi Todes, Ethan Park, Yuchen Wang

Introduction

Object detection in Point Cloud is popular in HD Map and sensor-based autonomous driving.

There are basically four types of object you can obtain in daily scenario:
Road surface - painted lane marking and pavement area
Support facility - road boundary, road sign, **light pole**, etc.
Uncorrelated object - sidewalk, building, etc.
Moving objects - pedestrian, vehicle, bicycle, etc.

The goal of this project is to detect light poles.

Methodology

Python Open3D library could be used to create point cloud objects and implement segmentation/filtering.

Raw point cloud data files should be processed by converting latitude-longitude-altitude coordinates to X-Y-Z cartesian coordinates.

Sklearn.cluster is for segmenting light poles from overall background.

Algorithms

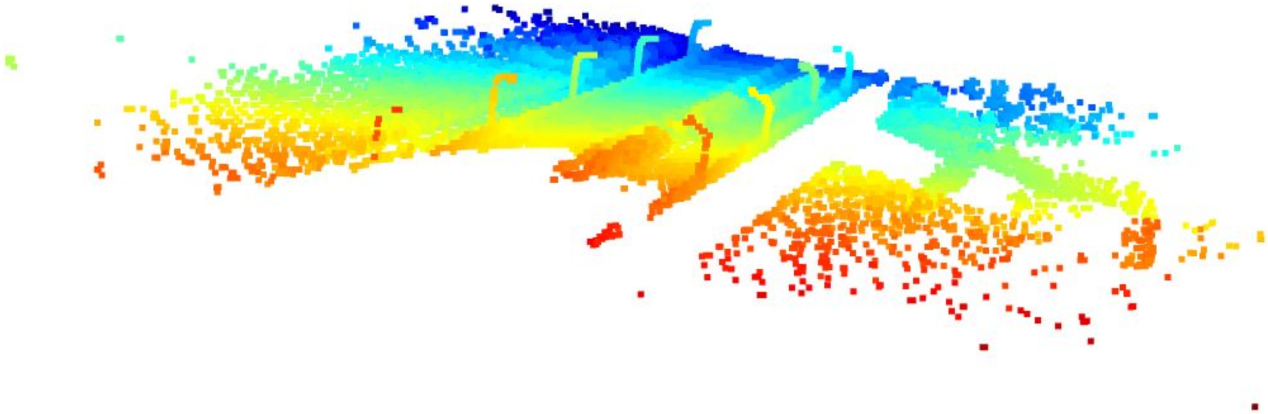
Preprocess Raw Data

The raw “point_cloud.fuse” file is in csv format containing latitude, longitude, altitude and intensity of each point cloud. The function takes the geographic coordinates and converts them into cartesian coordinates, as well as passing the intensity data. Then all XYZ points data are stored in file “All_data.obj”.

open3d.visualization.draw_geometries([point_cloud]) displays the 3D point cloud data.

Algorithms

Visualization



Algorithms

Downsample Point Cloud

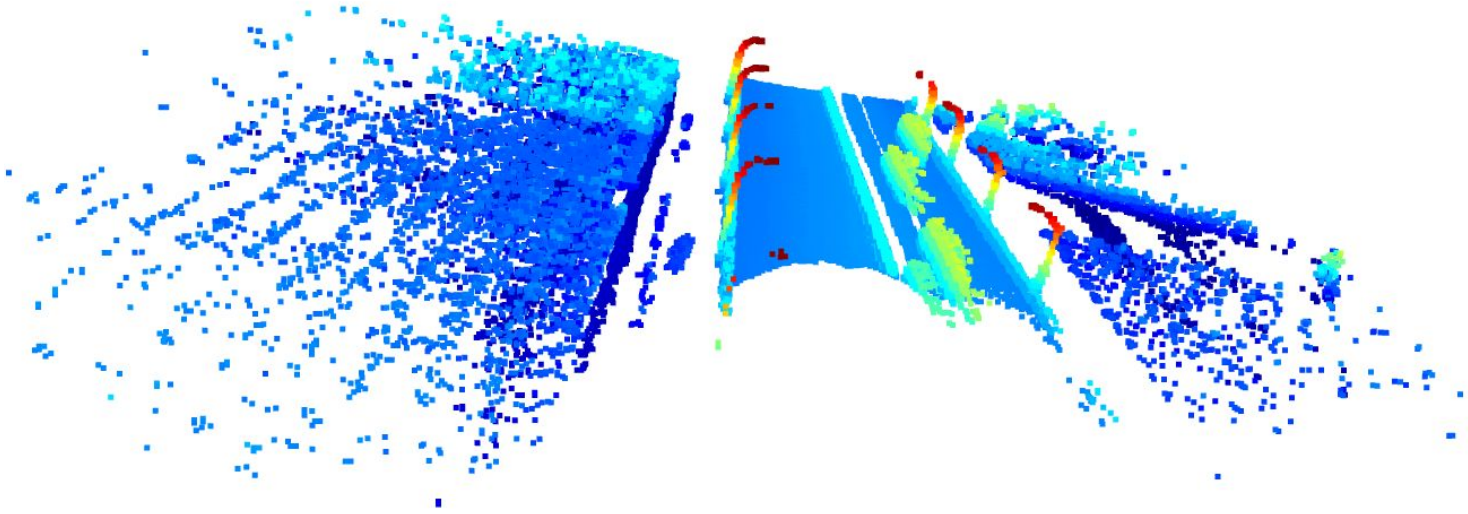
In order to reduce the number of points waiting for processing, we downsampled the input point cloud.

Set the vortex size as 0.8 and each occupied voxel generates exact one point by averaging all points inside.

Removed Statistical outliers. Took Number of points from 1292208 to 39630.

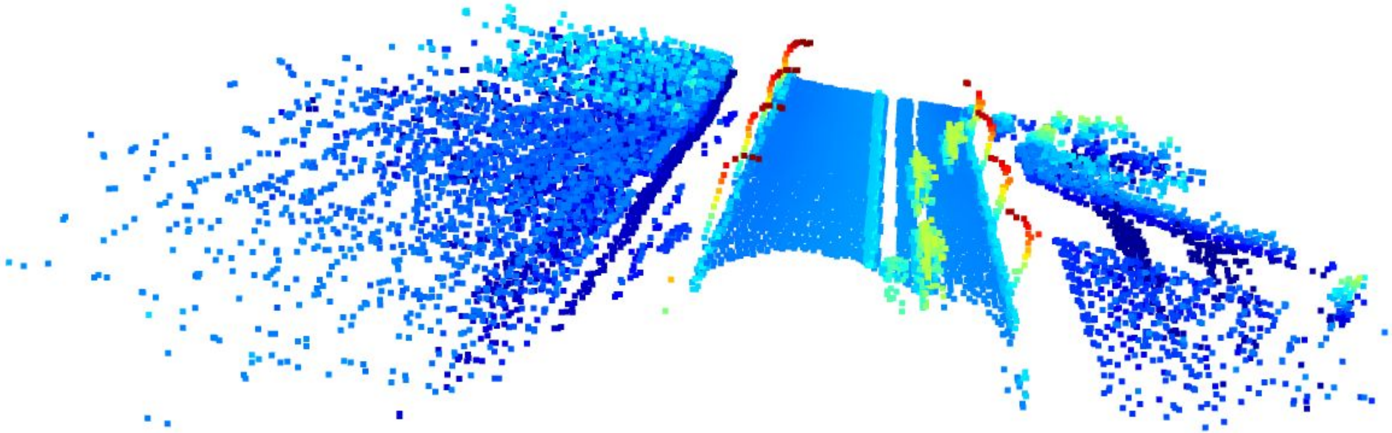
Algorithms

Before



Algorithms

After



Algorithms

Planar Filtering

Convert the point clouds into grid. For each set of points that lie on a vertex of two dimensions, that variation in the third dimension could be used to filter data points.

For this project, we snapped points in X-Y plane and used the variation in Z axis to filter out data points that don’t have a large variation in Z (aka. Not high enough).

Algorithms

Clustering: Separate the point cloud points into 2 clustering by their altitude.

sklearn.preprocessing() normalizes the XYZ point data(default as L2-norm)
sklearn.KMeans(n_cluster).fit_predict(point[z]) splits the data into **n** clusters based on their Z value which refers to the altitude

Algorithms

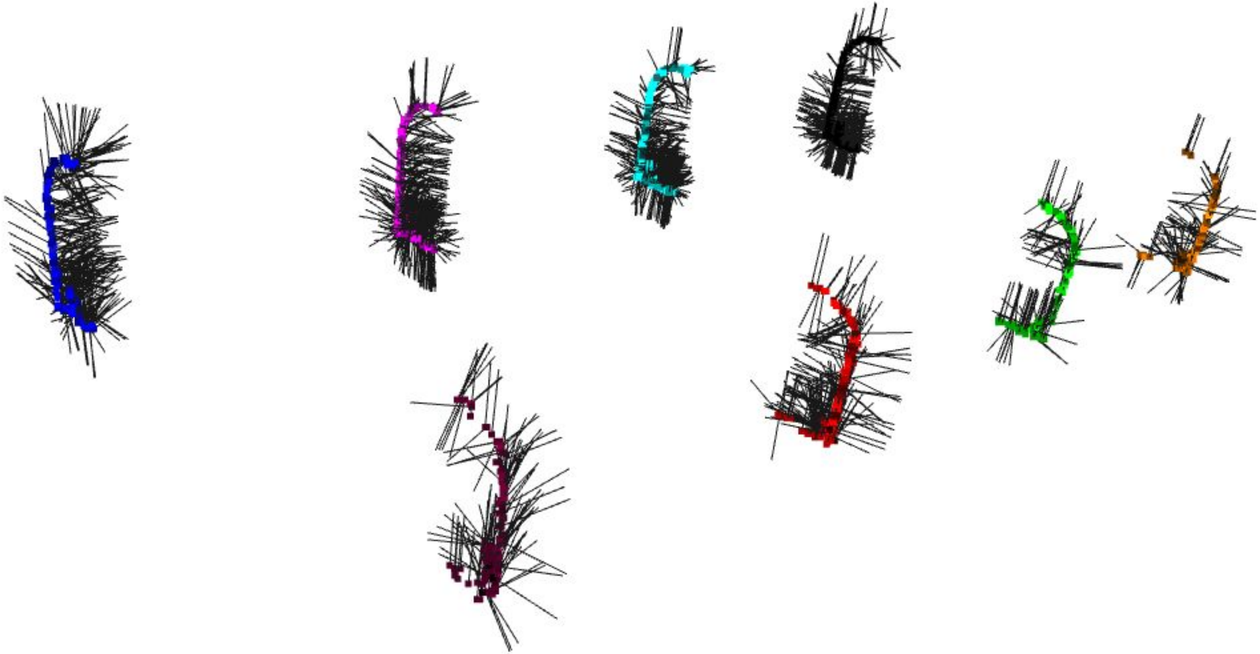
Clustering based on number of poles

Some more K-means clustering is done now using the knowledge of how many correct clusters (poles) there are in the cloud.

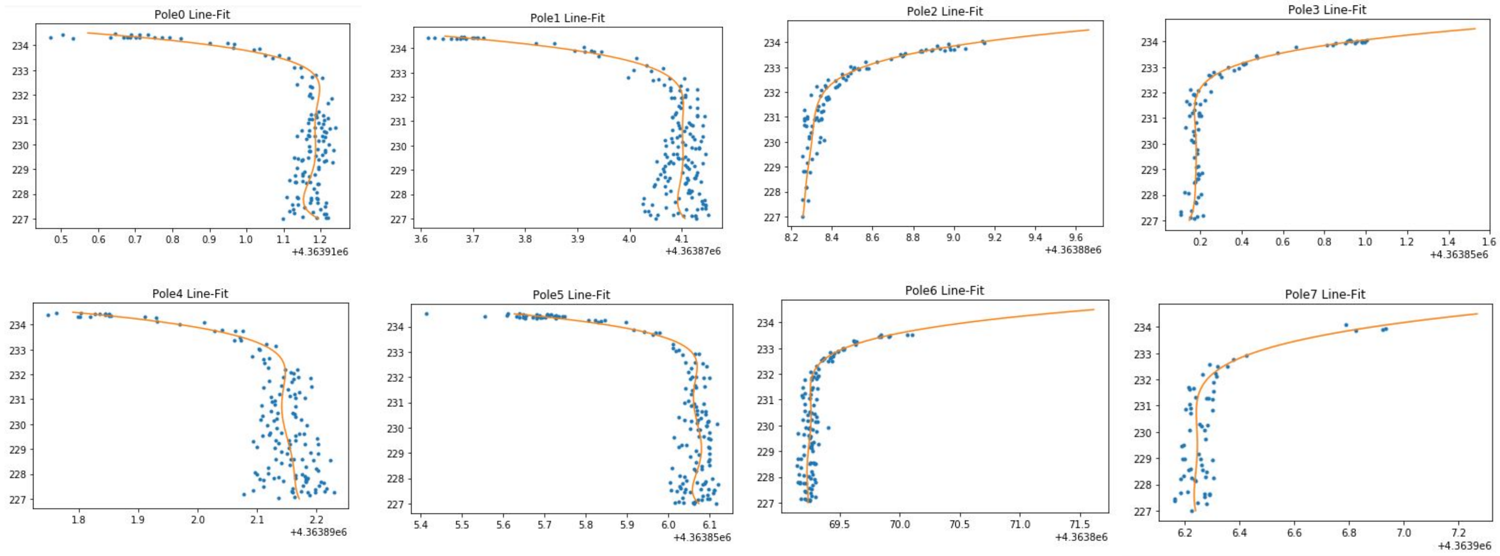
Clustering is done here on only x and y coordinates.

Result

Result



Result



Pole0 Line-Fit
+4.36391e6
Pole1 Line-Fit
+4.36387e6
Pole2 Line-Fit
+4.36388e6
Pole3 Line-Fit
+4.36385e6
Pole4 Line-Fit
+4.36389e6
Pole5 Line-Fit
+4.36385e6
Pole6 Line-Fit
+4.3638e6
Pole7 Line-Fit
+4.3639e6

Conclusion

- Successful pole detection and visualization
- Used downsampling and clustering on preprocessed data to isolate poles
- Resulting pole data enables other useful calculations

Future Work

- The first next step to take would be to test our algorithm on more point clouds - to test robustness to similar shapes in the cloud such as trees and signposts.
- Inquest into more efficient way to find whole pole - currently the top curve of the pole has one or two points missing.
- Robust pole detection could further lead to classification of different pole types

References

- LLA to XYZ: https://stackoverflow.com/questions/8981943/lat-long-to-x-y-z-position-in-js-not-working
- Elbow Method (Clustering): https://en.wikipedia.org/wiki/Elbow_method_(clustering)
- Point Cloud Lecture by David: https://www.youtube.com/watch?v=SJWJNbqynFY&feature=youtu.be
- Open3d Python-Api: http://www.open3d.org/docs/index.html