Proceedings of the 11th International Conference on Applied Informatics
Eger, Hungary, January 29–31, 2020, published at http://ceur-ws.org

FFT Based Airborne LIDAR Classification with Open3D and Numpy/Scipy

Peter Szutor

University of Debrecen
Faculty of Informatics
Hungary
`szppaks@gmail.com`

Abstract

The classification of the airborne LIDAR (also referred to ALS) point-clouds is a frequently used operation. The recently used algorithms use machine learning algorithms (like SVM), but teaching this algorithms often may be involved. In many instances we need to write a special application for working with our data. For the fast developing we need a very useful programming language (python+numpy is the best), and a C based point cloud library, for the fast execution. Open3d is a modern library for 3D data processing, including point clouds,meshes, RGBD pictures, it has good highly optimized algorithms, data types, IO functions and supports visualisation; and it has integration with numpy arrays, so we can easily use scipy and scikit modules. We present a new method for point cloud classification, with two clustering methods and Fast Fourier Transform. The idea is based on that the elevation in the point cloud clusters has different frequency domain depending on thesurface; the buildings shows wider spectrum then the vegetation, and the ground has a very small spectrum. To make contiguous parts from the entire point cloud, we use DBSCAN clustering (Open3D), and BIRCH clustering(scikit-learn).

Keywords: Open3d, LIDAR, point cloud classification

1. Algorithm

1.1. Introduction

The modern airborne laser scanners can save the waveform data for the points, and captures more information from the scanned object ([1]). This data determine what is the object (tree or building). In this case , we can use the point based classifications. These classifications are simple and efficient; but if we have a point cloud without reflectance values, we cannot use these ones.

There are numerous algorithms, which can classify the point clouds with only the x, y, z coordinates ([2, 3]). These algorithms are based on the decision of the height above a calculated ground level, for example The BlueMarble Geo classification (that examines the difference between the non ground points) and the Lastools (that make contours from points of the same height).

The third class of the algorithms is the machine learning based algorithm, such as the well-known software CloudCompare's plugin, the CANUPO ([4]), that uses SVM and LDA to classify the points. If you want to use these algorithms, you have to train the classifier on a training set.

1.2. The FFT based algorithm

To overcome the issues mentioned above I developed an algorithm based on FFT (Fast Fourier Transform). My classifier has six steps:

- DBSCAN clustering
- BIRCH clustering
- Compute the ground levels on a grid
- Defining the ground clusters
- Compute FFT on the non-ground clusters
- Classifying by the FFT values

Scikit ([6]), the popular python library offers 10 different general clustering method. We can use the methods where do not need to specify the number of clusters, so the K-means, Spectral, Ward algorithms are out of the box. During testing these methods the Agglomerative and Meanshift were found inappropriate, because they cannot handle great number of points. The fastest method was the DBSCAN ([5, 7]), and the BIRCH ([9]) worked well too. I have tried the Hierarchical DBSCAN ([8]), the advanced version of the DBSCAN, but It have made too small clusters.

Why the FFT? The FFT is a very fast algorithm, and in this case it's practical to use fast methods because of the huge number of points (for example I can use an algorithm based on standard deviation of the normals on the mesh created from the cluster points, but it's too compute-intensive)

1.2.1. DBSCAN clustering

The DBSCAN algorithm considers clusters as areas of high density separated by areas of low density. Due to this rather generic view, clusters found by DBSCAN can be of any shapes, as opposed to k-means which assumes that clusters are convex-shaped. It's surprisingly fast and good, but clusters do not form continuous surfaces. This algorithm has two input parameters, eps (distance value, two sample minimum distance) and minimum samples (how many samples make new cluster). On my point cloud sample I used eps=1.9 and minimum samples=30 - this settings made the best result. (The 3D view generated by the Open3D visualiser ([5]))

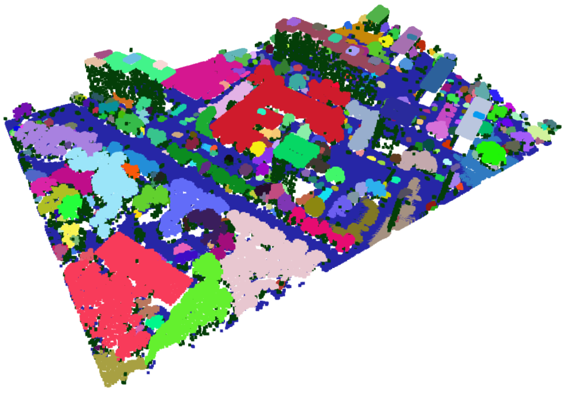


Figure 1: DBSCAN generated clusters

Assigning random colors to the generated clusters we can see there are clusters with points belonging to buildings and vegetation; therefore these clusters have to be divided into smaller ones if the clusters point density differ significantly from the whole point cloud or it has too many points. So I need use another clustering method based on nearest neighbour point distances. The used DBSCAN is an Open3D built-in point cloud method, simple to use and fast. I made a test with HDBSCAN (Hierarchical DBSCAN, az advanced version of the DBSCAN), but this method formed two many small clusters.

1.2.2. BIRCH clustering

The Birch builds a tree called the Clustering Feature Tree (CFT) for the given data. The Birch algorithm has two parameters, the threshold and the branching factor. The branching factor limits the number of subclusters in a node and the threshold limits the distance between the entering sample and the existing subclusters. This method is not a suitable procedure for divide the whole point cloud, but it can cut the big clusters after the DBSCAN. On my sample I used the following parameter values: branching factor 50, nclusters None, threshold 3.9, compute labels True.

This procedure is the part of the scipy.clustering.

The disadvantage of these methods are that make too small clusters, and leave non classified points. This problem leads to a failure in the FFT computing.

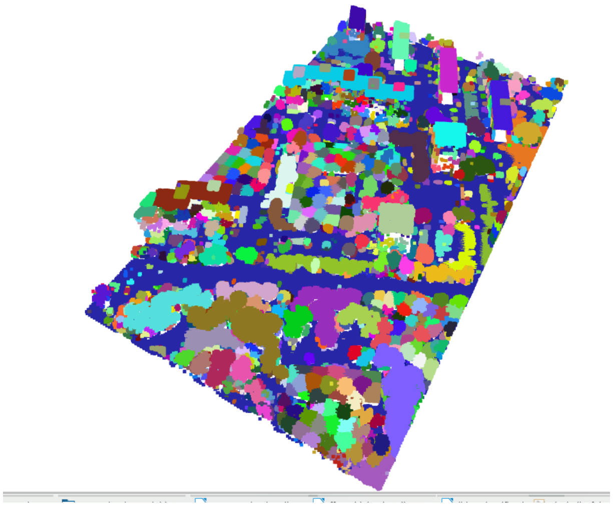


Figure 2: Birch generated clusters

1.2.3. Compute the ground levels on a grid

Decisioning the ground level is simple: I segment the whole point cloud in an axis oriented regular grid. The smallest Z value in a segment will the ground level, and the x,y,z coordinate will be the minimum point.

1.2.4. Defining the ground clusters

I set the cluster to ground cluster when the difference between the closest minimum point elevation (Z value) and the average elevation of the cluster's points is less then 0.4 meters, and the Z range of the cluster is less than 2 meters.

1.2.5. Compute FFT on the non-ground clusters

Computing FFT is a fast method, but it has a little handicap: this runs on values are forming a regular grid. But the clusters made from the LIDAR point cloud have points located randomly. Thus I have to generate a 100*100 regular grid from the points using an interpolation method; this is a disadvantage of my classifying algorithm, because this step requires lots of computation. I use the scipy LinearNDInterpolator module to make the grid. This interpolation is based on the Quickhull algorithm, and generates a Delaunay triangulation, and makes a baricentric projection on the triangles to calculate the grid values.

After the FFT transform in the frequency domain we can be see the frequency components; the artifical objects show higher frequencies because the sharp edges, the vegetation shows consistent spectrum. Figure 3. shows a cluster that contains a part of a forest. The upper graph represents the points, the middle one shows the interpolated grid, and the lower graph is the FFT image. The next sample on the

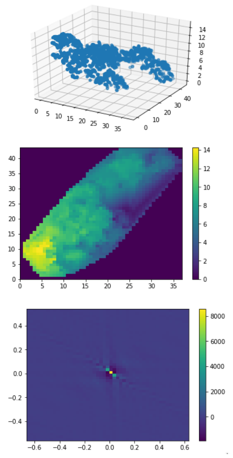
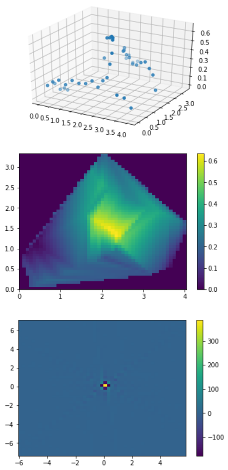


Figure 3: Trees: Points, linear interpolation and FFT

right shows a building; we can see that the clustering algorithm made a fault and joined the tree to the building, but the FFT frequencies have a wider domain than the forest. On the left there is a small cluster with few points; the interpolation made a poorly evaluable result.


(a) Pitched roof

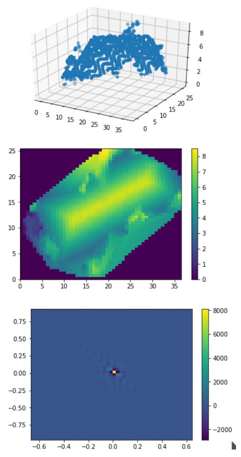

(b) Too small cluster

Figure 4: Computed FFT images

1.2.6. Classifying by the FFT values

During developing the earlier version of this algorithm I tried to compare the computed FFT image with sample images (forest, flat roof, pitched roof). The standard deviation of the two FFT that showed me the class of the cluster; but it has made wrong results, because the scale and rotation of the sample and the examined clusters had differences.

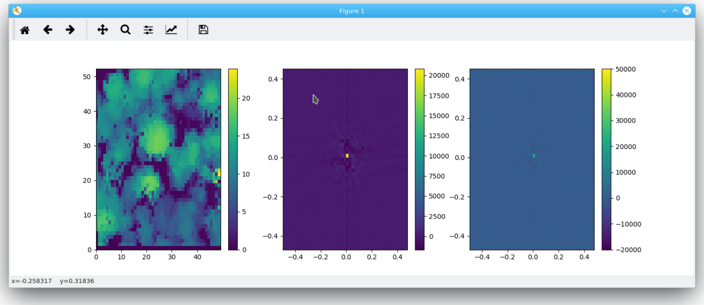


Figure 5: Sample forest FFT image

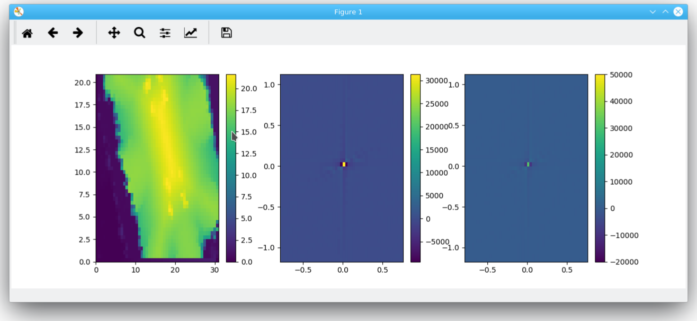


Figure 6: Sample pitched roof FFT image

I found a working method: examining the width of the frequency domain. But it has a problem, because if there is no enough points in the cluster, the FFT generates lower frequency range. Because this I need to divide the range with the number of the points, so I use five indicators: FFT variance/number of points; FFT spectrum maximum and minimum; FFT standard deviation / number of points; average of the FFT values; average elevation of the cluster's points. On the non classified cluster I calculate an elevation, and make it "ground" if this are bellow a threshold value. Some clusters leave non classified (for example, the walls).

2. Results and evaluating

I ran the algorithm on two sample point clouds. These clouds have low resolution, and contain metropolitain and town areas. There are trees mixed with family houses, it is hard to recognize and separate, even for the human evaluator. The figure on the left shows a smaller area, the right one represents a bigger one. The green color is the vegetation, the blue color denotes the ground, and the red patches represent buildings; the luminosity indicates the confidence of the classifying: dark red is the "I'm absolutely sure", the light red is the "hmm, maybe it's a building". I used the same parameters on both samples. On the small sample, the classifier

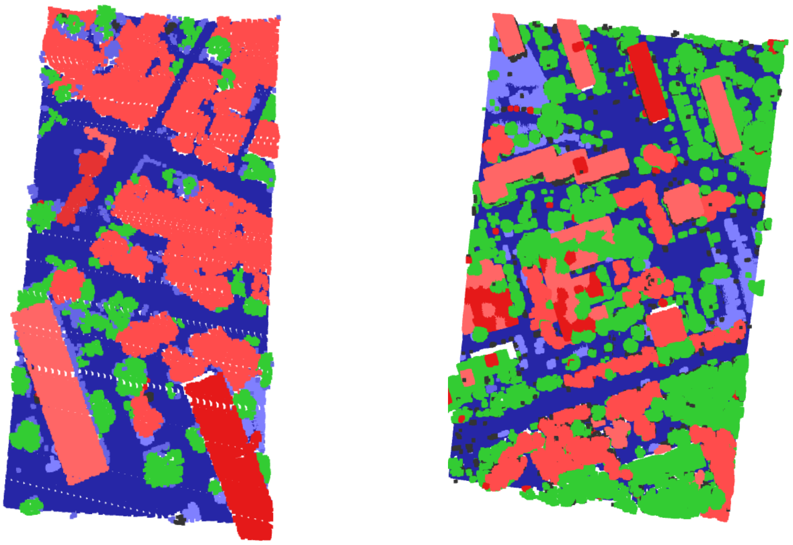

(a) Classified small area (b) Classified large area

Figure 7: Computed FFT images

made building from the trees, but on the bigger area the family houses changed to forest. But it has good result in the recognizing of the flat roofed buildings, and recognized the ground well.

3. Conclusion

The popular clustering methods based on calculate the distances between the points and searching the nearest neighbour. All of them use spatial indexing to accelerate this searching, still it is a compute-intensive operation, and the execution time is increasing logarithmically by the number of points. So, if we have large ares we need to segment it, and run the clustering on the tiles. The DBSCAN clustering is fast and simple to use, easy to set the parameters, but it has limited application.

The Birch clustering makes mistakes and is slow, and has many parameters. It is not a feasible algorithm for the point cloud operations.

Finally I think these clustering methods -based only on the point distances- are not too applicable for working with ALS point clouds. An effective clustering method should consider the elevation of the point to make a cluster. Basically, there is no really useful clustering method in this popular environment for the ALS LIDAR files. Examining the FFT image can give satisfying result but based on a suitable clustering method.

References

[1] Marc Bartels, Hong Wei (2009) Point-based classification http://www.cvg.reading.ac.uk/projects/LIDAR/publications/phd_reading/bartels_wei_LIDAR_ML_PRRS_06.pdf

[2] BlueMarble Geo Global mapper nonground classification https://www.bluemarblegeo.com/knowledgebase/global-mapper-19/Lidar_Module/nonground_classification.htm

[3] C. Hug, P. Krzystek, W. Fuchs (2004) Advanced lidar data processing with LasTools https://www.researchgate.net/publication/228347125_Advanced_lidar_data_processing_with_LasTools

[4] Nicolas Brodu (2019) CANUPO classifier http://nicolas.brodu.net/common/recherche/publications/canupo.pdf

[5] Open3D documentation www.open3d.org

[6] Scipy clustering https://scikit-learn.org/stable/modules/clustering.html#dbscan

[7] Martin Ester, Hans-Peter Kriegel, Jiirg Sander, Xiaowei Xu (1996) A Density-Based Algorithm for Discovering Clusters https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf

[8] Ricardo J. G. B. CampelloDavoud MoulaviJoerg Sander (2013) Density-Based Clustering Based on Hierarchical Density Estimates https://link.springer.com/chapter/10.1007/978-3-642-37456-2_14

[9] Sklearn Clustering Brich documentation https://scikit-learn.org/stable/modules/generated/sklearn.cluster.Birch.html#sklearn.cluster.Birch