LiDAR: Analysis and Classification

Steven Spiegel

Saturday 20th April, 2019

Abstract

LiDAR (Light Detection and Ranging) is ubiquitous in its use and applications. Some of its applications is in computer vision and robotics, surveying, emergency management, and object classification. this project aims to introduce the reader into the science of LiDAR, processing tools, and applications. In particular, this paper will look at PDAL (Point Data Abstraction Library) and its use in point cloud processing and application. The point cloud in question is a segment of a North Dakota collect. In conjunction with Python and Jupyter Notebook, this paper seeks to efficiently remove outliers, segment the ground, and classify the remaining points of an airborne LiDAR collection.

1 Introduction

LiDAR (Light Detection and Ranging) is used in numerous scientific contexts, particularly in the realm of computer vision and surveying. At it's heart, LiDAR measures the distances between transmitted and received signals. There are several main types of LiDAR:

1. Pulse: Emits a pulse and measures the time between hitting a target and the return time.
2. Phase: Measures distances using interferometry.
3. Geiger: Use sensors that are triggered by photons.

The mode of LiDAR used in my project is pulse based [1].

In addition to the different types of LiDAR, there are a number of different collection techniques. The four primary types are terrestrial, airborne, mobile, and Unmanned Aerial Systems (UAS). Terrestrial collects require the use of reflectors or planar regions to register scans together. The main struggle with this type of collect is the amount of work in moving the scanner from different scan positions. However, these scanners often have ranges that are on the kilometer scale, allowing for massive data collects. Mobile scans use the power of Robotic Operating Systems and feature matching to create a point cloud. As the price of small LiDAR scanners and Unmanned Aerial Systems have become increasingly affordable, UAS LiDAR collection is becoming more common. What would usually take days for surveyors to collect with a terrestrial scanner, UAS can collect in hours. A plethora of software suites exist to process the LiDAR data for relative accuracy as well as absolute accuracy.

In this context, we are focused on classifying point clouds once they have been collected and processed. There are a number of techniques to accomplish this. In general, the strategy is as follows: Classify the ground, segment the point cloud, and classify the segments, as demonstrated in the master's thesis of Nina Varney [5]. Other strategies can be taken to classify point clouds, such as PointNet, which utilizes an Convolution Neural Network to segment and classify point clouds [3]. In our case, we directly classify the ground and then use the geometric properties of the remaining points to identify what class they should be in. This is a more simplistic method than shown in [3] and [5], however it will be sufficient for our purposes. We know that in our scene we are only interested in classifying the vegetation and buildings. Hence, we can mainly focus on looking at the geometric properties of the points themselves and then deduce what class they should be in. The general strategy for processing the point cloud is as follows:

1. Remove outliers via a statistical or distance method.
2. Use morphological filtering and return numbers to classify ground returns.
3. Use point covariance to determine normal planes of remaining points.
4. Use Singular Value Decomposition of the Covariance Matrix to determine dimension of the points.

5. Classify based on the dimension of points.

In order to accomplish this, we will employ PDAL pipelines which are written as JSON files and used in a Jupyter Notebook [1]. PDAL has a python interface that allows one to create custom python code in addition to PDAL pipelines. These functions are written in C++, but PDAL allows python to be used in addition to the pipelines.

The next section will demonstrate these methods as well as give a brief discussion on the theory of each one. Then, a basic PDAL pipeline will be demonstrated For the purpose of this exercise, our goal is to classify outliers, ground, buildings, and trees. We will be classifying an airborne LiDAR collect called `CSite1_orig-utm` [1].

2 Methodology

The theoretical approach to point cloud classification is enumerated in section 1. We will use a PDAL pipeline in addition to python processing in Jupyter Notebook to remove outliers, classify ground returns, and classify vegetation and buildings. The next part of the methods section will give theoretical primers to each one of these tasks. Example PDAL pipelines are detailed in the appendix.

2.1 Outliers

The first thing we need to do is to classify the outliers. Outliers are noise points that aren't attributed to things we are interested in. In airborne LiDAR this could be due to dust, birds, etc. Hence, the first step is to eliminate outliers. There are 2 primary methods: Statistical and Radial.

2.1.1 Statstical

Statistical Outlier Removal requires 2 calculations across the point cloud. On the first pass, for point cloud P with n points, and for each point p_i for $i \in 1 \dots N$, the mean distance μ_i is computed for k nearest neighbors. Then, we calculate the global mean of means in the following:

$$\mu = \frac{\sum_{i=1}^{N} \mu_i}{N} \tag{1}$$

and the standard deviation (σ) is defined as

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N} (\mu - \mu_i)^2}{N - 1}} \tag{2}$$

Thus, the threshold (t) for outlier classification is as follows:

$$t = \mu + \sigma m \tag{3}$$

where m is a user defined input. Thus, we have the following criteria for a statistical outlier:

$$outlier_i = \begin{cases} true & \text{if } \mu_i > t \\ false & \text{otherwise} \end{cases} \tag{4}$$

. Hence, any point $p_i > t$ is considered an outlier and is given the classification of 7 (standard noise class for LiDAR).

2.1.2 Radial

The other outlier method is a radial outlier filter. The first calculation involves calculating k_i nearest neighbors for each p_i. Then, we select a minimum number of neighboring points k_{min} and create the following rule.

$$outlier_i = \begin{cases} true & \text{if } k_i < k_{min} \\ false & \text{otherwise} \end{cases} \tag{5}$$

An example pipeline can be found in the appendix ().

The next step is to classify ground returns (classification value of 2). In order to do this, We will employ mathematical morphology with a progressively growing window to find ground returns.

2.2 Ground Classification

Morphological filtering dates back to mathematical set theory and digital image processing [6]. Instead of using dialation and erosion of pixels and digital numbers, Progressive Morphological Filtering (PMF) dialates and erodes elevation values within a given window. Consider a point, p_i with corresponding coordinates (x_i, y_i, z_i). The dialation d_p of point p is defined as

$$d_p = \max_{(x_p, y_p) \in w} (z_p) \tag{6}$$

where points (x_p, y_p, z_p) are the set of neighbor point within a window, w. The erosion e_p of point p is defined as

$$e_p = \min_{(x_p, y_p) \in w} (z_p) \tag{7}$$

The erosion operatoin finds the minimum in a window w. Hence, we have the following equations that describe opening and closing operations.

$$Opening = erosion\ then\ dialation \tag{8}$$

$$Closing = dialation\ then\ erosion \tag{9}$$

The primary aspect that is different about progressive morphological filtering is the window size w does not stay at a constant size. An opening operation is done to remove smaller object like trees from the ground returns. Often buildings will be classified as ground returns since rooftops are often larger than the window size. However as window w gets progressiviely larger, buildings

are removed since the new lower elevations are ground returns within the window. In order to preserve terrain that might be sloping, a differential height threshold dh_T is introduced. The idea behind this is that buildings will have a larger height differential than terrain. These parameters are introduced so that ground returns won't be misclassified as non-ground returns [6]. An example of a progressive morphological filter pipeline can be found below. It filters ground points and classifies the ground points as 2 in the output las file.

Ground filtering pipeline

```
{
"pipeline":[
                    {
                            "type":"readers.las",
                            "filename":"test.las"
                    },
                    {
                            "type":"filters.pmf",
                            "cell_size":1.5,
                            "exponential":false,
                            "max_window_size":50,
                            "slope":0.4

                    },
                    {
                            "type":"writers.las",
                            "filename":"test_ground_classified.las"
                    }

          ]
}
```

2.3 Covariance of Points and Rank Estimation

2.3.1 Covariance and Curvature

Shape and plane estimation is extremely vital in determining structure within a point cloud. First and foremost is determining the normal vector, $\vec{n}$ at point p with k nearest neighbors. Hence, we are interested in estimating a normal plane tangent to the surface at point p, which becomes a least squares problem [4]. Let x and $\vec{n}$ be a point and a normal vector describing the plane, respectively. Then the distance d_i of a point p_i to the plane is as follows:

$$d_i = (p_i - x) \cdot \vec{n} \tag{10}$$

with the desire to minimize d_i. If we let x be the centroid of p_i (which we will call $\overline{p}$) we have

$$x = \overline{p} = \frac{1}{k} \sum_{i=1}^{k} p_i \tag{11}$$

We solve for $\vec{n}$ by looking at the eigenvalues and eigenvectors of the covariance matrix, C such that

$$C = \frac{1}{k}\sum_{i=1}^{k}(p_i - \overline{p}) \cdot (p_i - \overline{p})^T \tag{12}$$

Computing the eigenvalues and eigenvectors of this $3x3$ Covariance matrix we get

$$C\vec{v}_i = \lambda_i \vec{v}_i \tag{13}$$

where $i \in 0, 1, 2$. If we have $\lambda_0 < \lambda_1 < \lambda_2$, then normal vector $\vec{n}$ would be $\vec{v}_0$ [4].

One final note about eigenvalues and eigenvectors. We can compute the curvature of a point p by taking the smallest eigenvalue λ_0 and dividing it by the sum of all the eigenvalues. Hence, curvature at a point p labeled σ_p is as follows:

$$\sigma_p = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{14}$$

This estimates the residuals of a point p. The higher the residuals, the greater the eigenvalues and thus the greater the curvature. Lower values of σ_p indicate the point fits well on a plane [4].

2.3.2 Rank Estimation

In addition to estimating planes and computing point curvature, we can use singular value decomposition to determine the rank (dimension) of a point with k nearest neighbors [1]. The idea is to use the covariance matrix as illustrated in equation 12. Note that a covariance matrix is square and symmetric ($C = C^T$) so we can look at the spectral decomposition of the matrix, C [2]. Hence, we can write C as follows:

$$C = USU^T \tag{15}$$

where the columns of U are the eigenvectors of C and the diagonals of S are the eigenvalues of C. Hence we count the number of eigenvalues that are above a certain threshold, t (usually close to 0). Hence the dimension or rank of the point is equal to the number of eigenvalues above t.

A PDAL pipeline will be used to conduct the filtering. Then using Jupyter Notebook we will bring the points into a dataframe and view the points in a ipyvolume viewer. Then through a simple pandas query we can then distribute the rank 2 points that are not classified as ground to be buildings, since 2 dimensional objects in this scene are rooftops. Then, we can set points with rank 3 to be vegetation. This will give us a reasonable classification of the scene. Below is a full pipeline in order to classify the scene.

3 Results and Discussion

Figure 1 shows the results of the pipeline. As we can see, the buildings are yellow and the vegetation is green. There seems to be misclassification between some ground returns and buildings.

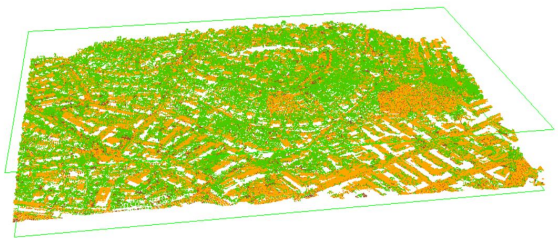

Figure 1: Output via PDAL and Jupyter Notebook

This can be avoided by changing some of the parameters with the filter. Note that instead of using a progressive morphological filter I used a simple morphological filter, which is based on the pmf. Also note that cars would not be correctly classified. There are better methods of classifying point clouds. There are different ways to classify point clouds as noted in section 1. Better methods include segmenting point clouds into different objects and using machine learning techniques. Some of these methods include support vector machine and deep neural networks.

Even though

References

[1] Andrew Bell, Brad Chambers, Howard Butler, and Michael Gerlek. Introduction to lidar - pdal.io. `https://pdal.io/`, 2019.

[2] Rasmus Elsborg Madsen, Lars Kai Hansen, and Ole Winther. Singular value decomposition and principal component analysis. *Neural Networks*, 1:1–5, 2004.

[3] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[4] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Computer Science department, Technische Universitaet Muenchen, Germany, October 2009.

[5] Nina Varney. Lidar analysis for automatic region segmentation and object classification. Master's thesis, University of Dayton, Dayton, Ohio, 12 2015.

[6] Keqi Zhang, Shu-Ching Chen, Dean Whitman, Mei-Ling Shyu, Jianhua Yan, and Chengcui Zhang. A progressive morphological filter for removing nonground measurements from airborne lidar data. *IEEE transactions on geoscience and remote sensing*, 41(4):872–882, 2003.