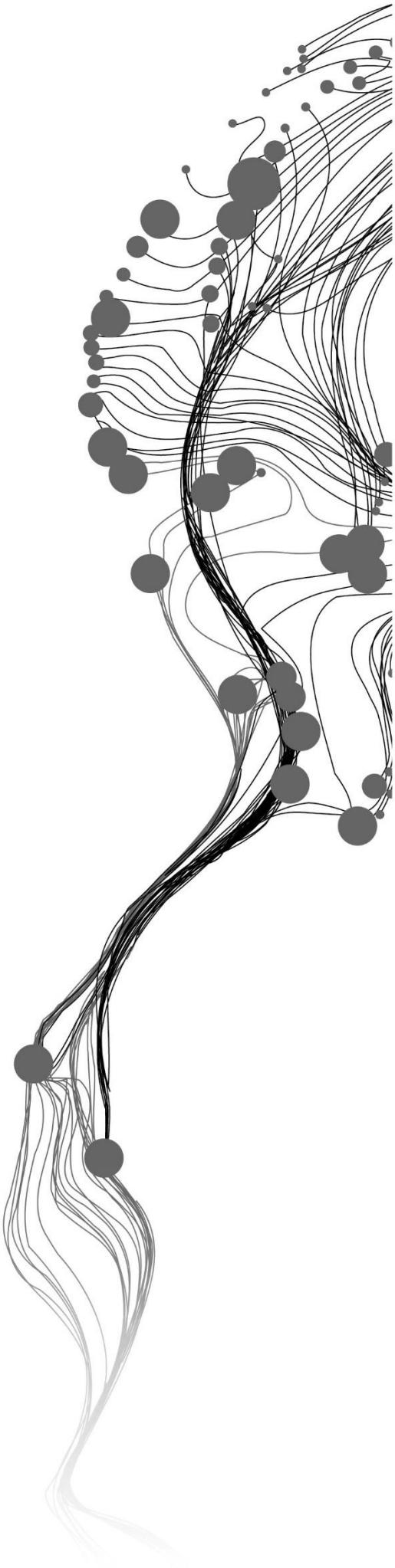

City of Amsterdam – ITC UTwente

Individual tree delineation

Internship project

Jorges Nofulla

December 2022 – March 2023

Table of Contents

Introduction 3
Methods and Results 4
Initial delineation 4
Method 1 5
Cluster cleaning and local maxima calculation 6
Individual tree trunks delineation 7
Method 2 9
Extracting tree parameters 10
Final delineation 11
Balance the newly delineated trees. 11
Load and compare 12
Discussions and Conclusions 13
References 15

List of figures and tables

Figure 1 - KD-tree Algorithms (2D, 3D) 4
Figure 2 - The workflow of the first method 5
Figure 3 - Image of a group of trees taken in Amsterdam 6
Figure 4 - Point cloud representation of a group of trees, using point density instead of height value 7
Figure 5 - Output of the first method where individual tree trunks are highlighted 8
Figure 6 - Flowchart of the second method 9
Figure 7 - BoxPlot of the number of points per cluster 10
Figure 8 - Output of the second method. 2D polygons of delineated trees. 12
Figure 9 - An image of a cluster of trees taken from a distance in the city of Amsterdam 13
Figure 11 - A closer perspective into another cluster of trees in the city of Amsterdam 14
Figure 10 - A closer look into a tree cluster of the city of Amsterdam 14

Table 1 - Average parameters for each tree type 10

Individual tree delineation

Introduction

Vegetations are a vital part of our daily lives and are crucial for urban planning and the continuity of ecosystems. Accurate identification and delineation of individual trees in point cloud data is an important task in a variety of fields, including forestry, urban planning, and environmental management. The information we get from individual trees is a critical aspect of forest stand knowledge, which constitutes a fundamental component of forest management (Wang et al., 2004). Individual tree delineation is a very popular and challenging topic in the field of point cloud analysis. Their complex and diverse characteristics make these objects very hard to delineate.

There are a number of methods and algorithms that have been developed for identifying and delineating individual trees, including the use of morphological image analysis (MIA) (Sarabia et al., 2020), mean shift segmentation (Xiao et al., 2019), and KD-trees. Several related works have explored the use of more complex models such as deep learning for aerial image analysis tasks, and one notable contribution in this area is the "Transformer for Tree Counting in Aerial Images" paper (Chen & Shang, 2022), which presents a novel approach for accurately counting trees using transformer-based models.

Previous research has also investigated neural network-based approaches for tree counting in UAV remote sensed images, such as the work by (Kestur et al., 2018) which proposed a spectral-spatial method for tree crown detection, delineation, and counting. Additionally, (Marques et al., 2019) proposed an automatic detection and monitoring system for chestnut trees using UAV-based remote sensing. These works demonstrate the potential of machine learning and remote sensing technologies for accurate tree counting and monitoring.

Recently, high-density point clouds have become more available, enabling more detailed studies to be conducted. An important characteristic of point clouds is their ability to penetrate through trees and capture their structure. Point cloud data can be collected using methods such as airborne laser scanning (ALS), terrestrial laser scanning (TLS), and structure from motion (SFM), which provide a dense, high-resolution representation of the surface of an object.

In addition to the works on tree counting from aerial images, there are also studies that have explored individual tree extraction and segmentation from LiDAR point clouds. For example, (Zhang et al., 2015) developed a method for individual tree segmentation from LiDAR point clouds in urban areas, aimed at facilitating urban forest inventory. Another study by (Hui et al., 2021) proposed a self-adaptive mean shift segmentation method for individual tree extraction from UAV LiDAR point clouds. These studies demonstrate the potential of LiDAR point clouds for accurate and efficient individual tree extraction and segmentation, which is crucial for forest management and conservation.

In this study, we have access to the Actueel Hoogtebestand Nederland (AHN) dataset, a publicly available high-resolution point cloud dataset generated by aerial laser scanning (ALS) throughout the Netherlands. This dataset also provides semantic labeling at the point level for four classes: "ground, water, buildings, and others." The dataset that we will be using

as the starting point for this study was previously labeled by the Amsterdam Intelligence team, with all tree points classified.

In this report, we present 2 different approaches for identifying individual trees. Both of the methods initial delineation is based on utilizing a KD-tree algorithm (Bentley, 1975) and vectors to make tree clusters. After we make these tree clusters :

The first method identifies local maxima within a designated region through analysis of point density in that area. The concept of using point density was inspired by Professor Sander Oude Elberink. He utilized it to represent high-density parts of the trees, which are often tree trunks since they typically have the highest point densities around them in the 2D plane. The process includes calculating each point's value based on how many other points are present in a 2D buffer around it.

The second method uses some parameters to further split or merge the clusters. These parameters are based on the average tree characteristics such as tree height, diameter and point cloud density. To get parameters as close to reality as possible, the data is first cleaned from all the outliers and categorized into small, medium and tall trees. Then in a decision tree type of algorithm, these parameters were used to delineate individual trees.

Methods and Results

Two different approaches were tested to delineate individual trees. The initial delineation is a common step for both of the methods. These methods are described below.

Initial delineation

To reduce the density of the raw point cloud data, the x, y, and z values of the LAS file were concatenated and converted to full integers using the NumPy library. This resulted in a flat voxel index, from which a subset of coordinates was selected by identifying a unique set of indices using the np. unique() function. The selected coordinates were sorted by their z value, and a list of "point" class objects was created for each set of coordinates. An ordered array was then generated with the size of the remaining coordinates, effectively reducing the number of points fed into the algorithm. This step was necessary to decrease the processing time and improve the efficiency of the algorithm.

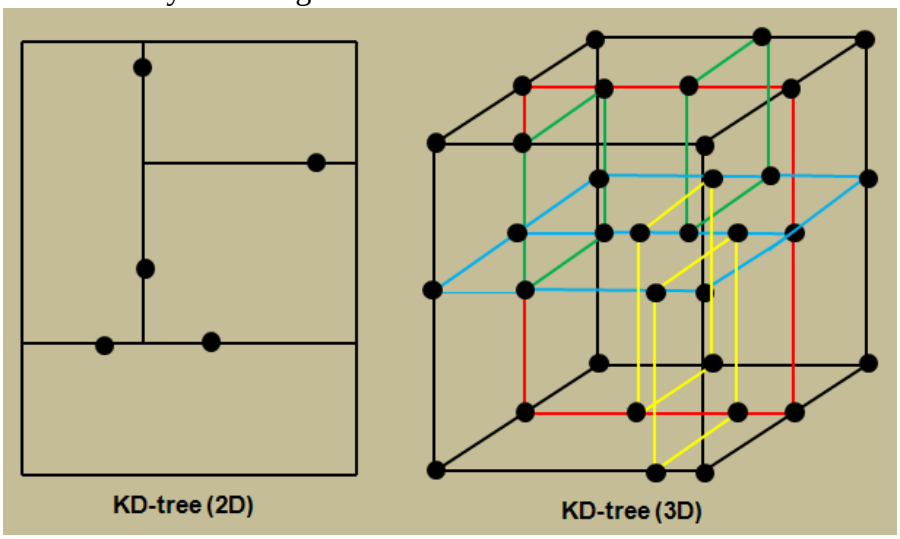

Figure 1 - KD-tree Algorithms (2D, 3D)
Available from: https://www.researchgate.net/figure/The-example-of-the-KD-tree-Algorithms-2D-3D_fig3_263937521 [accessed 9 Mar, 2023] (Park et al., 2010)

The radius used for clustering was manually selected based on the density and characteristics of the point cloud data. The KD-tree algorithm with closest neighbors was then utilized to delineate trees, with the core functionality remaining the same as in the paper by (Hess et al., 2021). Our version of the code has been streamlined to enhance its efficiency and to better align with the specific requirements of our study, while still maintaining the core functionality.

KD-trees are effective in identifying groups of points belonging to the same tree based on proximity to one another. The algorithm identified points within a certain distance from a given point in 3D space and calculated the centroid of the neighbors with higher elevation to determine the top of the tree. A link was then established to the neighbor closest to the centroid, and the resulting clusters were used to create geometric networks representing individual trees. Each point was assigned a label based on its network, which was later used to segment the trees. The approach demonstrated promising results in accurately segmenting deciduous trees in our study area, highlighting the potential of KD-trees and geometric networks in processing airborne lidar point clouds for tree segmentation. Finally, all points within a specified radius were clustered with their neighbors to create a group of points. The centroid of this group of points was calculated, and the neighbor closest to the centroid was selected as the parent point. This process was repeated until all points in the dataset were assigned a parent point. The output of the initial delineation step was saved as a NumPy array and then fed to both of our methods below.

Method 1

The workflow of the first method is shown below (Figure 2):

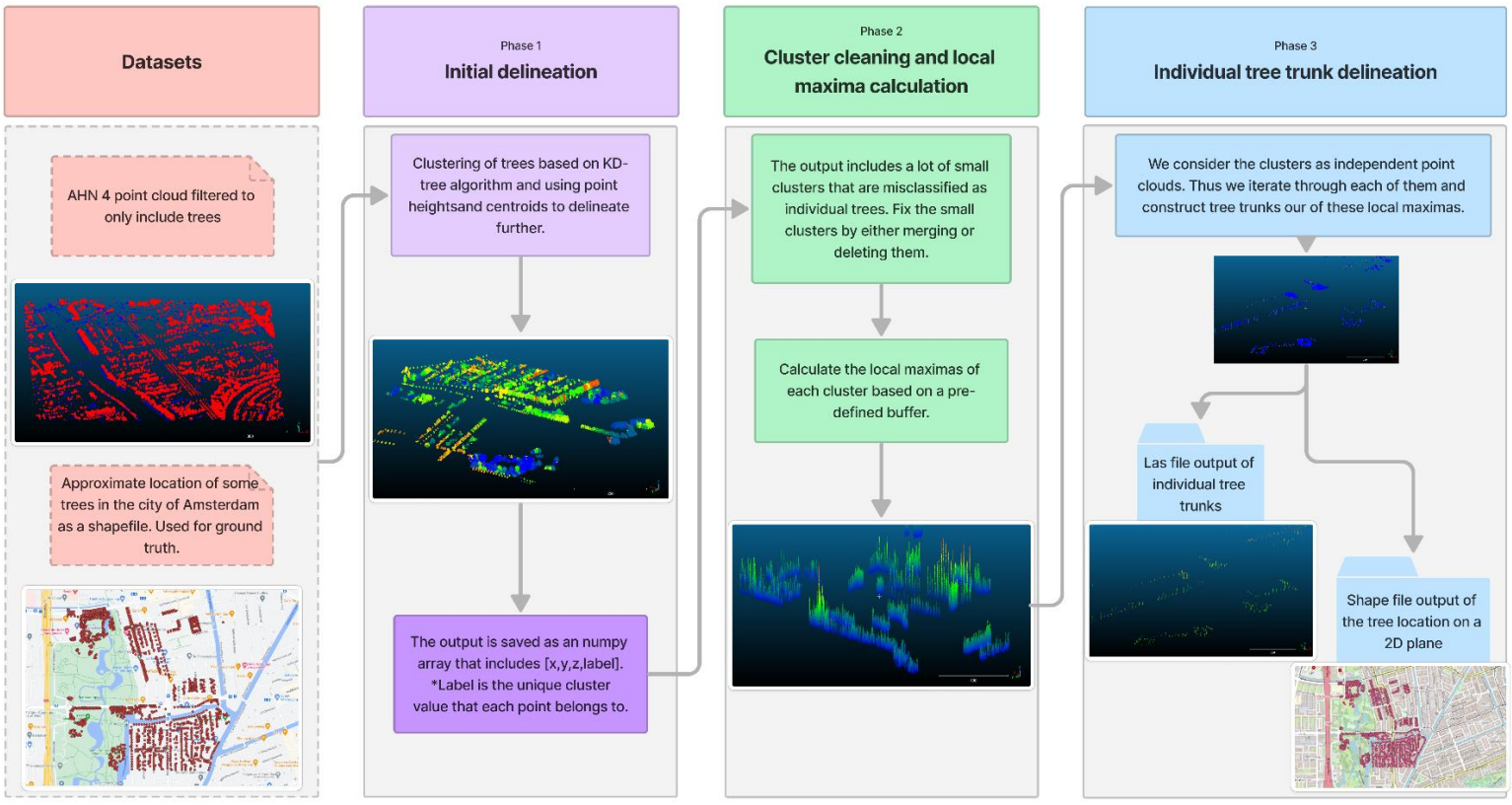

Figure 2 - The workflow of the first method

Cluster cleaning and local maxima calculation

Once the initial delineation process is complete, the resulting data may contain numerous small clusters erroneously identified as individual trees due to over-segmentation or incorrect labelling of small objects. To mitigate such errors, a threshold was established to filter out these clusters by either merging them with the nearest cluster, assuming they are branches of the same tree that were mistakenly classified separately, or deleting them if their size and location do not align with that of a tree.

After the data is cleaned from these clusters, a buffer was constructed for every single point cloud and created a new feature. This new feature is called point density and it counts how many other points are inside this buffer in the 2D plane. The algorithm iterates through every point cloud in the file and saves the number of points in the buffer. The output array will look like this → [x, y, z, label, point_density].

Where:

- x, y and z are the coordinates of the point.
- label is the unique cluster number that this point belongs to (from the initial delineation).
- point_density is the number of points that are within the 2D buffer of this point.

The idea behind this point density is to use it when searching for trees centroids, instead of using the height value. Depending on their branch distribution, a tree can have multiple local maxima in height as seen in Figure 3.

Figure 3 - Image of a group of trees taken in Amsterdam

This occurrence is less common when using point density as the local maxima. Usually, the local maxima of point density will be on the tree trunks, as the location of the trunk has extra point clouds compared to other tree parts. Thus when searching in a 2D plane, it's almost certain that the peak will be on the tree trunk area.

Below is a visualization of trees using the point density value instead of height value (Figure 4). It's clear that the peaks are much more visible compared to a normal point cloud that uses Z value to represent trees height.

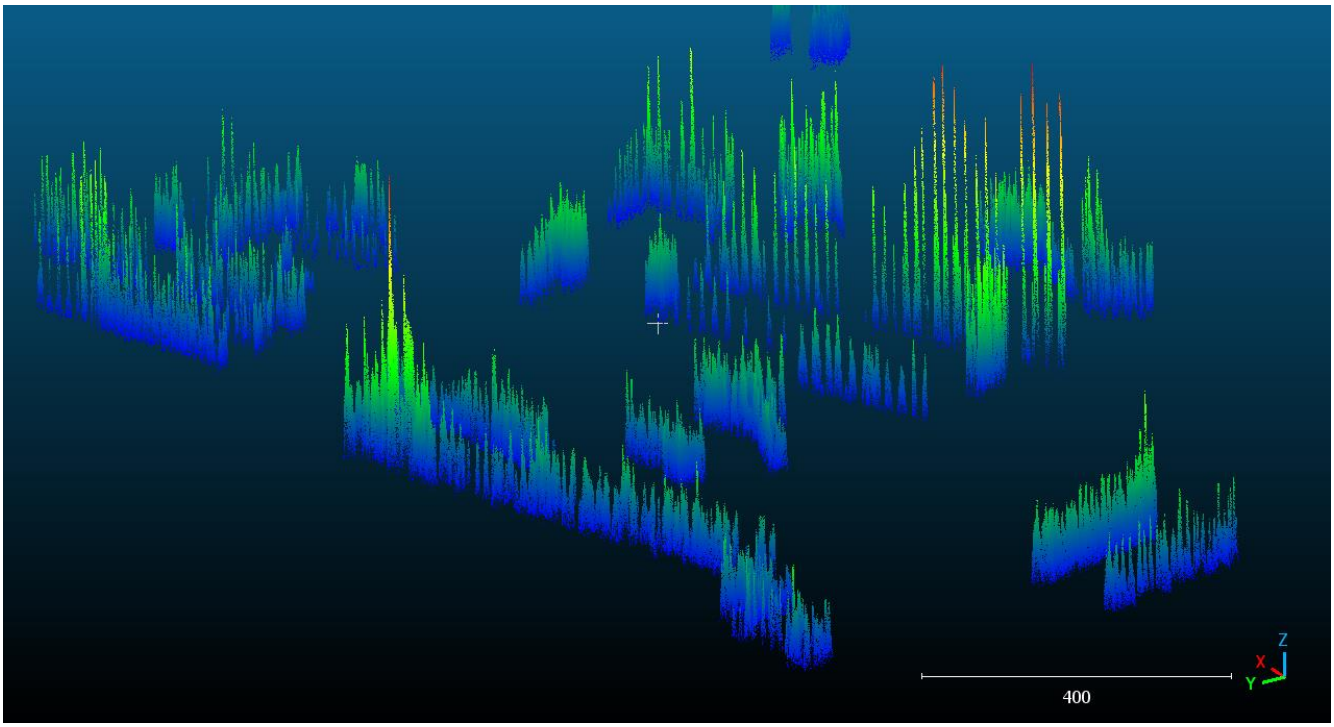

Figure 4 - Point cloud representation of a group of trees, using point density instead of height value

Individual tree trunks delineation

Point density calculation for the purpose of finding the local maxima can be also calculated in the entire dataset without pre-clustering but there are many problem and errors that come with it such as:

- Ignoring small trees that are next to tall trees, as the window size is static. The local maxima will be set on the tall trees, leaving small trees undetected.
- If we iterate through all the points instead of through the clusters, the time complexity is exponentially higher. It will be impossible to run it on a huge dataset, especially AHN4 which has a really high point density.

These are the reasons why we pre-cluster the data and then we iterate through the points of each cluster individually instead of the dataset as a whole. The concept behind this comes from "divide and conquer", which involve breaking down a larger problem into smaller, more manageable subproblems (Kleinberg & Tardos, 2006) (Béjar Alonso, 2013). This way we get a much faster algorithm and the boundaries of the cluster won't allow for a lot of false positive trunks or ignore small trees.

The proposed method is done via Python code, and the way it works is that it makes an array of all the local maximas that it finds in each tree cluster. To further increase the accuracy, the code checks for minimum distance between 2 local maximas of the same cluster, as we can't have 2 trunks next to each other.

The code assigns an unique label to the peaks and then labels every points in a small 2D radius with the same label in order to visualize the whole trunk as show in the Figure 5. This unique label will represent them as individual tree trunks.

These labeled clusters can help reveal patterns or trends in the data, and can be further analyzed to extract useful information. The algorithm is flexible and can be adapted to fit the needs of a particular analysis, making it a valuable tool for data exploration and analysis.

The parameters are made easily to change in order for the user to have a more personalized code for specific uses.

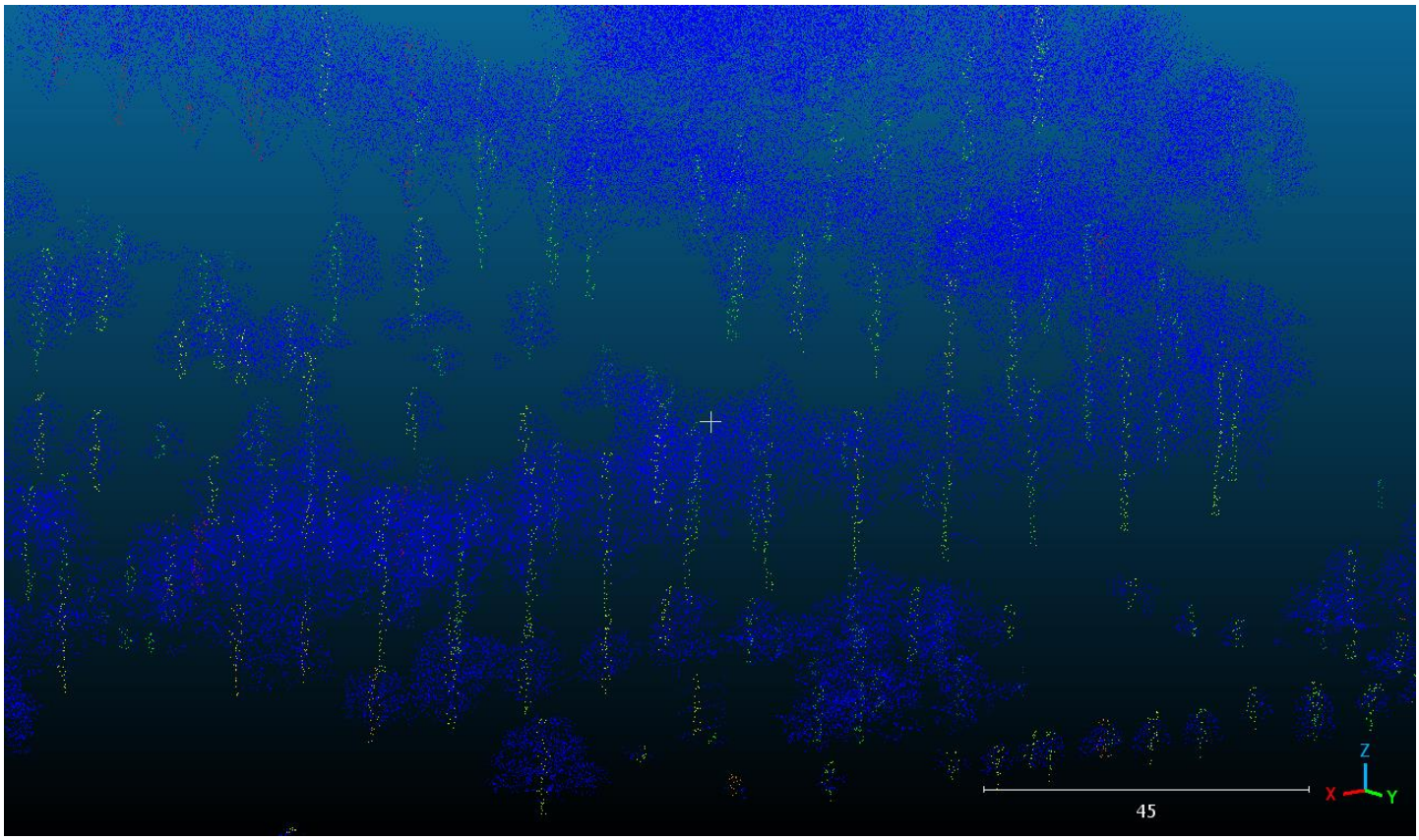

Figure 5 - Output of the first method where individual tree trunks are highlighted

Method 2

The flowchart of the second method is shown below in Figure 6:

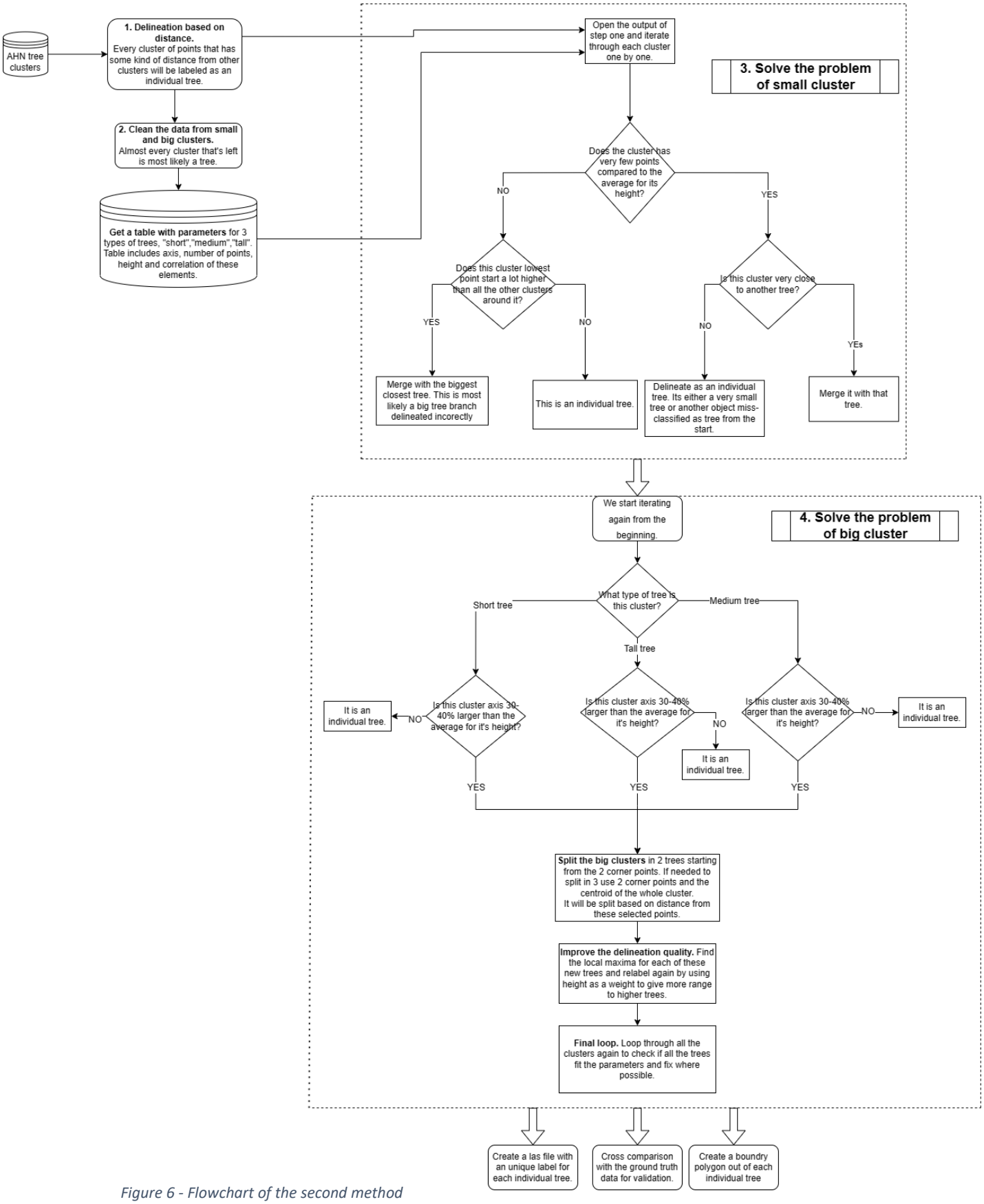

Figure 6 - Flowchart of the second method

Extracting tree parameters

Using the primary delineation obtained in the first step, small and large clusters are removed upon a threshold. The threshold of removing these clusters is decided by data distribution (Figure 7).

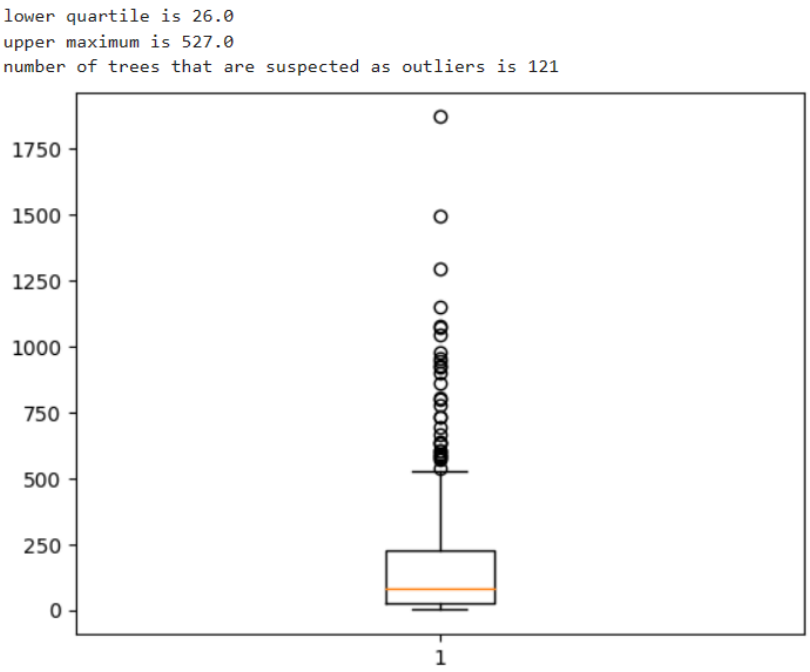

Figure 7 - BoxPlot of the number of points per cluster

This step is necessary to eliminate noise and improve the accuracy of the final delineation. As we can see from Figure 7, most of the correct trees should be between 26-527 points and there are 121 trees that are suspected as outliers (In this specific case, in another dataset these numbers will change). We remove the all the "upper quartile +20%" clusters and the remaining clusters, which are mostly correctly delineated trees, are then split into three groups: small, medium, and tall.

From the 3 groups of trees, height, axis, the number of points, and correlations are obtained . This set of parameters was extracted from clusters of points in the data, and later (on the next step) a classification algorithm was used to classify the clusters as either trees or non-trees based(Maas et al., 2008). All of these parameters are saved in a txt file for later use (Table1).

Tree type	Height	Major axis	Number of points	Height/Major axis
Short	3.0	5.3	49.74666667	0.63
Medium	6.4	7.7	134.8657718	0.98
Tall	11.1	11.2	300.3066667	1.09

Table 1 - Average parameters for each tree type

Final delineation

The Las file saved in the first step is loaded, and the first problem to solve is small clusters. Small clusters can be difficult to accurately identify as individual trees, as they may not contain enough points or have a clear shape. To address this problem, we checked these characteristics:

1. The cluster is small. This can be determined based on the number of points in the cluster or the size of the bounding box surrounding the cluster.
2. The cluster is very close to another big tree. This can be determined based on the distance between the centers of the clusters.
3. The cluster's lowest point starts from a much lower elevation than other clusters around it. This can be determined based on the height of the lowest point in the cluster relative to the surrounding clusters.

If these conditions are met, the cluster is merged with the nearest big tree . Otherwise, it is considered a small tree. More detailed on the flowchart of the workflow (Figure 6).

The second problem is trees that are merged together into a single large cluster. This can occur when multiple trees are very close together and their crowns overlap. To solve this problem, a major axis check can be done, in order to find the trees that have unusually large axis(Sarabia et al., 2020). The following steps are taken in the code:

1. Iterate through each cluster identified as an individual tree.
2. Determine if the cluster is a small, medium, or tall tree based on the criteria established in Table 1.
3. Apply the following rules to each cluster, according to its category: If the axis is more than the average axis for the tree category(small,medium,tall), or the point cloud number is more than the average for the tree category, it means that this cluster is not a single tree, the cluster is split in multiple trees, depending on how much larger than the average are these values. More detailed on the flowchart of the workflow (Figure 6).

Balance the newly delineated trees.

After the "Final delineation" step, every cluster that has doesn't fit the Table 1 parameters is either split or merged. But the split is always equally done for each cluster. However, this should not be the case when a tall and small tree are merged, as tall trees should have more points and are bigger than small trees. In this case, to solve this problems we first find the centroids and local maxima of each newly split tree. The different weights are applied to different heights. For example, the highest tree will get a larger weight than the lowest tree.

In the end we loop through all the clusters again and check if the parameters fit each new tree cluster. If not, we apply small adjustments.

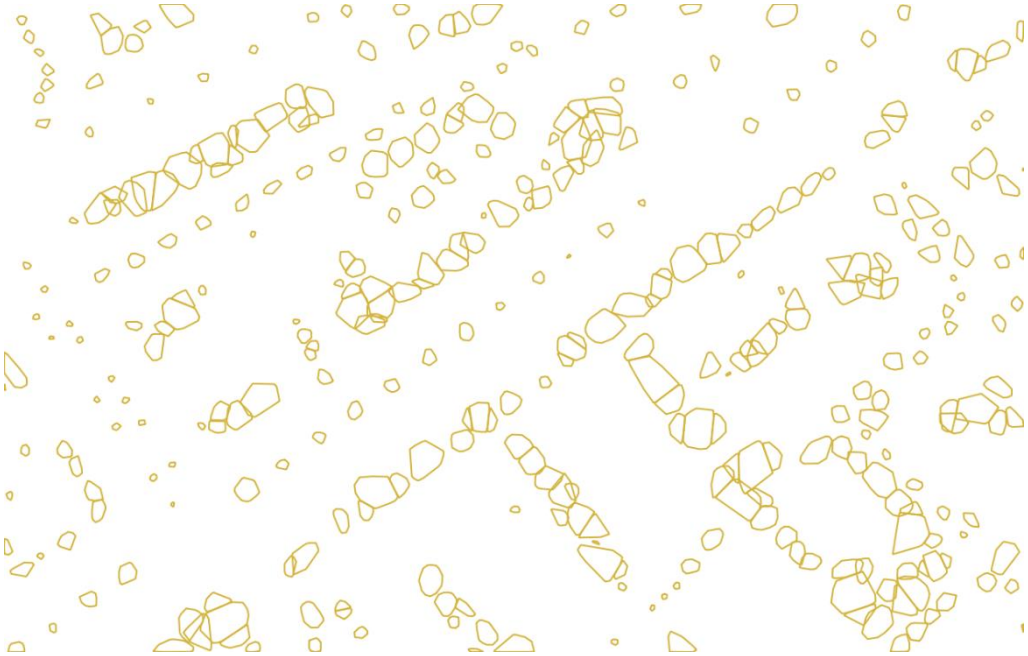

Figure 8 - Output of the second method. 2D polygons of delineated trees.

Load and compare

The initial and modified delineations are loaded and compared to ensure the code is functioning as intended. Ground truth data is used to validate the results. For better visual inspection a 2D polygon of each individual tree border is created(Figure 8).

Discussions and Conclusions

Both of the proposed methods show promising result when explored in relatively small datasets and they appear to be effective when counting the total number of trees in clusters that contain just one or a few trees. The situation is different when giving the algorithm larger point cloud tiles and more complex tree clusters.

The first method is quicker than the second since fewer computations are required for each tree. There is no need to extract any pre-calculated parameters from the data; instead, the utilized feature is retrieved by iterating over the points, making the technique quicker and more versatile.

The second approach does delineate the entire tree, whereas the first method only finds the tree trunks. The primary issue with the second method was that tree parameters were not as easily generalized as first anticipated. Since trees differ so significantly, they cannot be generalized into a finite number of types. The algorithm attempts to delineate trees using the dataset parameters, however balancing this delineation proved to be an incredibly difficult process.

The deeper we go into the data and structure of tree clusters, the clearer it becomes that trees in cities are quite distinct and unique, since they do not follow any particular pattern, especially in various regions of the city. For example, trees that are located alongside streets have dissimilar traits when compared to the trees in parks, and there can be variations in characteristics even among trees within the same vicinity. Figures 10 and 11 demonstrate the heterogeneous and unpredictable nature of tree diversity.

Figure 9 - An image of a cluster of trees taken from a distance in the city of Amsterdam

While the trees in Figure 9 appear to have a pattern and are fairly divided, a closer inspection reveals that things are really different (Figure 10 and 11).

These are only a few of the instances that demonstrate why the second strategy is ineffective for identifying trees in cities.

Figure 11 - A closer look into a tree cluster of the city of Amsterdam

Figure 10 - A closer perspective into another cluster of trees in the city of Amsterdam

The first method works around this diversity because, despite their differences, trees all have one thing in common: they all have a trunk that has more points given to it than any other section of the tree. This approach also performs significantly better in densely packed clusters. Looking for trunks was more successful than relying on how a tree should look.

When parameters were applied to the second method, many large trees were split unnecessarily since they simply did not meet the "average" specifications for their height; consequently, this method is not the solution we want. The first technique is also superior since it is much easier for the user to grasp and is more adaptable to manual modifications because it does not rely on automatic parameters derived from the dataset.

The code may be improved further so that it detects not only the position, but also the entire tree and its type. Making the sliding window dynamic to fit better into different tree clusters might be one approach to improve accuracy.

References

Hess, M., Rheinwalt, A., Brell, M., & Bookhagen, B. (2021, March 3). *Deciduous tree segmentation using airborne lidar point clouds and geometric networks: Examples from the Park Sanssouci, Potsdam, Germany*. CO Meeting Organizer EGU21. Retrieved from https://doi.org/10.5194/egusphere-egu21-4155

Béjar Alonso, J. (2013). *Strategies and algorithms for clustering large datasets: a review*.

Bentley, J. (1975). Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM September, 1975. Vol. 18: Pp. 509-517 : Ill. Includes Bibliography., 18*.

Chen, G. & Shang, Y. (2022). Transformer for Tree Counting in Aerial Images. *Remote Sensing*, *14*, 476. https://doi.org/10.3390/rs14030476

Hui, Z., Li, N., Xia, Y., Cheng, P. & He, Y. (2021). INDIVIDUAL TREE EXTRACTION FROM UAV LIDAR POINT CLOUDS BASED ON SELF-ADAPTIVE MEAN SHIFT SEGMENTATION. In *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences: Vols. V-1–2021*. https://doi.org/10.5194/isprs-annals-V-1-2021-25-2021

Kestur, R., Angural, A., Bashir, B., S N, O., Anand, G. & Meenavathi, M. (2018). Tree Crown Detection, Delineation and Counting in UAV Remote Sensed Images: A Neural Network Based Spectral–Spatial Method. *Journal of the Indian Society of Remote Sensing*, *46*. https://doi.org/10.1007/s12524-018-0756-4

Kleinberg, J. & Tardos, E. (2006). *Algorithm design*. Pearson Education India.

Maas, H.-G., Bienert, A., Scheller, S. & Keane, E. (2008). Automatic forest inventory parameter determination from terrestrial laser scanner data. *International Journal of Remote Sensing*, *29*, 1579–1593. https://doi.org/10.1080/01431160701736406

Marques, P., Pádua, L., Adão, T., Hruška, J., Peres, E., Sousa, A. & Sousa, J. (2019). UAV-Based Automatic Detection and Monitoring of Chestnut Trees. *Remote Sensing*, *11*, 855. https://doi.org/10.3390/rs11070855

Park, H., Lim, S., Trinder, J. & Turner, R. (2010). *Voxel-based volume modelling of individual trees using terrestrial laser scanners*.

Sarabia, R., Aquino, A., Ponce, J. M., López, G. & Andújar, J. M. (2020). *Automated Identification of Crop Tree Crowns From UAV Multispectral Imagery by Means of Morphological Image Analysis*. 1–23. https://doi.org/10.3390/rs12050748

Wang, L., Gong, P. & Biging, G. (2004). Individual Tree-Crown Delineation and Treetop Detection in High-Spatial-Resolution Aerial Imagery. *Photogrammetric Engineering & Remote Sensing*, *70*, 351–357. https://doi.org/10.14358/PERS.70.3.351

Xiao, W., Zaforemska, A., Smigaj, M., Wang, Y. & Gaulton, R. (2019). Mean Shift Segmentation Assessment for Individual Forest Tree Delineation from Airborne Lidar Data. *Remote Sensing*, *11*. https://doi.org/10.3390/rs11111263

Zhang, C., Zhou, Y. & Qiu, F. (2015). Individual Tree Segmentation from LiDAR Point Clouds for Urban Forest Inventory. *Remote Sensing*, *7*, 7892–7913. https://doi.org/10.3390/rs70607892