A VOXEL-BASED METHODOLOGY TO DETECT (CLUSTERED) OUTLIERS IN AERIAL LIDAR POINT CLOUDS

BY
SIMON GRIFFIOEN
2018

TU Delft

MSc thesis in Geomatics for the built environment

A VOXEL-BASED METHODOLOGY TO DETECT (CLUSTERED) OUTLIERS IN AERIAL LIDAR POINT CLOUDS

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of

Master of Science in Geomatics for the Built Environment

by

Simon Griffioen

November 2018

The work in this thesis was made in the:

3D geoinformation group
Department of Urbanism
Faculty of Architecture & the Built Environment
Delft University of Technology

Deltares

Supervisors:	Dr. Hugo Ledoux
	Dr. Ravi Peters
	ir. Maarten Pronk (Deltares)
Co-reader:	Dr. Martijn Meijers

ABSTRACT

To obtain 3D information of the Earth's surface, airborne LiDAR technology is used to quickly capture high-precision measurements of the terrain. Unfortunately, laser scanning techniques are prone to producing outliers and noise (i.e. wrong measurements). Therefore, a pre-process of the point cloud is required to detect and remove spurious measurements. While outlier detection in datasets has been extensively researched, in 3D point cloud data it is still an ongoing problem. Especially, clustered outliers are hard to detect with previous local-neighborhood based algorithms.

This research explores the possibilities of using a voxel-based approach to automatically remove outliers from aerial point clouds. A workflow is designed in which a series of voxel-based operations are integrated, with the aim to detect all types of outliers and minimize false positives. Voxels can be processed more efficiently than 3D points for two reasons: (1) A voxel-grid can be analyzed using efficient image processing techniques; (2) Voxels group inner points before feature extraction using neighborhood operators. Outliers are detected in two steps. First, the source point cloud is voxelized. Secondly, outliers are detected by computing connected components and labeling voxels not connected to the largest region as outliers. Simultaneously, analysis of the point's local density, shape (planar) and intensity minimize classification of false positives.

The presented algorithm generally detects outliers with a higher accuracy than previous local neighborhood-based methods. A comparison with an existing approach shows that more outliers are detected. Above all, clustered outliers are removed. However, some issues can still be improved. First, more research is necessary to classify outliers based on non-arbitrary decisions. This could potentially be improved by introducing supervised learning algorithms. Secondly, more attention is required to process massive point clouds that do not fit in internal memory. This study proposes a possible streaming solution.

ACKNOWLEDGEMENTS

I would like to thank all the people who guided me during my graduation project. First of all, thanks to my supervisors Ravi Peters and Hugo Ledoux for their guidance, constructive meetings and feedback, throughout the entire project. I would like to thank Martijn Meijers for his feedback in the final period of my graduation.

I also would like to thank Deltares, and specifically Maarten Pronk, for giving me the opportunity to collaborate on my graduation project, and support my research interests. Moreover, Maarten has given me very detailed and in-depth feedback. We had many pleasant and constructive discussions, and his extensive knowledge of the topic helped me guiding this project.

Furthermore, I want to thank Jeroen Leusing (Het Waterschapshuis) for sharing his knowledge about point cloud processing and for bringing me in contact with Frank van den Heuvel (Aerodata International Surveys). I would like to thank Frank (Aerodata), Maarten (Deltares) and Willem van Hinsbergh (Kadaster) for handing me a various collection of datasets, which made this research possible.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACRONYMS

AHN Actueel Hoogtebestand Nederland 7
ALS Airborne Laser Scanning xi
CCL Connected Components Labeling xii
DEM digital elevation model 1
DIM Dense Image Matching 4
DSM Digital Surface Model 1
DT Delaunay triangulation 11
EMST Euclidean Minimum Spanning Tree
FN False Negatives 2
FP False Positives 2
FNR False Negative Rate 44
FPR False Positive Rate 44
GG Gabriel Graph
GIS geographical information system 11
kNN *k*-Nearest Neighbors 22
LiDAR Light Detection And Ranging ix
LoD level of detail 12
NN Neural Network 64
PCA Principal Component Analysis 7
SVM Support Vector Machine 64
TIN triangular irregular network 11
TLS Terrestrial Laser Scanning 5
TN True Negatives 45
TP True Positives 32

1 INTRODUCTION

Laser scanners help acquiring the 3D geometry of real-world objects in an efficient and simple way. To obtain 3D information of the Earth's surface, airborne LiDAR technology is used to quickly capture high-precision measurements of the terrain. The generated raw point cloud reflects the characteristics of a Digital Surface Model (DSM) of the measured region. Unfortunately, laser scanning techniques are prone to producing outliers and noise (i.e. wrong measurements) due to limitations of the sensor, illumination situations, and undesirable artifacts in the scene [Jenk et al., 2006; Han et al., 2017]. Usually, elevation levels of outliers are higher or lower than their neighboring points, i.e. they are mainly measurements that do not obey the local surface geometry [Matkan et al., 2014]. Fig. 1.1 gives examples of ALS point clouds with many outliers. Besides affecting the visual appearance of the dataset, these points cause geometric differences making it unable to achieve desired results from further processing steps, e.g. extracting a digital elevation model (DEM) or object recognition. Therefore, a pre-process of the point cloud is required to detect and remove spurious measurements.

While outlier detection in datasets has been extensively researched, in 3D point cloud data it is still an ongoing problem. Several methods for detecting outliers in LiDAR data are proposed and studied. Typical methods are designed and developed in commercial software, e.g. lastools[1]. Free alternatives exist as well in the form of open source projects like Point Cloud Library[2] (PCL). Often a density-based approach is offered, which depends on the local density of the neighborhood points. The neighborhood can be defined by various constructions, such as a volume (e.g. sphere, cube) or k-nearest neighbors. Usually this performs well for detecting isolated points (Fig. 1.1a), but are incapable of removing clustered outliers [Shen et al., 2011] (see Fig. 1.1b and Fig. 1.1c) or outliers spatially close to real objects (Fig. 1.1d). More examples and other approaches are described in § 2.2.

Removing clustered outliers requires more complex computations. However, datasets become larger—especially Airborne Laser Scanning (ALS) has significantly increased the size of areas that now can be investigated. Not only are the large datasets hard to handle, but it also makes the need for manual intervention a time consuming process. A high degree of automation is desired, but efficient and effective methods are needed [Papadimitriou et al., 2002], i.e. large computations on point cloud data need to be efficient.

Both scale and detection accuracy make outliers in point clouds still a critical problem. This thesis considers the problem of automatically removing outliers from ALS point clouds.

1 https://rapidlasso.com/lastools/
2 http://pointclouds.org

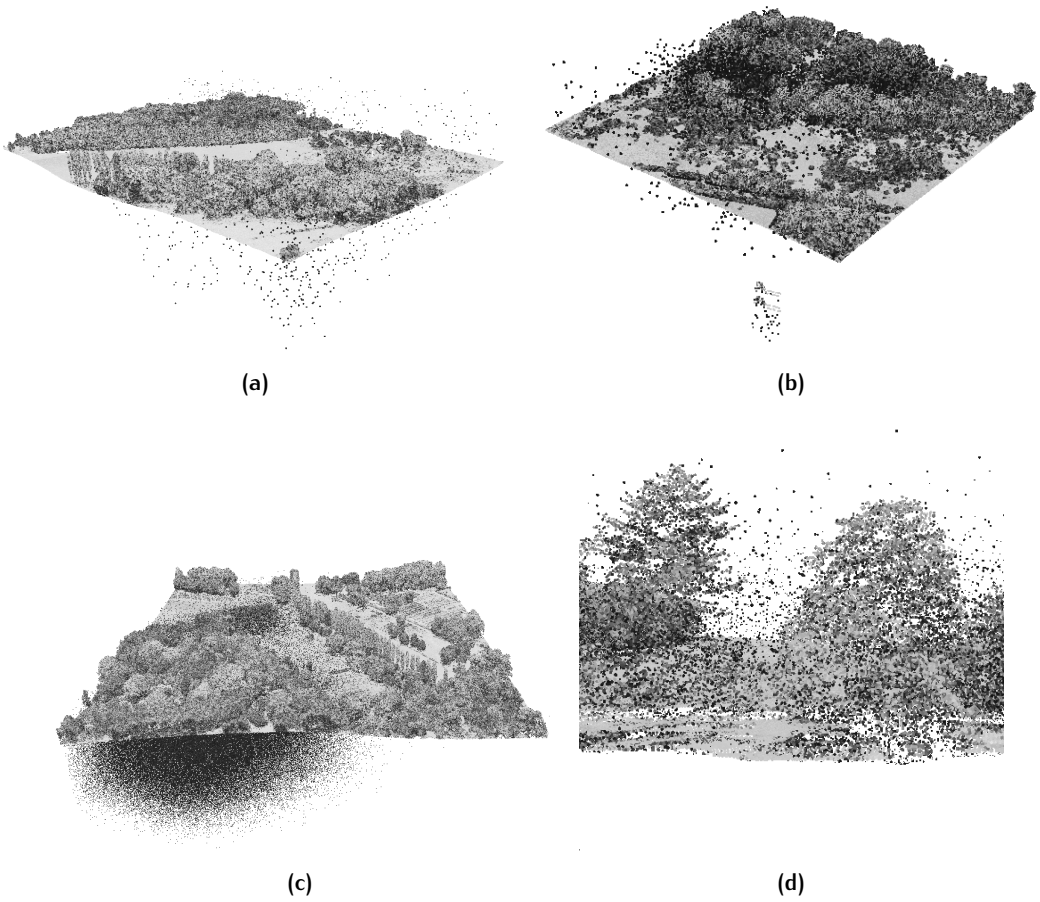


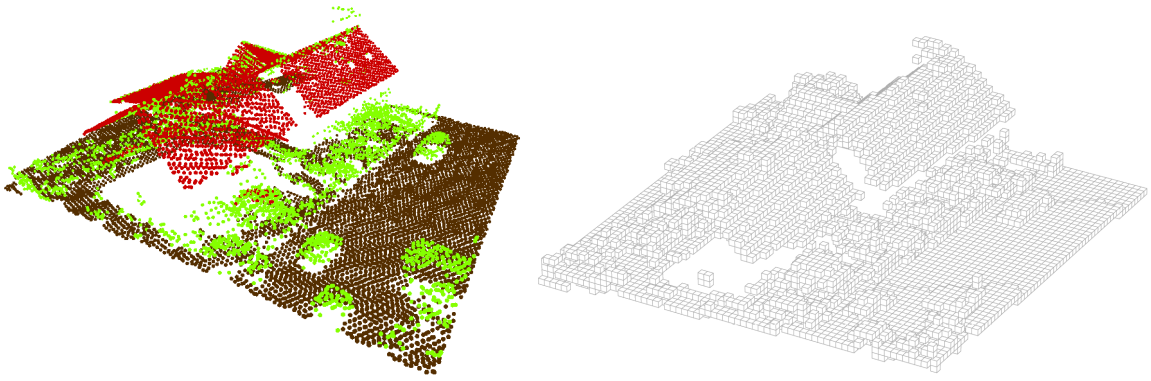
Figure 1.1: Raw ALS point clouds with various types of outliers. (a) Isolated outliers. (b) Isolated and clustered outliers. (c) Clustered cloud of outliers. (d) Outliers spatially close to objects.

1.1 RESEARCH MOTIVATION

To effectively analyze point cloud data, it first needs to be removed from outliers. This is necessary because raw data often includes a lot of errors. This is an issue for everyone who needs to carefully analyze 3D point cloud data. Deltares is a company that is facing these problems, which is why this thesis is carried out in close collaboration with them. Deltares is a Dutch based independent institute for applied research in the field of water and subsoil.

Existing software tools mainly use the density-based or distance-based approach to detect and remove outliers. However, clustered outliers require more complex computations and cannot be solved within density- and distance-based approaches. Fig. 1.3a shows a point cloud containing clustered outliers with varying local densities. Density-based methods perform poorly on this type of noise, leaving many undetected outliers (False Negatives (FN)), as is seen in Fig. 1.3b). Furthermore, low density features—such as electricity cables (Fig. 1.3c)—may be classified as outliers (False Positives (FP)), shown in Fig. 1.3d.

Filter tools have to deal with geometrical discontinuities caused by occlusions, no prior knowledge of the existence of outliers—or any statistical



(a) (b)

Figure 1.2: (a) Source point cloud. (b) Voxel structure of (a).

distribution, and varying local point densities, which makes it not a trivial task [Sotoodeh, 2007]. Moreover, large 3D datasets ask for efficient data processing.

Existing approaches make use of point-based techniques, where each 3D point is analyzed based on the features from its local neighborhood. The main drawback of this approach is the memory and execution time required to process each point in the point cloud [Plaza et al., 2015].

A second group of techniques for 3D processing is group-based techniques, which aim to speed up point cloud classification by reducing the scene to a grid or voxel model [Plaza-Leiva et al., 2017]. Instead of extracting features at the neighborhood of each point, they are extracted at each group of points defined by a voxel. An example of a voxel model from a point cloud is shown in Fig. 1.2.

Besides reducing the amount of data, translating it into a regular voxel grid (i.e. a raster structure) creates opportunities to use existing image processing techniques, such as mathematical morphology and connectivity. The full possibilities of 3D raster processing has not been studied yet for the identification of outliers in airborne LiDAR point clouds.

1.2 OBJECTIVES & RESEARCH QUESTIONS

This thesis explores the possibilities of using a voxel model from a point cloud to detect outliers in different ways. The main goal is to automatically identify outliers (i.e. isolated points—high and low, and outlying clusters) in airborne LiDAR point clouds. This can be done by (1) using the voxel grid as input for raster processing techniques and (2) extracting features from groups of points defined by voxels. This study focuses on how a voxel grid can be used in different ways to detect outliers, and how to combine multiple operations to come to a final classification. The effectiveness of the proposed method is assessed in terms of accuracy and speed.

It is not a goal to develop a method that will outperform (or even compare to) the speed of existing methods, but to provide a more accurate outlier detection tool. The corresponding main research question is therefore:

Question 1. *Is a voxel-based approach a viable option to automatically detect outliers from aerial LiDAR point clouds?*

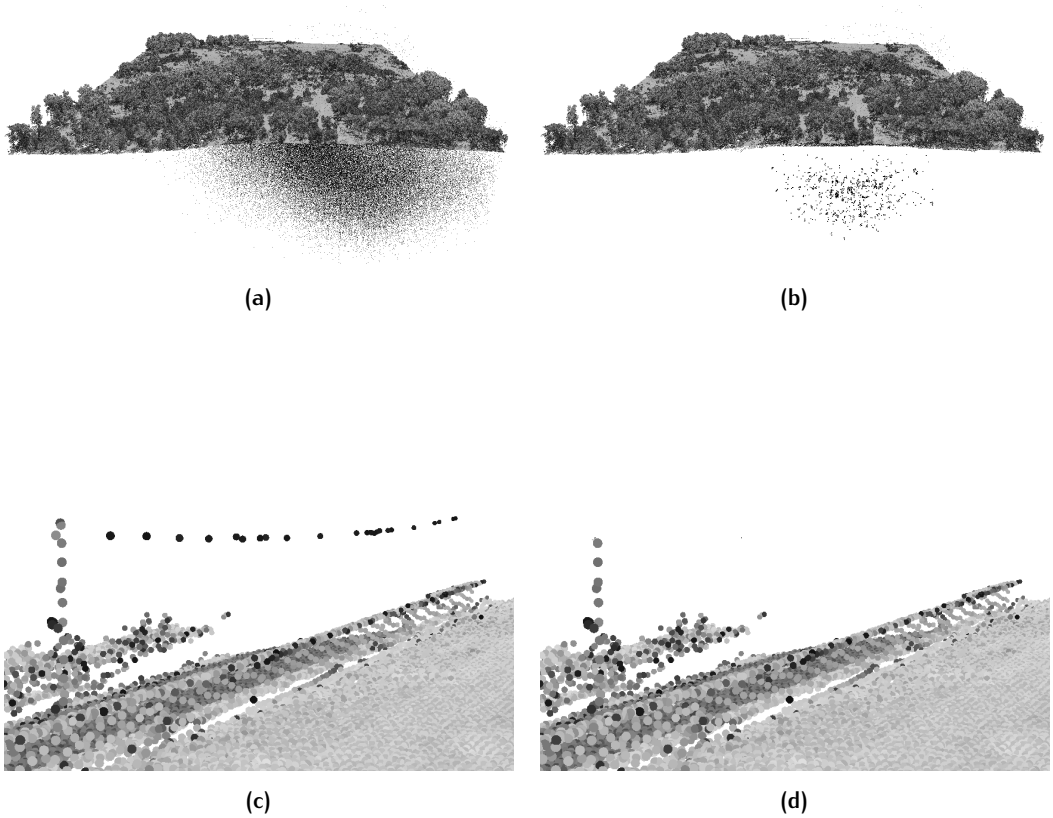


Figure 1.3: Unsatisfactory results from density- and distance-based outlier detection methods. (a) Raw source point cloud with clustered outliers. (b) Poor outlier detection of (a) by density-based method, due to varying local point densities of outlier cloud. (c) Raw point cloud of electricity cable. (d) Electricity cable removed by density-based method.

This question incorporates the entire process from a raw source point cloud to a cleaned version using a voxel structure. Besides the main research question, this study looks into individual steps and requirements and are formulated as follows:

Question 2. *How can raster processing techniques—in particular Connected Components Labeling—be used on a 3D voxelized point cloud to detect outliers?*

Question 3. *What is the influence of different voxel resolutions for outlier detection algorithms, in terms of accuracy and computation load?*

Question 4. *Can a series of analysis methods within a voxel model achieve higher classification quality?*

Question 5. *What outlier detection quality is achievable and how to influence the trade-off between true positives and false positives?*

Question 6. *What is the influence of scaling the dataset on the algorithm in terms of time and memory?*

Question 7. *Can the same outlier detection algorithm be used for both point clouds extracted from LiDAR (aerial) and Dense Image Matching (DIM)?*

1.3 SCOPE OF RESEARCH

To set the scope for this research it is important to mention that this study will use—and test algorithms on—ALS data from natural environments. Other laser scanning techniques (such as Terrestrial Laser Scanning (TLS)) will be disregarded. Urban environments are only used to see the effect of different environments, i.e. developed methods will not be specified for urban environments. Furthermore, the main focus of this study will be on combining different methods to classify outliers in a voxel grid structure, because this is where the most contributions can be made. On each voxel, different methods can be applied to extract information for outlier classification. In this thesis, it is not so much about the individual methods itself, but how to combine all this information in a raster for outlier detection.

Furthermore the research aims to develop an efficient method since it should be applicable on large data sets. It is beyond the scope of this study to benchmark different methods as much as developing a prototype which is efficiently coded. This means a prototype is developed with Julia programming language (used by Deltares) to run experiments, which is different from using C++ with more capabilities. Therefore, the search for an efficient approach will be discussed in a theoretical way. However, results from testing several approaches can be discussed, including the effect of increasing the size of the dataset.

1.4 THESIS OUTLINE

The thesis is structured in the following way:

CHAPTER 2 introduces the relevant theory for this thesis. It covers the fundamentals of LiDAR and raster processing techniques.

CHAPTER 3 describes and analyses existing approaches in point cloud outlier filtering from practice and literature. Difficulties with those approaches are illustrated and a comparison of general characteristics is made.

CHAPTER 4 is used to motivate and describe a voxelized approach with image processing methods.

CHAPTER 5 continues with the implementation and experiments of the proposed approach. A set of objective metrics is defined that are used to quantify the effectiveness of the proposed approach with respect to its fundamental requirements. Also, a comparison with a existing method is made.

CHAPTER 6 provides all conclusions that can be drawn based on this study. The research questions will be answered and the most important contributions are summarized. Furthermore, suggestions for future work are made.

2 THEORETICAL BACKGROUND

This chapter provides the necessary theoretical background for this thesis. First, the basics of airborne LiDAR ALS are given (§ 2.1), to understand why it is unavoidable to capture point clouds without outliers. Subsequently, general outliers detection methods are discussed (§ 2.2). Following, three different data models are discussed that can be used for representing point clouds (§ 2.3. Finally, processing techniques that are used for this study to extract information from a point cloud are explained. § 2.4 summarizes binary image processing techniques, which are extensively described in Jain et al. [1995]; Szeliski [2011]; Shapiro and Stockman [2001]; Birchfield [2017]. § 2.5.1 explains how Principal Component Analysis (PCA) can be used to compute the curvature (i.e. spatial shape) of a set of points.

2.1 AIRBORNE LIGHT DETECTION AND RANGING

In 1997 a nationwide DEM of the Netherlands (Actueel Hoogtebestand Nederland (AHN)[1]) was created with up to one height point per 16 m^2, using airbore LiDAR . In the meantime LiDAR has evolved and is recognized as a technology with many advantages and application purposes. LiDAR features high accuracy and precision with a high level of detail. The technology is based on a laser scanner combined with both GPS and inertial technology to create a three dimensional set of points (point cloud). Point clouds can easily contain up to millions of points per square kilometer. High levels of automation and quick data capturing make it very convenient for detailed 3D-reconstruction of the real world.

2.1.1 Discrete and full waveform LiDAR

As laser pulses move towards the ground, they hit objects such as tree branches and buildings. Some of the energy reflects off of those objects and returns to the sensor, but some energy may continue towards the ground surface. This allows one laser pulse to record multiple reflections.

The energy distribution—or intensity—that returns to the sensor creates a waveform. Areas that reflect more energy create peaks in the waveform and often represent objects such as a branch or a building. This energy distribution can be read in two ways:

1. Discrete return LiDAR systems detect peaks and record a individual point (discrete) at each peak location in the waveform. Every individual point is a return form one single laser pulse. A discrete system may record 1–4 (and sometimes more) returns from each laser pulse. Last returns are often used to quickly identify ground points.

1 http://www.ahn.nl

2. A Full Waveform LiDAR System records a distribution of returned intensity. Full waveform LiDAR data captures therefore more information than discrete systems, but are thus also more complex to process.

Usually LiDAR data is made available in the form of discrete points. In this study, only this format is considered when referred to LiDAR data.

2.1.2 Properties of LiDAR data

LiDAR point cloud data is commonly stored as `.las` format supported by the Americal Society of Photogrammetry and Remote Sensing (ASPRS). Isenburg [2013] developed the `.laz` format for point clouds which compresses large `.las` files into compact files without information loss.

Information stored on these files can vary. Obviously, point cloud data points will at least have related x, y and z coordinate values. Depending on data collection and processing methods this can be extended with more attributes. Most point cloud include at least an intensity value for each return. Some also have a classification, representing the type of object the laser returned from (e.g. ground, building, vegetation). This requires an additional processing step and various algorithms to identify objects in ALS data have been developed by researchers.

The input point cloud is defined as a list of n points. Each point p_i is defined as a vector with the Cartesian coordinates x, y, z (attributes such as intensity not considered), described as:

$$pointcloud = \begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{bmatrix} \tag{2.1}$$

2.1.3 Errors in aerial laser scanning

Outliers are basically erroneous measurements caused by the limitations of the laser scanner. Some well knows reasons are described in this section. The first is caused by the footprint of a laser beam, which is an ellipse, see Fig. 2.1a. It is possible that the beam is divided when it hits the boundary of an object. Thus the reflection value of this point would be a weighted average of the reflection from both surfaces, creating virtual points in between [El-Hakim and Beraldin, 1994]. A second reason can be caused by

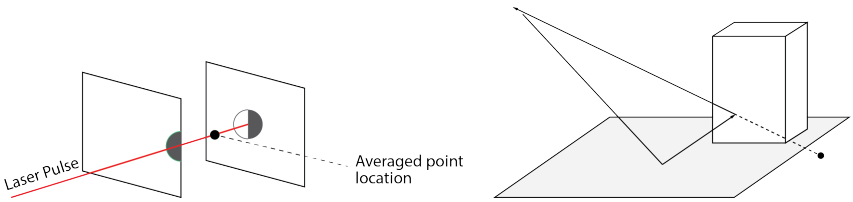


Figure 2.1: Laser scanning errors. (a) Erroneous measurement at the boundary of occlusion. The laser foot print is split over two surfaces, by which the point location is averaged between both. Fig. inspired by [Sotoodeh, 2006] (b) Multipath effect where a laser pulse is reflected of an object before it returns to the scanner. Therefore, the traveled path is longer which results in a extended point location below ground surface.

surfaces with very high or low reflectance. Such surfaces, such as black objects, glasses or metals can cause bias in the distance measurement, because the receiver cannot resolve the reflected beam [Beraldin, 2004]. The third reason is multi path reflection, see Fig. 2.1b. When the laser beam hits a surface under a angle, most of the beam can be deflected onto other close surfaces and then reflected back into the receiver. This creates a false longer distance and thus a wrong data point [Sotoodeh, 2006].

The mentioned limitations of LiDAR cause different types of outliers. Many studies distinguish two types, high and low outliers (also positive- and negative outliers) [Kobler et al., 2007]. A third type extends the list: clustered outliers.

1. *Low outliers* - These are points that normally do not belong to the surface. They come from multi-path errors and errors in the laser scanner [Sithole and Vosselman, 2004]. Normally, filters work on the assumption that the lowest point belongs to the ground.

2. *High outliers* - These are points that also normally do not belong to the surface. They originate from hits off objects like birds, low flying aircraft, or errors in the laser scanner [Meng et al., 2009; Sithole and Vosselman, 2004]. Most filters handle such outliers, because they float high in the air far from neighboring points.

3. *Outlier in cluster format* - These points are similar to high and low outliers, but form a cluster. This makes them harder to detect, because they are not isolated individual points. This asks for different detection methods, which many existing tools do not offer. They can be caused by a flock of birds, but also exist due to limitations of the sensor.

2.2 GENERAL OUTLIER DETECTION APPROACHES

Most work on outlier detection is done in the field of statistics [Barnett, 1994; Hawkins, 1980]. Algorithms in machine learning and data mining have considered outliers, but mostly in a way to deal with them in order to successfully run an algorithm [Angluin and Laird, 1988]. Hawkins [1980] defines an outlier as *an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism*. Many studies propose methods to find these data points, but most of them suffer from two problems. First, they are univariate, i.e. they only consider one variable. Point cloud data is multidimensional (at least x, y, z), making it unsuitable. Secondly, in all cases you have to perform expensive testing to find a distribution that fits the data [Johnson et al., 1998]. This makes outlier detection in point cloud data different than what we are used to from statistical methods. Outliers are caused from different sources and are seen as measurements that differ from their local neighborhood, i.e. data points that do not fit the local surface geometry [Sotoodeh, 2006]. Different approaches are studied to identify these outlying points.

Papadimitriou et al. [2002] classifies five different categories of filters as distribution-based, depth-based, distance-based, density-based and clustering approaches.

DISTRIBUTION-BASED METHODS deploy some standard distribution model (e.g. Normal distribution) and classifies outliers those points that de-

viate from the model. However, for arbitrary datasets without prior knowledge of the distribution of points (e.g. point cloud datasets), finding the which model fits the data best. Local surface fitting approaches, for instance moving least squares or RANSAC, are also used for outlier detection. These methods can work well when a point cloud is dense and is obtained from a smooth surface, e.g. urban environments. Point clouds with discontinuities or high curvature areas, for example natural environments, are not suited for surface fitting [Sotoodeh, 2007].

THE DEPTH-BASED APPROACH is based on computational geometry and computes different layers of k-dimensional convex hulls [Johnson et al., 1998]. Objects in the outer layer are identified as outliers. In these approaches, based on some definition of depth, data objects are organized in layers in the data space, with the expectation that objects in the outer layer are more likely to contain outlying points. To avoid the above mentioned problems of distribution fitting and restriction to univariate datasets, depth-based approaches have been developed [Johnson et al., 1998]. However, algorithms for this method cannot cope with large three-dimensional datasets [Sotoodeh, 2007].

CLUSTERING ALGORITHMS are not optimized for outlier detection, since the main objective is clustering. However, they can be used for this, but outliers are by-products, i.e. don't belong to a cluster [Jain et al., 1999]. Teutsch et al. [2011] clusters point cloud data to identify sets of neighboring points that belong to the same region. Most of the resulting regions contained less than 10 points. They considered regions with less than 1000 points outliers, and removed all regions below this threshold. Sithole [2005] segments ALS data to separate terrain, house roofs, bridges and trees. Points that do not belong to a predefined class are labeled as outliers.

THE DISTANCE-BASED APPROACH is originally proposed by Knorr et al. [2000]. An point in a data set P is a distance-based outlier if at least a fraction β of the points in the local neighborhood is further than r from it. This outlier defintion is determined by two parameters, r defines the distance and β defines a threshold for number of data points. Many existing point cloud filters are based on this approach. TerraScan, PCL and CGAL implemented filters which removes all points from the input point cloud that do not have at least some number of neighbors within a certain range. It performs well on 3D point cloud data to detect isolated points. However, this can lead to problems when the data set has both dense and sparse regions [Breunig et al., 2000].

THE DENSITY-BASED APPROACH is proposed in Breunig et al. [2000]. It was observed that taking a global view of the dataset captures only certain kinds of outliers. However, real-world datasets—which can exhibit a complex structure—can have objects that are outlying relative to their local neighborhoods. These *local* outliers can be seen as objects, which depends on the local density of its neighborhood. The neighborhood is defined by the distance to n nearest neighbors. Value n is a predefined value and determines the size of a local neighborhood, used to calculate the density. This method works without prior knowledge of the distribution of points and handles different local point densities.

The distance- and density-based approach attract more attention for outlier detection, because they are more appropriate for high dimensional, large data sets [Papadimitriou et al., 2002]. This is also reflected in the available point cloud outlier detection methods on the market, e.g. TerraScan, LAStools and PCL have routines based on distance and/or density of neighboring points.

2.3 3D DIGITAL REPRESENTATIONS

LiDAR technology aims to measure the properties of the earth's surface in order to create a model out of it. Since we can only take a finite number of measurements with a finite amount of accuracy, such a model is limited to some degree. But even though a complete and accurate model seems impossible, an approximation of the surface's properties can be of great use.

There are several ways to represent these measurements in a digital model. A surface is commonly represented as a piecewise tessellation [Goodchild, 1992]. By dividing the surface in pieces it becomes possible to show it in digital form. One approach to reconstruct a surface from a point cloud is by triangulation of the points (§ 2.3.1). Another possibility is to divide the space in a rectangular pieces—instead of triangles—such as with rasters (§ 2.3.2).

The choice of a data model also incorporates the implementation of a data structure. A data structure can ensure topological relations depending on the data model. When choosing for a particular model, one should also consider storage efficiency, scalability and speed. Note that in this thesis the raster model lies at the core. However, a brief explanation of a triangulation helps to motivate this choice.

2.3.1 Triangulation

Triangulation in geographical information system (GIS) refers to triangular irregular network (TIN), which is a triangular subdivision of a plane. It is an effective structure to model the terrain surface measured with for instance LiDAR technology, which is also called a DSM. The vertices in a TIN are formed by points in the source point cloud, creating a vector-based representation (see Fig. 2.2a). An advantage of using a TIN over a rasterized model is that geographical locations of sample points are represented with a vertex in the digital model. Moreover, triangulation is adaptive to varying point densities and patterns.

When given a set of points, there exist many candidate tessellations. For instance, some have a higher proportion of thin, long triangles and may be judged inferior to others. A highly desired triangulation is the Delaunay triangulation (DT) with triangles as equilateral as possible [Worboys and Duckham, 2004].

A TIN or DT implies topology between adjacing vertices, edges and faces. To implement such topology a list is defined with information of each triangle, its vertices—including coordinates, edges and adjacent triangles. The implementation details are more complex than defining neighboring pixels in a raster structure (only in a structural sense and not morphological correspondence). Although vector data is useful for topology rules and network analysis, it comes at the cost of intensive processing [Kumler, 1994].

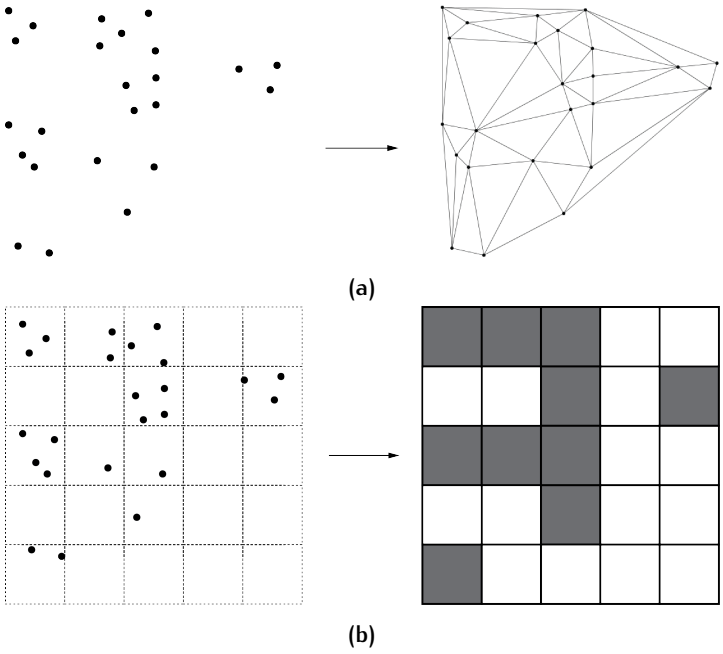


Figure 2.2: (a) Representing a set of irregularly distributed points using a (Delaunay) triangulation. (b) Representing a set of irregularly distributed points in a raster structure.

2.3.2 Rasterization and Voxelization

Raster data is a spatial model that defines space as an array of equally sized cells in rows and columns. Every cell or pixel contains a value and location in the the raster. The value of a pixel can be discrete or continuous, and depends on the data acquisition method. For instance, in remote sensing, a pixel can have an integer value for a specific land cover class. But in a LiDAR derived DEM, each cell represents an elevation value for that location on the earth. Therefore, the exact meaning of every cell varies and is not uniquely defined [Fisher, 1997].

When rasterizing vector objects, each pixel can be treated as binary labeling of the space as empty or occupied. Figure 2.2b shows the rasterization of point vector objects ($0D$).

The same can be done with all inputs embedded in three-dimensional Euclidean space, i.e. x, y and z coordinates. Space partitioning of LiDAR point clouds in a regular grid can be done in the same way as rasterizing a 2D vector object. However, opposed to TIN and Delaunay Triangulation, exact point locations will be lost in raster cells, i.e. the voxel does not directly correspond to the point coordinates. The spatial resolution or voxel size greatly influences the voxelized reconstruction. Significant geographical features might be lost when a large voxelsize is chosen, while other voxels might not even contain any samples at all. Chosing a small voxel size might represent the geographical features more accurate, achieving a higher level of detail (LoD). However, rasters can never represent a continuous space, because of discrete jumps between voxel values, i.e. voxelized objects always have bias in the estimation of surface areas and cause misrepresentation for objects not aligned on the grid (unless the voxels are infinitely small, which is impossible). Moreover, The computational cost of a voxel grid can grow cubicly with the desired level of detail [Boulch et al., 2014]. In addition, ge-

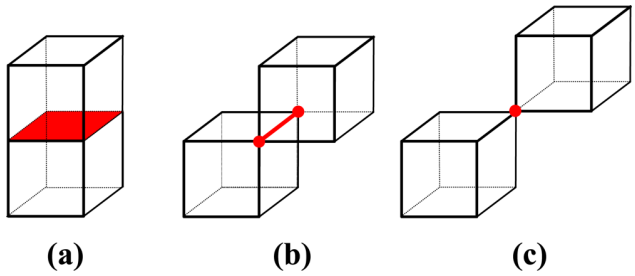


Figure 2.3: (a) Shared face. (b) Shared edge. (c) Shared vertex. Figure from Wu et al. [2013].

ographical surface captured by LiDAR can differ notably between places. A fixed spatial resolution from a regular grid disregards this variability. For these reasons, the spatial resolution of a raster should be carefully selected, and attune with the data samples it represents.

Figure 2.3 shows that a voxel consists of six faces, twelve edges and eight vertices. When adjacent voxels share a face, edge or vertex they are connected. These topological relations are described in § 2.4.1 in relation to raster processing. Although, voxels are spatial cubics in 3D Euclidean space, they are not stored as such. In contrast to a vector data set, where point locations (coordinates) are recorded explicitly, the voxels locations is defined implicitly by its position in the array (its index). Computer systems benefit from this structure in two ways: (1) reduced storage memory capacity, because coordinates of every voxel are not stored; (2) essentially, a raster is a 2D array and a voxel grid a 3D array, and these basic data structures are straightforward and efficient to implement in processing algorithms.

However, the potential of a raster model for computer processing can not be the motivation for choosing such as model. As Fisher [1997] points out, the raster model may be the best option for real world modeling, but the meaning of each pixel and the consequences of different samplings and resolutions should be known.

Usually, the whole space is not filled with objects, causing many empty voxels in a grid. It sounds obvious that iterating over objects is more efficient than iterate over all possible voxels in a bounding box. Therefore, voxels can be stored in two different ways:

A DENSE ARRAY: which behaves as a typical matrix where each cell in the array represents a value. Every cell is positioned at a row and column number, and can therefore be easily accessed. For an $M \times N$ array, the memory required is related to $M \times N$. In this data structure topological relations are stored implicitly.

A SPARSE ARRAY: this is an array in which many cells have a value of zero. Sparse arrays on a computer requires significantly less storage space. This is realized by only storing the non-zero cells. Fig. 2.1 shows an example where only non-zero values are stored in a three column representation. However, the trade-off is that accessing individual cells are not as straightforward anymore as accessing a dense array by two indices. Also, topological relations have to be stored explicitly.

$$M = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 3 & 0 & 0 & 0 \end{bmatrix}$$

Row	*Column*	*Value*	
4	**4**	**3**	= total
1	2	1	
3	3	4	
4	1	3	

Table 2.1: Only non-zero values of example matrix M are stored in a three column representation of a sparse array.

The fact that a rasters is so straightforward and efficiently represented on a computer, has given rise to many raster-based algorithms that perform spatial analysis. This is probably also why raster data has become very common in GIS. The next section (§ 2.4) goes into the processing of raster data. Many processing techniques are present today for a wide range of applications, often originated from image processing. Section 2.4 covers only a small portion, relevant for this thesis.

Noteworthy, are the many names that are used in literature for a raster and its cells. In this thesis, I may use raster, array, and grid interchangeably, as well as pixel and cell (2D) and cell and voxel (3D).

2.4 BINARY RASTER PROCESSING

Within the field of computer vision, image processing techniques are used for gaining high-level understanding from digital images. An image contains a continuum of intensity values, or gray values. Every pixel contains its own gray level, most commonly represented by image intensities of 256 different gray levels. Evidently, more different intensity levels allow for a more accurate representation of the actual scene. However, this goes at the cost of more storage. For many applications a much simpler representation is enough for extracting useful information, e.g. counting objects and finding connected pixels [Shapiro and Stockman, 2001]. A binary image contains only two gray values: 0 and 1. This has an advantage over color images: computing properties of binary images tend to be less expensive and faster than image processing systems that operate on gray level or color images [Jain et al., 1995]. Logically, because of significantly smaller memory and processing requirements. In addition, as described in Section 2.3.2 boolean values can label empty and non-empty voxels.

In this section two basic binary vision operations are described. They will serve as a foundation upon which we can build in later chapters. First, the basics of image processing are described. Secondly, the connected components labeling operator is discussed, which is used to give each separate connected group of pixels a unique label. Thirdly, mathematical morphology is introduced. These operators can be used for several tasks, such as closing holes, image thinning and thickening, but most relevant for this thesis is to join and separate components.

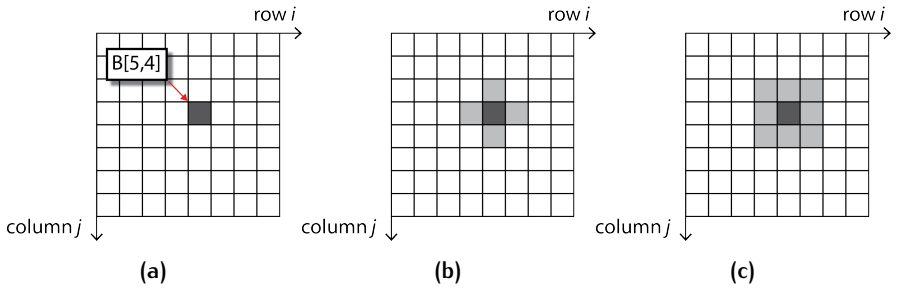


Figure 2.4: Pixel and pixel neighborhoods. (a) Binary image B with a *true* pixel at row 5, column 4. (b) 4-neighborhood of pixel [5,4]. (c) 8-neighborhood of pixel [5,4].

2.4.1 Pixels, Neighborhoods & Topology

Pixel

In image processing, gray scale or color images need to go through an operation to segment object pixels from background pixels. This can be a difficult problem and will not be addressed here. For explaining the following concepts, we will assume that the initial input for the tasks is a binary image B. A pixel of a binary image B is located at row i, column j of the image array. Subsequently, in Fig. 2.4a the value of the pixel at row i, column j is denoted with $B[i,j]$, and can only be 0 or 1. A two-dimensional $M \times N$ image has M rows and N columns, and is nothing more than an array with values, much like a matrix. Arrays can have more dimensions, and essentially, every operation possible for 2D arrays also works for 3D arrays. In the 3D raster domain a pixel would be a voxel, as explained in § 2.3.2. The 3D space is more relevant when working with 3D point clouds, but this section shall examine 2D image arrays for simplicity. Primarily, because the key concepts are easiest to introduce in 2D, and then can simply be extended to 3D.

Neigborhood & Topological Properties

The location of a pixel in a image array with location row i and column j is spatially close to several other pixels. That is to say, a pixel has a common boundary with four pixels and a common boundary or corner with eight other pixels. This neighborhood is important in image processing, because for many algorithms, not only the value of a single pixel, but also the values of its neighbors are used at the same operation [Shapiro and Stockman, 2001]. The two most common neighborhoods for a pixel in a 2D digital image are:

4-NEIGHBORHOOD: if two pixels share a common boundary, so that $N_4(i,j)$ of pixel (i,j) includes pixels $(i-1,j)$, $(i+1,j)$, $(i,j-1)$, and $(i,j+1)$, which are often referred to as its north, south, west, and east neighbors (Fig. 2.4b).

8-NEIGHBORHOOD: if two pixels share at least one corner; $N_8(i,j)$ of pixel (i,j) includes each pixel of the four-neighborhood plus the diagonal neighbor pixels $(i-1,j-1)$, $(i-1,j+1)$, $(i+1,j-1)$, and $(i+1,j+1)$, which can be referred to as its northwest, northeast, southwest, and southeast neighbors (Fig. 2.4c).

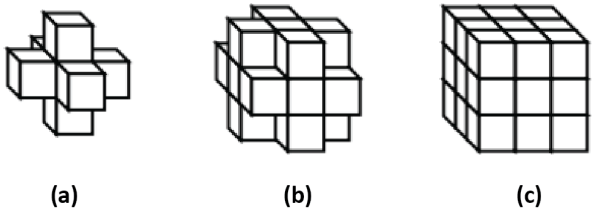


Figure 2.5: Neighborhoods in 3D voxel space. (a) 6-neighborhood. (b) 18-neighborhood. (c) 26-neighborhood.

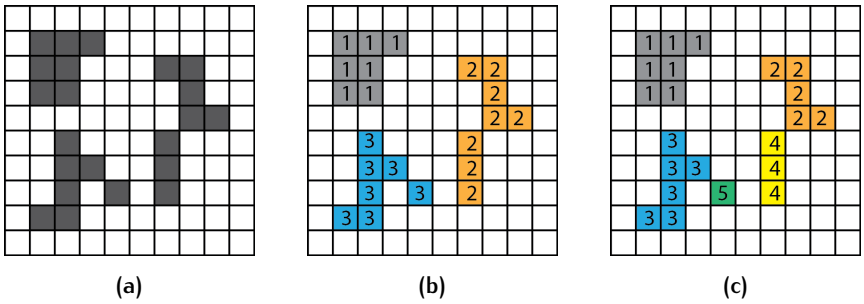


Figure 2.6: Connected Components Labeling of binary raster. (a) Source binary raster. (b) Connected Components Labeling with 8-connectivity. (c) Connected Components Labeling with 4-connectivity.

Similarly, we say a pixel is 4-connected (or 4-adjacent) to its 4-neighbors and 8-connected to its 8-neighbors. This idea can be extended to the 3D voxel space based on the definitions of connectivity given below:

6-CONNECTED a collection of voxels in which every voxel shares at least one face with an adjacent voxel. (Fig. 2.5a).

18-CONNECTED: a collection of voxels in which every voxel shares at least one edge with an adjacent voxel. (Fig. 2.5b).

26-CONNECTED a collection of voxels in which every voxel shares at least one vertex with an adjacent voxel. (Fig. 2.5c).

A set of pixels in which each pixel is connected to all other pixels is called a *connected component* [Jain et al., 1995]. How connectedness between pixels can be used to separate objects is further described in the next paragraph.

2.4.2 Connected Components Labeling

In computer vision CCL is used to detect regions (or objects) in binary digital images. A connected set of pixels (i.e. pixels with the same value and spatially adjacent) in a image is called a region, and possibly represents an object [Jain et al., 1995]. Connected components labeling can be used in a variety of applications, but often it is used for counting objects and compute its characteristics (e.g. size, position, orientation and bounding rectangle) [Birchfield, 2017]. Other applications can be separating individual letters in a scanned document [Szeliski, 2011]. The algorithm is based on graph theory, and therefore not limited to digital images, e.g. Heinzel and Huber [2016] finds connected components in a voxel grid.

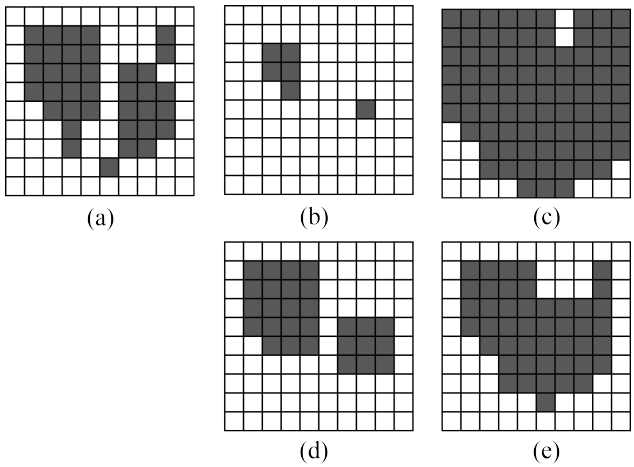


Figure 2.7: Mathematical morphology on binary raster data with a 3×3 structuring element. (a) source binary raster. (b) Erosion. (c) Dilation. (d) Opening, i.e. erosion followed by dilation. (e) Closing, i.e. dilation followed by erosion.

In raster data a connected component consists of a set of pixels in which each pixel is connected to all other pixels [Jain et al., 1995]. A connected components labeling of a binary image, is a labeled image in which the value of each pixel is the label of its connected component [Shapiro and Stockman, 2001]. So, consider the binary image Fig. 2.6a. We can define 4- and 8-connected components, depending on the type of adjacency selected. For example, in Fig. 2.6b the components are 8-connected, resulting in three regions and in Fig. 2.6c they are 4-connected resulting in five regions.

Two-pass algorithm

To compute the connected components of a raster, suppose we have binary raster B. In order to identify connected regions, the algorithm makes two passes. On the first pass the operator scans the raster pixel by pixel until it comes to a point p where $B(i, j) = 1$. It then examines the four neighbors that are already passed in the scan, i.e. the west, north-west, north and north-east pixel (considering 8-connectivity) and labels the pixel p as follows:

1. If all four neighbors are 0, assign a new label to p, else
2. If only one neighboring pixel has value 1, assign pixel p with the same label, else
3. If more than one neighboring pixel has value 1, assign the lowest label to p and add a note to the list of equivalences.

After completing the first pass, a second scan is made through the raster, during which each label is replaced by the label assigned to it in the equivalence list.

2.4.3 Mathematical Morphology

Morphological processing changes the form or structure of a region in an image. The basic operations in mathematical morphology are performed over

a neighborhood specified by a structural element (or kernel). The structuring element represents a shape; it can be of any size and form represented by another binary image, usually much smaller than the input image (i.e. a subset of the space or grid). However, there are a number of common structural, mostly it is an $(n \times n)$-window. This window is shifted over the image and at each pixel of the image, the window is compared with the set of the underlying pixels. Translations of the structuring element can be placed anywhere on the image and can be used to either enlarge a region by that shape (dilation) or to make it smaller (erosion) [Shapiro and Stockman, 2001].

The basic operations of binary morphology are *dilation* (Fig. 2.7c) and *erosion* (Fig. 2.7b). Dilation enlarges the source region shown in Fig. 2.7a, erosion makes it smaller. Based on erosion and dilation, two other operations, *opening* (Fig. 2.7d) and *closing* (Fig. 2.7e), can be derived. Opening is just another name for erosion followed by dilation, and closing is the reverse of opening (i.e. dilation followed by erosion). A closing operation can close up holes in the image (depending on the size of the structural element). Opening can get rid of small portions of the region that jut out from the boundary into the background region [Shapiro and Stockman, 2001]. The mathematical definitions of the four basic operations are described below:

Definition 1. *The **dilation** of binary image B with structuring element S is denoted by $B \oplus S$ following:*

$$B \oplus S = \bigcup_{b \in B} S_b \tag{2.2}$$

The union in Equation 2.2 is a neighborhood operator. The structuring element S moves over the image array. Each time the origin of the structuring element (i.e. the center pixel) touches a binary 1-pixel, all underlying pixels of the structural element shape are set to 1 in the output array.

Definition 2. *The **erosion** of binary image B with structuring element S is denoted by $B \ominus S$ following:*

$$B \ominus S = \{b \mid b + s \in B \, \forall \, s \in S\} \tag{2.3}$$

Definition 3. *The **opening** of binary image B with structuring element S is denoted $B \circ S$ following:*

$$B \circ S = (B \ominus S) \oplus S \tag{2.4}$$

Definition 4. *The **closing** of binary image B with structuring element S is denoted by $B \bullet S$ following:*

$$B \bullet S = (B \oplus S) \ominus S \tag{2.5}$$

Mathematical Morphology in 3D

These raster operations can easily be translated into the 3D domain. Gorte and Pfeifer [2004] designed a method based on 2D mathematical morphology raster processing, and transferred it into the 3D domain. Connectivity and neighborhood relations between voxels in a 3D raster can be established much easier than between the original (x, y, z)-points, using morphological operations. Closing and opening are applied to close gaps and holes, and remove isolated voxels. The shape and size of the structuring element controls the maximum size of holes that can be repaired. This means that choosing

the shape of the structuring element influences the outcome of the operation. So is it possible to indicate a preferred direction for the operations, e.g. connecting voxels above each other rather than voxels at the same height [Gorte and Pfeifer, 2004].

2.5 LOCAL POINT STATISTICS

Outside the raster domain, point-based techniques are common where each 3D point is analyzed based on its neighborhood.

2.5.1 Planarity of 3D distributed points

Knowledge of geometric features is useful for object recognition [Plaza-Leiva et al., 2017; Plaza et al., 2015; Zhuang et al., 2015; Xiong et al., 2011], for instance elevation and flatness characteristics can be used to classify roof surfaces and urban objects [Sun and Salvaggio, 2013]. In this context, the spatial shape of a group of points inside each voxel could potentially be useful for outlier detection. The identified types of outliers usually appear random in the point cloud, i.e. outliers form a scattered region and rarely fit in one plane.

Classification of geometric features based on PCA is widely used in recent point cloud processing methods [Maligo and Lacroix, 2017; Lehtomaki et al., 2016; Hao and Wang, 2014]. The shape of a group of points (i.e. planar, tubular or scatter shape) is identified by a dispersion indicator called covariance matrix [Lalonde et al., 2006]. Conceptually, one fits a plane to neighborhoods and measures the goodness of this fit. From a neighborhood with N points $\{X_i\} = \{(x_i, y_i, z_i)^T\}$ with $\overline{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$ the symmetric positive covariance matrix is defined as:

$$\frac{1}{N}\sum_{i=1}^{N}(X_i - \overline{X})(X_i - \overline{X})^T \tag{2.6}$$

The matrix is decomposed into principal components for analyzing data distribution using the eigenvectors and eigenvalues, referred to as PCA. Eigenvalues for each neighborhood are sorted in ascending order: $\lambda_0 \geq \lambda_1 \geq \lambda_2$. The smallest eigenvector of this matrix will indicate the normal vector of the plane best fitting to this neighborhood, because it corresponds to the direction of least variation. This means, in case of planar points $\lambda_0 \simeq \lambda_1 \gg \lambda_2$ (Fig. 2.8b). When variation is similar in all directions, such that $\lambda_0 \simeq \lambda_1 \simeq \lambda_2$, its region is scattered (Fig. 2.8a).

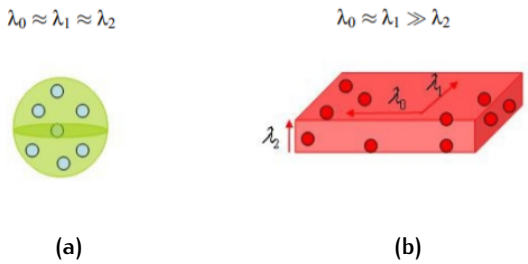


Figure 2.8: Geometric properties of a neighborhood, from [Lalonde et al., 2006]. (a) Scatter shape. (b) Planar shape

3 RELATED WORK

Point cloud filtering has been an on-going research problem in several fields, including computational statistics, computer graphics and more. As a result many filtering methods have been proposed. Here, only studies associated with large scale point cloud filtering are given attention. Methods for ALS captured data sets are given highest priority, but also some other relevant literature is mentioned. Firstly, common tools that are used today for filtering point clouds are described (§ 3.1), § 3.2 continues to discuss previous studies. This chapter ends with a conclusion (§ 3.5) to match the existing methods with the goals of this thesis, and frame the gaps that need further research.

3.1 EXISTING FILTERING TOOLS

Outlier detection and removal software is already available in different forms. Many software products designed for LiDAR data analysis have a tool for removing outliers. The following tools are widely used:

LASTOOLS [1] is a software package with several different functions for LiDAR data, e.g. classify, convert, tile and remove noise. This specific noise detection function is called lasnoise. This tool flags or removes outliers from `.laz/.las` files. The function searches for isolated points and can be categorized as density-based approach. The tool tries to find points that have only a few points in their surrounding $3 \times 3 \times 3$ grid of cells (i.e. its 26 neighbors). Cell size and maximum number of points within the cells must be determined by the user.

POINT CLOUD LIBRARY [2] (PCL) is open source project for point cloud processing. It includes two outlier removal tools categorized as distance-based approach. The first computes the mean distance to all its nearest neighbors. When this distance falls outside an interval defined by the global distances mean and standard deviation, it is considered an outlier. The second method counts the number of neighbors within a specified radius for every point. Points that do not have enough neighbors in its radius are flagged as outliers.

TERRASCAN [3] is software for LiDAR data processing by TerraSolid. It has a similar tool for outlier detection as PCL. It searches for isolated points, by counting the number of neighbors in a specified radius. It also has a tool for detecting low points below the ground level due to double reflection. TerraScan is often used in practice. For example, parts of AHN3 are removed from outliers by the help of this tool.

1 http://lastools.org
2 http://pointclouds.org
3 http://www.terrasolid.com

CGAL [4] the Computational Geometry Algorithms Library is available under an open source license. It contains implementations of many common point cloud processing and analyzing algorithms. Processing includes smoothing, simplification and outlier removal. The outlier removal function deletes a user-specified fraction of outliers from an input point cloud. The function does this with a distance-based approach: all points are sorted in increasing order of average squared distances to their *k*-Nearest Neighbors (kNN). When this value exceeds a specified threshold, it is considered an outlier.

The available tools are able to detect outliers in an efficient way, due to its simplicity. However, they all share one major issue: outlying clusters of points are not identified. They also require a lot of parameter adjustment by the user to gain maximum result.

3.2 PREVIOUS WORK—OUTLIER DETECTION

Besides the tools mentioned in § 3.1, work has also been done from a research perspective and is given attention is this section. Unfortunately, outlier detection in ALS data is not paid much attention, and a few different methods are proposed. Related work will be divided into one of the categories defined in § 2.2. Clearly, a substantial amount of proposed methods can be categorized in the distance-based (§ 3.2.1), density-based (§ 3.2.2), distribution-based § 3.2.3 and clustering (§ 3.2.4) category. Methods based on mathematical morphology are covered in § 3.2.5. Other related work that does not fall into a single category—or any category at all—is discussed in § 3.2.6.

3.2.1 Distance-based Methods

Together with density-based methods, is the distance-based approach most implemented for filtering outliers in point cloud data. Shen et al. [2011] detects outliers by calculating the mean distance for every point and its *k*-nearest neighbors. If the average distance is larger than an adaptively predefined value, the point is regarded as an outlier. Finding nearest neighbors is a heavy computation for a large data set, such as a point cloud. Therefore, they construct a *k*d-tree of an airborne LiDAR point cloud data set to efficiently find the *k*-nearest neighbors. This method is also able to find clusters of outliers by choosing a *k* larger than the cluster. Large computational cost is often a problem when analyzing 3D data. Many processing steps involve the local neighborhood for every point in the point cloud.

3.2.2 Density-based Methods

The density-based approach proposed by Breunig et al. [2000] is introduced in § 2.2. Outliers can be seen as isolated points, which depends on the local density of its local neighborhood. They introduce a Local Outlier Factor (LOF) for each object in the dataset, which is the degree the object can be considered an outlier. The LOF is dependent on a single parameter to determine the number of nearest neighbors to define the neighborhood of an

4 https://www.cgal.org

object. The algorithm compares the density of a data point, with the density of its nearest neighbors. Subsequently, the lower the local density of the data point, and the higher the local density of its nearest neighbors, the higher the LOF value for this specific data point. Points with a high LOF value are considered outliers, because they deviate from the local point density. Sotoodeh [2006] adopted the algorithm for outlier detection in ALS point clouds and knows implementations in Eisenbeiss [2009]. The local behavior of the LOF does not suffer from varying densities in the point cloud. The algorithm successfully detects single isolated outliers, but it does not detect outlier clusters with the point density higher than its k-nearest neighbors (where k is a predefined threshold). The detection of cluster outliers with lower density than k seems satisfactory. While the algorithm detects a part of outliers, the problem of the detection of the cluster outliers is still a challenge.

3.2.3 Distribution-based Methods

Distribution of elevation values is commonly used to identify outliers as a pre-processing step for ground filtering algorithms [Silván-Cárdenas and Wang, 2006; Meng et al., 2009]. An elevation histogram distribution is computed and shows the elevation range of ground and above-ground features. Points with elevations out of range are considered outliers. Elevation thresholds are set up to eliminate the lowest and highest tails from the distribution. Remaining outliers are detected by comparing elevation values of each point and compare them with all its neighbors. Delaunay triangulation was used to define the neighbors of each point. When a point is too high or too low compared to its neighbors, it removed from the dataset. It is a popular method to quickly remove a number of outliers, but it lacks a solution to detect clustered outliers or less obvious single outliers. This is mainly due to an arbitrary threshold used to examine the height difference. For example, points from trees can potentially be much higher than its triangulation neighbors. Meng et al. [2009] noticed this and coped with it by using a twice as high threshold for high outliers compared to low outliers. Evidently, this requires many unwanted trial and error iterations to have a satisfactory result.

3.2.4 Clustering Methods

Sotoodeh [2007] presents a new algorithm to detect single and clustered outliers. The method works in a hierarchical setup, by first applying a global view and then a local view on the dataset. In more detail, the first step provides a rough global approximation of the sampling intervals over the EMST. Initially, the Delaunay triangulation of the point cloud is computed. The underlying topology of this graph can be used to construct the EMST (Fig. 3.1b), and in the second phase GG (a detailed motivation for these graphs is given in Sotoodeh [2007]). Using the Delaunay triangulation structure reduces the complexity to $O(n \log n)$ for each. Based on edge lengths in the EMST, tree edges not in a predefined confidence interval are pruned. This disconnects clusters if their connecting edge is too large; creating a rough clustering of the point cloud and eliminating global outliers (Fig. 3.1c). In the second step, for every cluster a GG is constructed (Fig. 3.1d) and statistics based on its edge lengths is computed. Finally, edges not in a predefined confidence interval are removed (Fig. 3.1e). This results in removing single

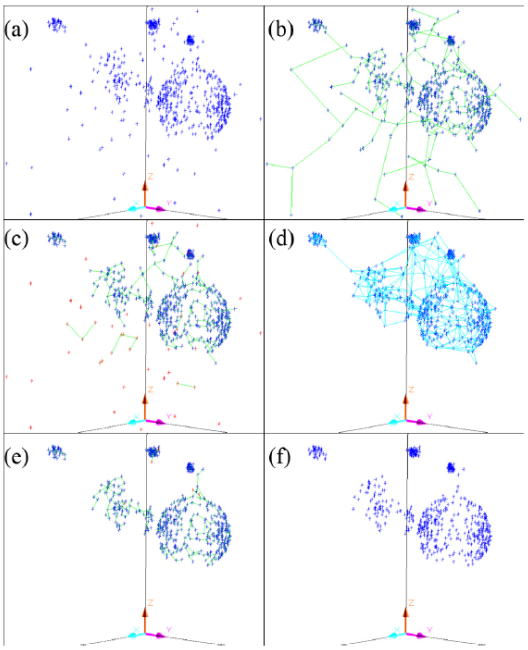


Figure 3.1: Algortihm proposed by- and figure obtained from [Sotoodeh, 2007]. (a) Source data. (b) EMST of the source data. (c) Pruned EMST by 99% confidential interval. (d) GG of the clusters of the first phase. (e) Pruned GG by 95% confidential interval. (f) Resulting dataset removed from outliers.

outliers; clusters with less point density than a threshold are also removed. The algorithm is tested on a simulated point cloud and TLS point cloud. In both cases it successfully detected single and clustered outliers. However, this does require a predefined inlier cluster density, i.e. only clusters with less points than this threshold are considered outliers, making it an arbitrary decision. Furthermore, the choice of constructing a DT limits the scene to a 2.5D model, which may cause difficulties in some real world situations where objects appear above each other (e.g. electricity cables).

A similar approach to identify outliers in sonar point clouds is proposed by Arge et al. [2010]. Although the acquisition of the data is different from ALS point clouds, outliers appear in a very similar way. Three different types of outliers were identified in the sonar dataset: (1) points that appear at random above and below the seabed; sometimes in larger groups (single and clustered noise); (2) points resulting from physical objects such as fish, forming larger groups of outliers (clustered noise); (3) structural noise, often in the form of ribbons (or strings) of points (clustered noise). The proposed method is designed to detect all types of noise. It computes a Delaunay triangulation of the points, and then removes edges of this graph (embedded in $\mathbb{R}^3$) where the difference in z-coordinate is larger than a threshold. The resulting graph is used to compute the connected components, and finally all vertices that do not belong to the largest component are labeled as outliers. The motivation behind the connected components analysis can be best described with an example. When a group of fish swim above the seabed with a height difference more than the threshold, its edges connecting the fish with the seabed are removed and the points on the fish form a small connected component (and is labeled as noise). Interestingly, points on a pipeline above the seabed with the same height difference as the fish are not detected as outliers, because its graph is connected to the seabed at some

point, i.e. it forms the same connected component as the seabed and is thus not labeled as noise. The same concept could potentially be used in ALS point cloud data, where similar types of outliers occur, i.e. outliers are not connected to the ground (e.g. birds, high and low noise).

3.2.5 Methods based on Mathematical Morphology

Mathematical morphology (described in § 2.4.3) is often used to filter LiDAR data and separate ground points from non-ground points [Li et al., 2017b; Quan et al., 2016; Pingel et al., 2013; Chen et al., 2007; Kobler et al., 2007; Sithole and Vosselman, 2004; Zhang et al., 2003; Kilian et al., 1996]. Outliers are seen as non-ground points and have to be filtered from the dataset. Kilian et al. [1996] proposed a method to remove remove non-ground points using a morphological operator on a gridded surface. This method first detects the point with the lowest elevation within a window size after an opening operation. Points within this window that fall a certain range higher are labeled as ground points. The selection of the window size determines the success of this method. Outliers are removed, but depending on the window size, also cars and trees (it was the objective of Kilian et al. [1996] to filter all non-ground points). This method is further developed by Zhang et al. [2003] including a progressively increasing window size to remove objects—of all sizes—with the opening operation. An elevation difference threshold based on elevation variations of the terrain was introduced to distinguish unwanted objects from terrain height differences. Quan et al. [2016] and Pingel et al. [2013] used this method to remove low outliers, by using a small window size and a high threshold for the terrain elevation difference. Obviously, this only removes single isolated outliers. Clustered outliers can be removed depending on the window size, but this could also remove other objects (e.g. trees).

3.2.6 Other Related Work

Not all methods can be categorized in one class. Some studies propose methods combining different approaches. Tian et al. [2012] combines distance-based method and density-based method, and it is effective for isolated outliers and clustered outliers in airborne LiDAR point clouds. Outlier detection is based on a kernel density estimation. This is used to identify low and high points. To deal with altering terrain heights, the dataset is segmented into sections. This is done by dividing the point cloud into many blocks in the 2D horizontal plane. For every separate block the density probability distribution is estimated. Then a density threshold is defined to eliminate outliers.

Another attempt to filter both single and clustered outliers from aerial LiDAR data is done by Matkan et al. [2014]. To solve this problem, a hierarchical iterative scheme based on cross-validation technique is proposed. The height of every point is predicted based on an interpolation of its neighboring points. The predicted height is compared with the actual height and assigned with an error value. Error values above a threshold are considered outliers and removed in a iterative process, where the threshold is determined by the maximum error.

Table 3.1: Summary of proposed outlier detection methods and existing tools.

Study	*Approach*	*Neighborhood definition*	*Types of outliers*	*Topology*	*Other*
[Shen et al., 2011]	Distance	kNN	h, l, c	No	
[Sotoodeh, 2006]	Density	Radius	h, l	No	
[Meng et al., 2009]	Distribution	DT	h, l	No	
[Sotoodeh, 2007]	Cluster	DT	h, l, c	Yes	
[Arge et al., 2010]	Cluster	N/A	h, l, c	Yes	For sonar data
[Zhang et al., 2003]	Mathematical Morphology	Regular grid	l	No	
Tool					
LASTOOLS	Density	Regular grid	h, l	No	
PCL	(a) Distance; (b) Distribution	kNN	h, l	No	
TERRASCAN	Density	Radius	h, l	No	
CGAL	Distance	kNN	h, l	No	

h = high outliers
l = low outliers
c = clustered outliers

3.3 GROUP-BASED VS. POINT-BASED FILTER TECHNIQUES

Many point cloud algorithms compute features for every point. These point-wise techniques use the points within its local neighborhood, called the support region. There exist several ways to obtain a support region, such as kNN or searching for all points within a defined space [Santamaria-Navarro et al., 2014]. Generally, point-wise techniques imply a high computational load [Plaza-Leiva et al., 2017]. Therefore, group-based techniques have been studied to reduce the amount of data. One example is to divide the point cloud in regular grid cells (i.e. voxels) to speed up point cloud processing. In this case, voxels segment points prior to some point-wise feature extraction in order to avoid a costly neighborhood search. [Plaza et al., 2015] uses voxels to group a set of points that are later used to identify spatial shape features. Lehtomaki et al. [2016] performs principal components analysis on a set of points defined by voxels. Plaza-Leiva et al. [2017]; Plaza et al. [2015]; Lehtomaki et al. [2016] showed an improvement in computation load by segmenting the point cloud in voxels prior to point-wise analysis. Moreover, Plaza-Leiva et al. [2017] concludes that "voxel-based neighborhood classification greatly improves computation time with respect to point-wise neighborhood, while no relevant differences in scene classification accuracy have been appreciated."

3.4 ADDITIONAL INTENSITY INFORMATION

As discussed before, almost all existing filtering point cloud filtering methods are designed based on solely the analysis of geometrical information.

Other attributes such as intensity data is rarely used, while both geometrical and radiometric data is simultaneously captured and available for LiDAR point clouds. Wang and Glenn [2009] noted that utilizing both height and intensity may be beneficial over using either data individually to separate ground points from forested surface. Similar remarks are made by [Vosselman and Maas, 2010]. So, merging additional intensity data with geometrical information into point cloud filtering methods has the potential to improve accuracy and reliability.

3.5 CONCLUSIONS & REMARKS

Many studies recognize that detecting clustered noise is more challenging than detecting isolated outliers and requires more complex detection algorithms. Some attempts are made and discussed here. A few studies claimed to successfully detect isolated and clustered noise [Sotoodeh, 2007; Arge et al., 2010; Tian et al., 2012; Matkan et al., 2014]. However, not in every case was this extensively elaborated and tested by the authors, which makes it difficult to validate the results. Furthermore, not many scenario's in which clustered noise could appear in ALS datasets were mentioned and thus tested. Most potential is shown in the methods proposed by Sotoodeh [2007] and Arge et al. [2010]. Both approaches are cluster-based and utilize some sort of topology between vertices, where other methods only consider geometric properties of the data points and their local neighborhoods. Other additional information to improve classification may be analyzing intensity values.

Large computational cost is often a problem when analyzing 3D data. Many processing steps involve the local neighborhood for every point in the point cloud. Different studies use different approaches for constructing the local neighborhood of a data point, such as k-nearest neighbors [Sotoodeh, 2006; Nurunnabi et al., 2015], Delaunay triangulation [Sotoodeh, 2007; Shen et al., 2011; Arge et al., 2010] or regular cell division [Tian et al., 2012; Li et al., 2017a]. Studies outside outlier detection, but within point cloud processing also include voxelization [Plaza-Leiva et al., 2017; Plaza et al., 2015] or octree segmentation [Vo et al., 2015]. Especially voxelization prior to some analysis shows potential to reduce the computational cost.

Table 3.1 summarizes the general findings of the different outlier detection methods that were discussed in this chapter. The first half reviews the most prominent papers and the second half covers some available tools. NB the colors green and red do not represent a qualification. The methods are generalized in terms of approach, how the local neighborhood is constructed, which types of outliers can be successfully detected and if the method uses typological relations. Interestingly, the methods that showed most potential are both cluster-based and analyze topological relations.

4 METHODOLOGY

As described in Chapter 3, different approaches have tried to detect outliers from airborne LiDAR point clouds. Although the majority of these approaches successfully filter isolated points, previous research proves that detecting all types of outliers (i.e. clusters) remains a challenge. Chapter 3 shows that attempts are made to filter the complete area of outliers, with a few showing promising results. However, there is always a trade-off between removing outliers and removing good points. The voxel-based approach presented in this chapter aims to combine the concepts of raster processing techniques, density-based methods, PCA and LiDAR intensity within a raster data structure for efficient and coherent data analysis.

This chapter begins with an overview and motivation of the proposed method (§ 4.1). After which every step is described in detail.

4.1 OVERVIEW OF BINARY VOXEL-BASED APPROACH

This research seeks to develop a scalable voxel-based outlier detection method for aerial LiDAR point clouds. This section will be a step-by-step explanation and justification of the proposed method. Different types of outliers can be found in aerial point clouds, as discussed in § 2.1. For the design of the proposed method I have grouped the different types of outliers as follows:

TYPE-1 – Points that appear at random above and below ground surface (i.e. high and low outliers); usually these are isolated points, but they can also appear in small clusters.

TYPE-2 – Cluster outliers—points always found in clusters and often along the direction of the scan line.

TYPE-3 – Points randomly scattered through the point cloud with different densities, in all probability due to error in the laser scanner.

4.1.1 Method Motivation: Detecting Outliers with Connectivity

Previous studies repeatedly mention two main complications in detecting all types of outliers. The first is the challenge to detect clustered outliers. Existing algorithms can successfully handle the first type of outliers from above (if they come as isolated points), since they deviate from its local neighbors quite significantly. For outliers from the second and third type, effectiveness of existing methods usually depends on the size of the local neighborhood they consider. Typically, they perform well if the neighborhood is large enough to include a substantial amount of good data points to catch the deviating trend of clustered outliers. However, these algorithms hinge on the selection of the neighborhood size and fail when clustered outliers are so large that the algorithm is unable to decide which points belong

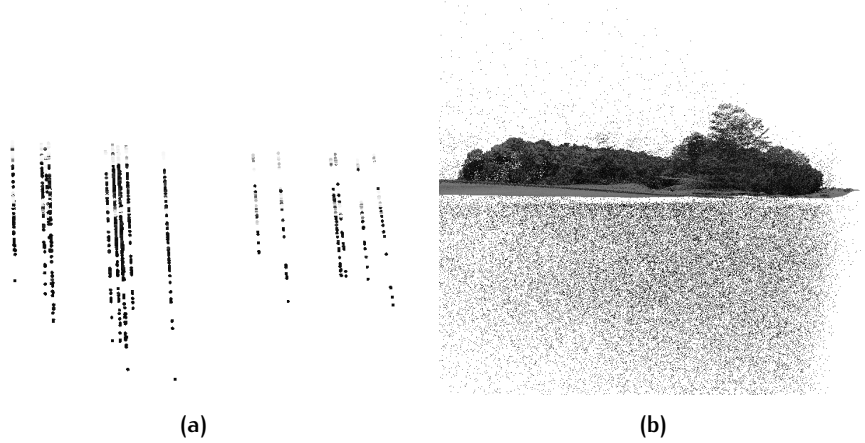


Figure 4.1: Examples of different types of outliers. (a) Clustered outliers along the direction of the scan line (type-2. (b) Scattered outliers (type-3).

to the surface and which points do not. A logical improvement may seem to be increasing the neighborhood size, but since the terrain can be very complex with many objects and varying terrain heights, local outliers are impossible to detect. For these reasons, Arge et al. [2010] proposes a new algorithm that is graph-based instead of neighborhood-based.

4.1.2 Method Motivation: Minimize False Positives

The second complication is that some points in the data set are of importance for the end user, but hard to distinguish from outliers (i.e. false positives are hard to separate from true positives). This seems unavoidable but can usually be overcome by adjusting the parameters of the method. However, there remains a trade-off between true positives and false positives, i.e. threshold fine-tuning can increase a higher accuracy of true positives (correctly classified), but come at the cost of more false positives (wrongly classified). [Matkan et al., 2014; Arge et al., 2010; Sotoodeh, 2007]. This calls for more analysis of the dataset to make better judgment between the two. Therefore, I propose a combination of methods in one single workflow to detect all types of outliers and minimize false positives.

Inspired by this study by the work of Arge et al. [2010], and the potential of using multiple processing techniques for better classification, I designed the workflow shown in Fig. 4.2.

4.1.3 A Voxel-Based Solution

A point cloud is modeled as a grid where each voxel is classified as outlier or non-outlier. The 3D grid data structure created after voxelization of the source point cloud lies at the core of the method. It serves as an efficient and organized data model when finding neighboring cells is as easy as searching an index. This is convenient when analyzing topological relations, which is essential for detecting clustered outliers as concluded in § 3.5. This voxel-based solution reduces data processing in comparison with point-based approaches.

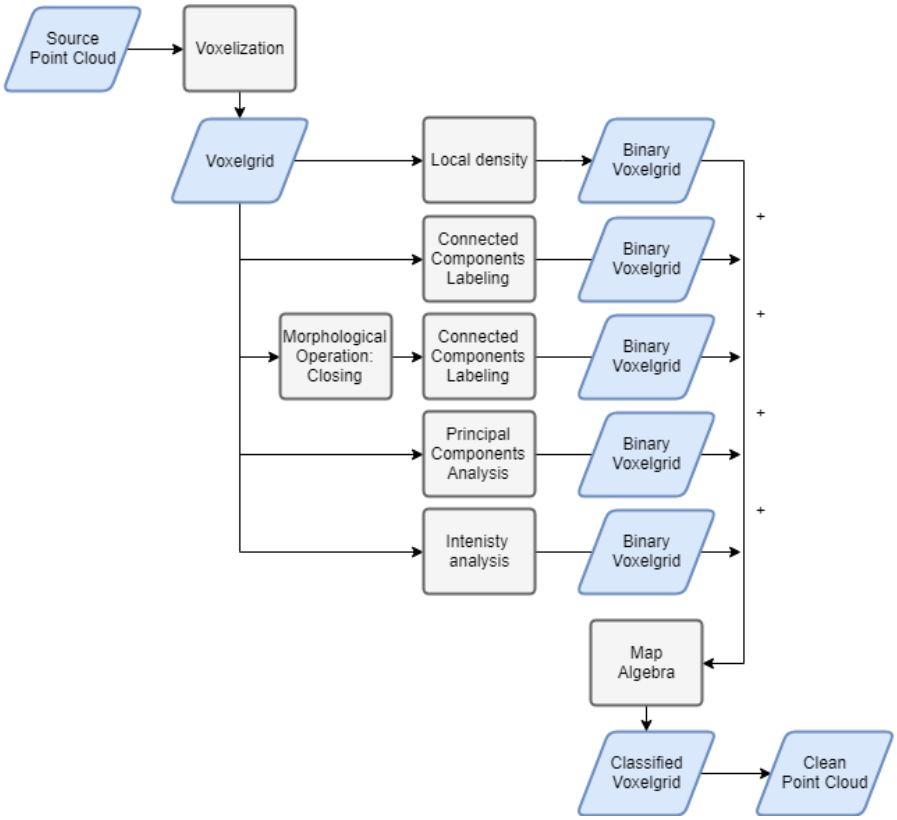


Figure 4.2: The source point cloud is voxelized. This voxelgrid is input for five analyses. These results together determine if a voxel contains outliers.

4.1.4 5 Analysis Techniques

To find all types of outliers and minimize false positives, different analysis techniques need to be performed on the source point cloud. Every analysis step can be seen as an individual operation which results in a boolean classification for every voxel stating it contains outliers or not. This means that for every analysis, all voxels with points inside will be flagged with a *true* or *false* boolean classification. *Note that a voxel can be classified as outlier by one operations and not by another.*

In this thesis I propose five different analyses to detect outliers in a binary grid structure:

1. Local density (§ 4.3);
2. Connected components labeling (§ 4.4);
3. Connected components labeling after a morphological closing operation (§ 4.4.1);
4. LiDAR intensity (§ 4.5);
5. Planarity—principal components analysis (§ 4.6).

This selection is based on two factors: The first selection of methods is based on the potential found in previous studies and need to be further explored. These include 1, 2 and 3 from above—these methods will be researched more intensively (2 and 3) in this thesis than the second selection and are purely focused on outlier detection. More specific, they are designed to *maximize* detection of all types of outliers (True Positives (TP)).

The second selection is more experimental. Meaning, these methods add extra knowledge about each voxel but are not specifically designed for outlier detection (4 and 5 from above), i.e. they are designed to *minimize* falsely classified outliers (FP). One could easily choose to add, remove or even change these methods and still use the same proposed workflow. For example, not all point clouds have intensity attributes available, so intensity analysis could easily be removed from the process. All these steps are conceptualized in Fig. 4.3.

Together, the methods provide sufficient information to decide if a voxel contains outliers or not. This is decided with simple map algebra, introduced by Tomlin [1983]. All binary classifications (i.e. 0 for non-outlier and or 1 for outlier) for each voxel are added together. This results in a voxelmap where each voxel is assigned with an integer value ranging from 0 to 5. Voxels with ≥ 3 are classified as outlier.

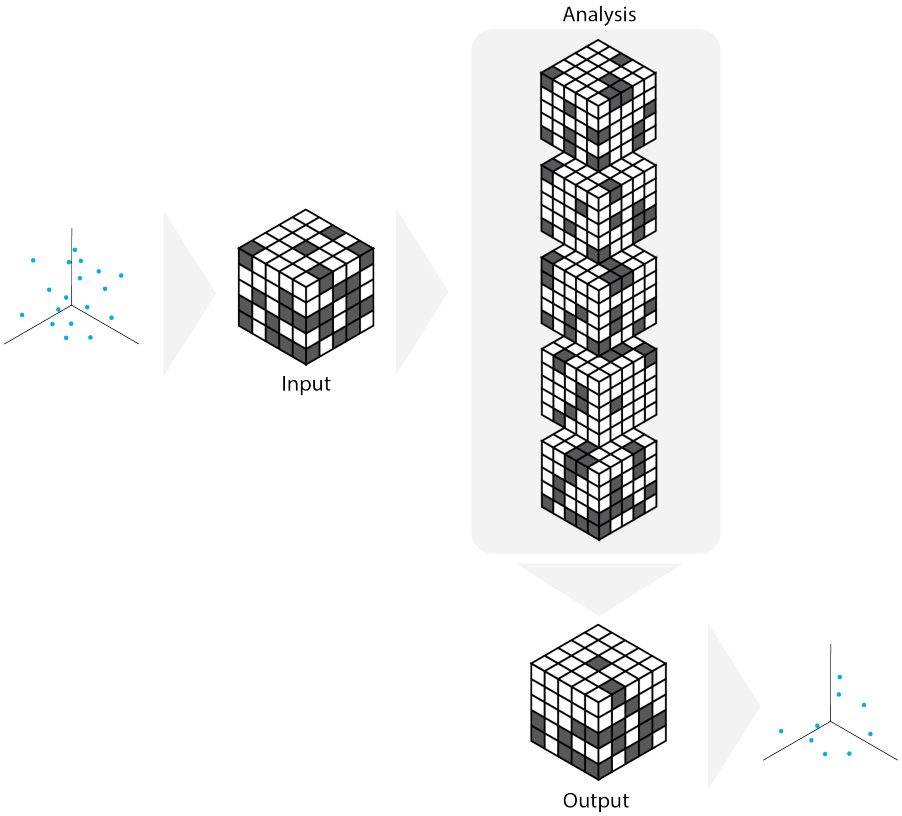


Figure 4.3: Input: voxelized point cloud. Analysis: individual operations to classify outliers. Output: add the result of all operations together.

4.2 VOXELIZATION

The first step involves creating a 3D regular grid. The process to build this structure is based on an input point cloud in Cartesian coordinates. In a point cloud, (x, y, z)-coordinates are stored explicitly. Additionally, topology should be stored explicitly as well. In a 3D raster, voxel locations are defined implicitly by the position in the grid. However, a grid also create voxels in locations where no data points are located. Therefore, one can decide against storing voxels without points inside to save memory space (see § 2.3.2 to read about the difference between a dense and sparse array). However, raster processing techniques and neighborhood operators benefit from the raster structure and require every voxel to be stored.

The voxel value will be limited to 0 for empty voxels and 1 for voxels with laser points inside, as explained in § 2.3.2. Crucial in this conversion is *spatial resolution*, i.e. the size of one voxel. This also determines the resolution of the 3D raster used for raster processing and how many voxels are created. The choice of a suitable spatial resolution depends on three requirements:

1. The density of the point cloud—points of real life features should form a connected set of voxels. When the point density of a point cloud is low and spatial resolution is chosen too fine, gaps between voxels appear which may cause problems during analysis (further explained in § 4.4). On the other hand, when voxel size is chosen too large, many points get converted into the same voxel at the cost of a lower level of detail.

2. Spatial resolution determines how many and which points each voxel contains after rasterization. When voxel size is chosen too large, it may happen that outlying points are part of the same voxel as inlying points, such that outliers can not be separated from good points (further explained in § 4.4).

3. Processing time should be kept to reasonable amounts. Usually processing time increases with the 3rd power of the resolution as well.

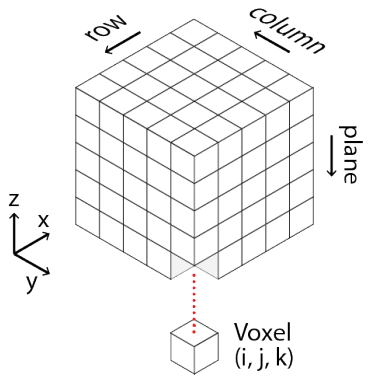


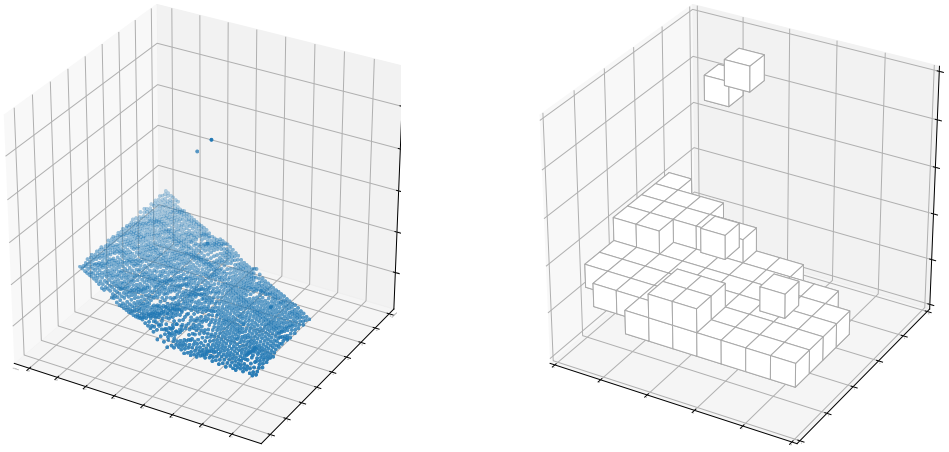
Figure 4.4: Voxel space with (i, j, k) coordinate system

Nourian et al. [2016] describes a voxelization algorithm for vector inputs (i.e. point cloud data). This algorithm creates a 3D bounding box from a input point cloud. Depending on the voxel size, it then finds how many voxels fit in each direction. For every data point, it is determined in which voxel it is located. The location of a voxel is defined by a row, column and plane integer number (see Fig. 4.4). Logically, the number of planes, rows and columns are determined by the voxel size and the minimum and maximum (x, y, z)-point coordinates of the source point cloud. Only voxels with points inside get a value 1, empty voxels 0, creating a boolean voxelgrid (1 bit values are minimal in size). This boolean voxelgrid is used for defining local point neighborhoods and raster processing techniques in further processing steps. An example of this voxelization can be seen in Fig. 4.5a, showing the source point cloud, and Fig. 4.5b showing its voxelgrid. In this figure, only voxels with points inside are shown; empty voxels are left out for clarity but are part of the boolean grid.

4.3 LOCAL DENSITY

Isolated outliers (type 1) can be successfully detected and removed by existing algorithms. Many are discussed in Chapter 3—especially distance- and density-based approaches seem effective. The density-based method, as described in § 2.2, is used in this study to filter isolated outliers. The motivation for using this approach is twofold: (1) ALS datasets usually do not suffer from varying local point densities from one place to another, because the surface is captured from the same height. (2) The voxelgrid provides a data structure to define local neighborhoods and determine its density in an efficient way.

Its implementation is uncomplicated and efficient when a grid is used. Usually, finding the nearest neighbors for every data point consumes the



(a) (b)

Figure 4.5: (a) source point coordinates. (b) voxelization of source point cloud.

most computation time. However, in a grid the nearest neighbors are located in neighboring voxels and can thus be easily accessed. See Algorithm 4.1, the algorithm starts with iterating over every data point and finding the voxel its located in. Then, for every voxel its local neighborhood is constructed by its 26-neighborhood. Every data point in these adjacent voxels are counted. When this count is lower than a threshold, the data point is considered an outlier.

Algorithm 4.1: DENSITY ($\mathcal{V}$, $\mathcal{D}$, τ)

Input: A VoxelGrid $\mathcal{V}$, dictionary $\mathcal{D}(v, p)$ key: voxel location v; value: points list p, and density threshold τ
Output: V_r: Boolean grid

```
1  V_r ← new bit array of size VoxelGrid ;
2  for voxel v in V do
3      sumpoints ← 0;
4      n ← get 26-neighborhood of voxel;
5      for v in n do
6          c ← count points in D[v];
7          sumpoints ← sumpoints + c;
8      if sumpoints < τ then
9          V_r[v] ← true
10 return V_r
```

4.4 CONNECTED COMPONENTS LABELING

Consider the binary grid shown in Fig. 4.7a, where outliers (type 1, p. 29) above the surface appear. It is easy to see that these voxels are not in connection with the surface voxels. For automatic separation of voxel groups

on a large scale, existing raster computations can be performed (§ 2.4). The binary grid created after voxelization of the source point cloud is used to define which voxels are connected in a computationally efficient way. This is achieved with connected components labeling (discussed in § 2.4.2). With CCL, regions of voxels with points inside are labeled with a unique integer, i.e. unconnected regions get different integer labels (see Fig. 4.7b). As discussed in § 2.4.1, in a 3D grid spatially adjacent voxels are 6-, 18-, or 26-connected. Logically, when only 6-connected voxels are allowed, CCL separates all voxels without a common face, including diagonal features. Obviously, diagonal features exist in real-life data sets where objects are more complex are not only horizontally and vertically oriented. Therefore, I search in the 26-neighborhood of every voxel with points inside for connected objects. When all voxels are marked with an integer of its according region, outliers can be detected in two steps:

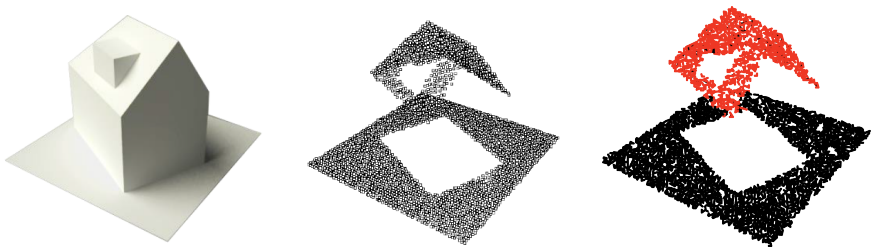
1. Count all region sizes to find the largest connected region;
2. Classify all smaller regions as outliers.

I assume here that the largest region of voxels will always be the ground surface and thus not outliers. Subsequently, a cluster of outliers is also detected, because its size is smaller than the largest region. Moreover, the size of the cluster does not matter, which makes this algorithm effective to detect all three types of outliers.

As one might expect, voxel size greatly influences the detection of outliers. A smaller voxel size translates real life objects with more detail into the raster domain, but it also determines the distance threshold for detecting outliers. In a 26-neighborhood, unconnected voxels have a minimum distance between them of one voxel edge length. Therefore, voxel size is the most important parameter for outlier detection. A smaller size may detect more outliers, but can also cause real objects to get disconnected when features have a low density of points, e.g. treetops with leaves and small branches. Every object that gets disconnected from the largest region is classified as an outlier and could decrease accuracy by an increased false positives rate. In the next section I propose a preprocessing step based on mathematical morphology to overcome this problem

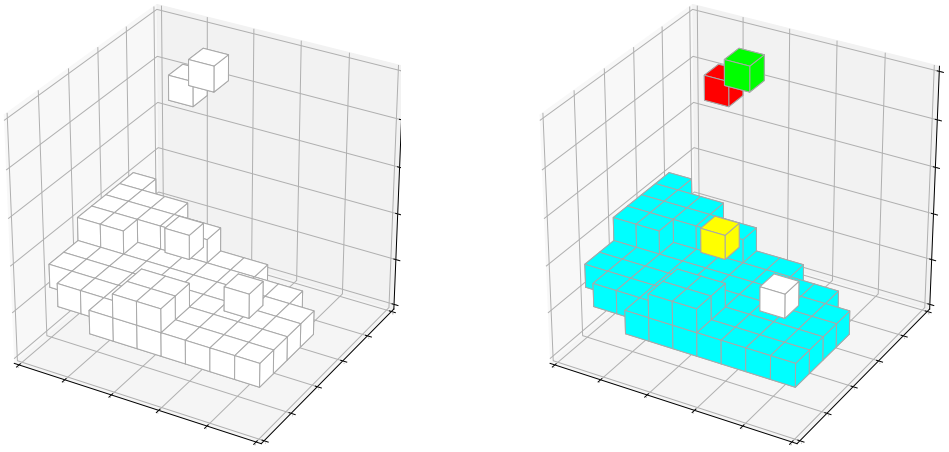
4.4.1 Morphological Transformation: Closing

The ideal surface of a 3D point cloud is fully sampled and forms one cluster of good points. However in practice, real life ALS datasets have various kinds



(a) (b) (c)

Figure 4.6: (a) Example house. (b) Point cloud of (a). (c) Roof (red) and ground (black) are disconnected due to occlusion.



(a) (b)

Figure 4.7: (a) Binary voxelgrid. (b) Labeled regions after connected components labeling.

of complex objects including concave geometry that may cause occlusion. For example the surface below a roof may be occluded, see Fig. 4.6. This could disconnect the roof from the ground surface, however in practice this occurs very rarely, because ALS often captures objects at an angle. Also influenced by other factors, such as a too low point density for small objects when scanned from high. Think of a tree with leaves and branches that are too small, such that the laser pulses only hit the tree surface at certain locations or branches occluding other branches. This causes discontinuity in the point cloud and thus disconnected and floating clusters with different sizes, i.e. a tree branch not connected to its trunk. Since some clusters are still part of the surface model, they can be connected to each other. As explained in § 2.4.3, mathematical morphology can change the structure or form of a voxel region. *Closing* (dilation followed by erosion, see § 2.4.3) is used to fill gaps between voxels (and thus laser points). The maximum size of the gaps that can be filled this way is controlled by the structuring element.

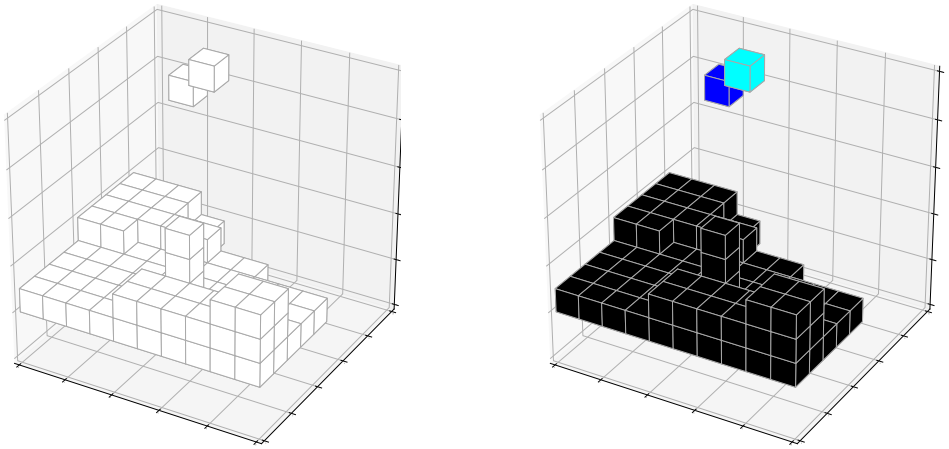
Structuring Elements

The structuring element controls filling of gaps in two ways:

1. By using different *sizes* of structuring elements, it is possible to control the size of the gaps that are repaired. Logically, a larger structuring element closes larger gaps.

2. By using different *shapes* of structuring elements, it is possible to transform a region in a preferred direction, e.g. connecting points on the same height instead of connecting higher and lower points can be achieved with a horizontally oriented structuring element.

It is important to understand the desired goal of the morphological operation in order to adjust these parameters and enhance the dataset optimally. In my case, I want to repair objects with gaps due to low point density or occluded features. As mentioned before, in real world datasets objects are



(a) (b)

Figure 4.8: (a) Binary voxelgrid after closing operation. (b) Labeled regions after CCL of (a).

oriented in any direction. Therefore, a 3D symmetrical structuring element is favorable for connecting features in any direction. Observe Fig. 4.8a, here gaps between voxels are filled with a $3 \times 3 \times 3$ structuring element. Fig. 4.8b shows its labeled version after CCL. Notice the difference with Fig. 4.7b—five regions are found in the original voxelset; three regions are found after the closing operation.

4.5 LIDAR POINT INTENSITY

Intensity values from each return are captured in most LiDAR datasets, but rarely used for point cloud processing. As mentioned in § 3.4, merging this information with geometrical data has the potential to increase accuracy.

Experiments with many datasets showed that outliers often have low intensity values compared to the complete range. Fig. 4.9 clearly shows low intensity values for outliers in a example dataset. The returned intensity

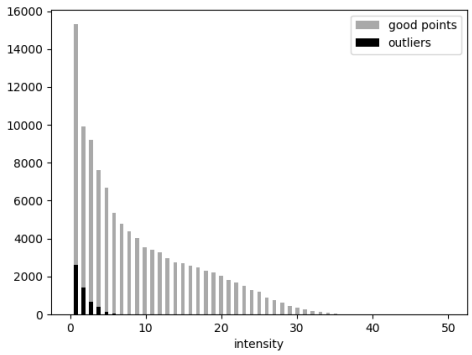


Figure 4.9: Intensity histogram of example point cloud.

is effected by atmospheric conditions and laser sensor type. Therefore, the range of values vary between different datasets and can not be used for general classification, without radiometric and atmospheric corrections. These corrections are not in the scope of this study, limiting intensity analysis to each dataset separately.

4.5.1 Data Statistics

Based on results from experiments, voxels with low intensity values are classified as outliers. For each voxel, the average intensity of all its containing points is compared to a threshold. This threshold is simply computed by a measure used in statistics indicating the value below which a given percentage of data falls, called percentile. Percentiles represent the area under the normal curve, so each standard deviation relates to a percentile. Due to a heavily right-skewed distribution (refers to Fig. 4.9), data right from -1σ is not considered an outlier. This is equal to the 15.87th percentile. Obviously, also good point can have intensity values below -1σ, due to low reflectance surfaces. Therefore, this analysis is not to detect outliers, but to identify possible good points in order to minimize false positives.

4.6 PLANARITY

The identified types of outliers usually appear random in the point cloud, i.e. outliers form a scattered region and rarely fit in one plane. As explained in § 2.5.1, PCA is widely used to describe a group of points as planar, tubular or scatter regions. Each class is defined by an equation with eigenvalues and eigenvectors from the covariance matrix of 3D points. In the approach described here, a point cloud is modeled as a voxel map where each voxel is classified as a flat surface, or a scatter shape based on the 3D point inside each voxel. Scattered regions are considered for outlier detection. Note this method by itself would never be sufficient to detect outliers, instead it provides useful information for later outlier classification.

The 3 by 3 symmetric covariance matrix for each voxel is determined by a set of n inner $\mathbb{R}^3$ coordinates, as described in § 2.5.1. A minimal of three points is needed to describe a plane, and only voxels with at least four points are considered—voxels with less are not sufficient for PCA and treated as outlier. Remember, for planar voxels $\lambda_0 \simeq \lambda_1 \gg \lambda_2$ (see Fig. 2.8b, p. 20). When variation is similar in all directions, such that $\lambda_0 \simeq \lambda_1 \simeq \lambda_2$, its region is scattered (see Fig. 2.8a, p. 20). In practice, the planarity of each voxel is defined by a curvature threshold θ. Consider σ as an indicator of surface curvature: $\sigma = \dfrac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}$. If σ is higher than 0.1 (given by experience) it indicates a scattered region. These voxels are considered outliers and classified as such.

5 IMPLEMENTATION & RESULTS

This chapter validates the effectiveness of the proposed voxel-based methodology introduced in the previous chapter. This is done by running a number of experiments on different datasets. § 5.1 starts with implementation details of the proposed methodology. § 5.3 gives the quality metrics that are used to assess the results from the experiments, § 5.2 describes the used datasets, and results are given in § 5.4. Finally, the method is compared with existing outlier detection tools and algorithms in § 5.6.

5.1 STRUCTURE OF DEVELOPED PROTOTYPE

To test the proposed method from Chapter 4, a translation to a software prototype is necessary. This developed prototype should run every experiment on any ALS dataset and analyze the results. This is possible with Julia (used by Deltares), which already has many available packages for point cloud processing. Everything from this implementation is freely available, including the algorithms I wrote (see the GitHub[1] repository).

5.1.1 Data Structures

This section describes the data structures that is employed in the proposed outlier detection model. The first step involves building data structures that are used for voxel classification. At the core of the method lies a voxel grid data structure. The voxelization structure is initially created from a `.las` or `.laz`[2] file as introduced in § 4.2. This process creates two data structures (see Fig. 5.1):

1. **3-dimensional binary array**. This data structure represents voxels in a 3D spatial grid with points inside. It is used as input for raster processing techniques.

2. **Sparse voxel dictionary**. This data structure describes which data points each voxel contains. In this dictionary the `key` contains a tuple of three voxel coordinates (i, j, k), the `value` refers to a list of point integers. This data structure is necessary to keep track of information of all the points from the source point cloud which is redundant with respect to the bit array. Furthermore, the sparse structure speeds up outlier detection by only iterating over voxels with points.

These two data structures contain the necessary information for the algorithms.

1 https://github.com/SimonGriffioen/Voxelized-point-cloud-filtering
2 A `.las` file is an industry-standard binary format for storing airborne lidar data—`.laz` is a compressed version.

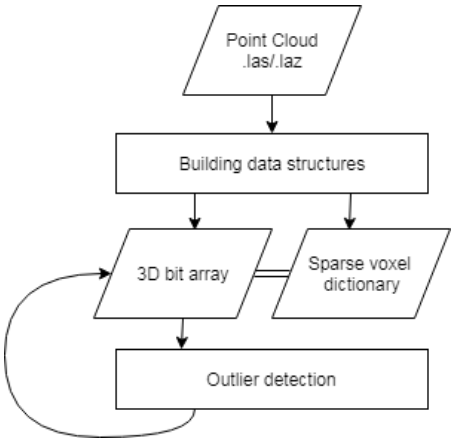


Figure 5.1: Data structures.

5.1.2 Implementation of Algorithms

All algorithms are implemented in Julia. This coding language knows many Open Source packages that can be used, including algorithms for point cloud- and image processing. To clarify how I developed this prototype, I have listed all used packages and algorithms implemented by myself. Important packages I used are:

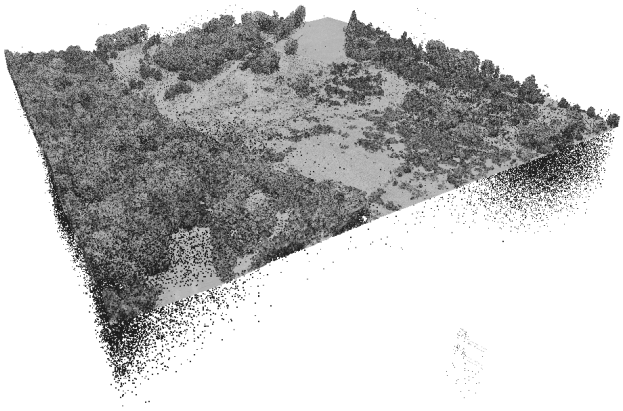
- `LasIO.jl`—a Julia package for reading and writing `.las` and `.laz` files.
- `LocalFilters.jl`—this Julia package implements multi-dimensional local filters (e.g. mathematical morphology). Specifically, it is used for the *closing* operation (§ 4.4.1).
- `Images.jl`—an image processing package for Julia. This is used for Connected Components Labeling (§ 4.4).

Algorithms implemented by myself are available on GitHub and include:

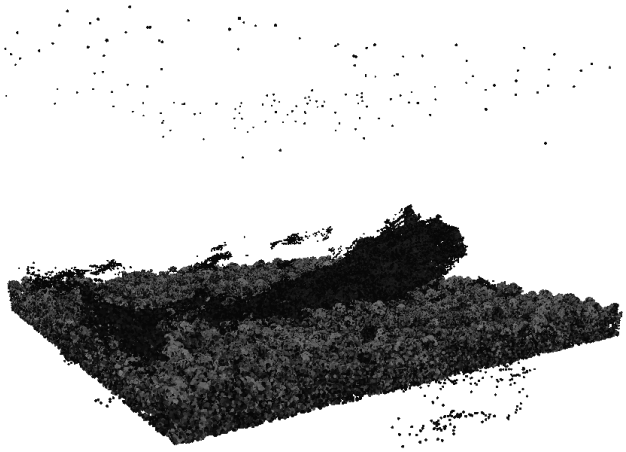
- `Bounding Box`—this is to find the minimum bounding box of a point cloud (§ 4.2).
- `Voxelization`—an algorithm to rasterize a source point cloud and create the two data structures (§ 4.2).
- `Count labels`—for finding the largest group of connected voxels (§ 4.4).
- `Local Density`[3]—for the detection of isolated outliers as explained in § 4.3.
- `Intensity analysis`—for outlier classification by point intensity as explained in § 4.5.
- `Planarity`—this algorithm classifies each voxel as plane (non-outlier) or scattered (outlier) as explained in § 4.6.
- `Boolean outlier detection`—this last algorithm is used to make the final classification by combining all results (§ 4.1).

3 This is almost an identical reimplementation with Julia of lasnoise from LAStools (coded in C++).

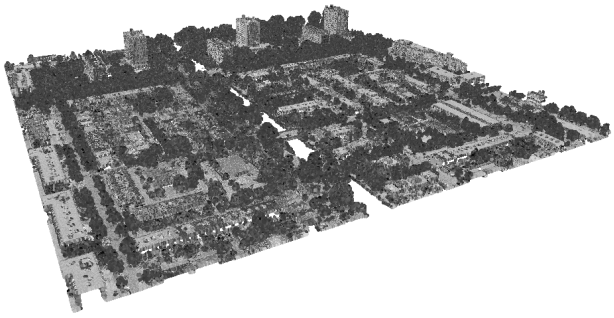
(a) A1.



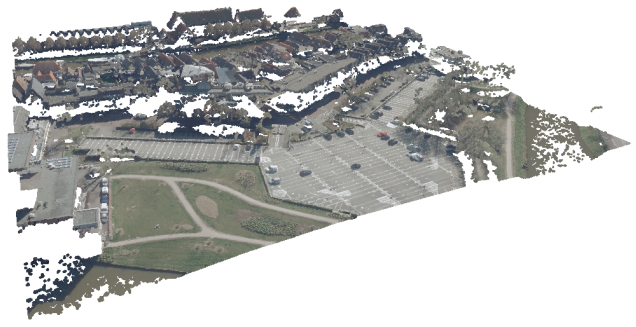
(b) A2.



(c) B.



(d) C.



(e) D.

Figure 5.2: Overview of source point clouds used for experiments.

Table 5.1: Overview of the datasets used for experiments

	Point cloud				
	A1	A2	B	C	D
Source	Aerodata	Aerodata	Deltares	AHN3	Kadaster
Technique	ALS	ALS	ALS	ALS	DIM
Area (km)	0.5 x 0.5	0.5 x 0.5	0.5 x 0.5	0.5 x 0.5	0,5 x 0,5
N points	5.7 mln	8.2 mln	1.7 mln	4.7 mln	5.5 mln
Points per m^2	23	33	7	19	22
Outliers	Many	Many	Many	None	None
Ground truth	Yes	Yes	No	No	No
Environment	Vegetation, built environment	Vegetation, built environment	Forest	Urban	Urban

5.2 DATASETS

The algorithm presented in Chapter 4 is tested on several different point clouds (see an overview in Fig. 5.2) to obtain a reliable result and to review how it handles various situations. Point clouds from four different sources are used to evaluate the proposed method, provided by Aerodata (Fig. 5.6a and Fig. 5.7a), Deltares (Fig. 5.9a), AHN (Fig. 5.2d) and Kadaster (Fig. 5.10a). All point clouds are acquired from ALS, except from Kadaster—this point cloud is produced with DIM[4] by photogrammetry software. Table 5.1 gives an overview of the different datasets that are used to evaluate the proposed method. This variation of point clouds is selected to explore the performance from two angles, including: (1) changing environments, such as natural-, forest-, and urban environments; (2) different acquiring techniques (ALS and DIM). The selection of datasets contain moderate to extreme amounts of noise, as well as different features.

Datasets B, C and D do not have ground truth available. Therefore, these point clouds are used to evaluate the False Positive Rate (FPR), which is almost equally important as the False Negative Rate (FNR). It should be noted that the ground truth that is available for dataset $A_{1,2}$ is not the absolute state of being true. The reference datasets from Aerodata are classified with automated methods and manually checked. It is not always clear for an operator which points should be classified as outliers, so well-cleaned datasets may still have outliers or falsely classified outliers. This should be considered when the accuracy is determined.

Computation time is also analyzed, therefore different sized point clouds are selected. To evaluate the effect of increasing the point cloud size, only computation time and memory allocations are considered—accuracy is not assessed in this case.

5.3 QUALITY METRICS

The simplest way to assess the results of outlier detection is by visual inspection. Comparing the source dataset with the cleaned version shows roughly if false positives and false negatives are made. But when large datasets are classified and reliable assessment is needed, quantitative evaluation is essen-

4 Dense image matching aims at computing a depth value for each pixel of multiple overlapping aerial images. This is a suitable alternative to airborne LiDAR for generating 3D point clouds.

		True Condition		
	total	**Positive**	**Negative**	
Predicted Condition	**Positive**	TP	acfp	Sensitivity
	Negative	FN	True Negatives (TN)	Precision
		FNR	FPR	

Table 5.2: Confusion matrix.

tial. There are many metrics that are used to measure the performance of a classifier. The apparatus depends on the measuring scale (nominal, ordinal, interval or ratio)[Lemmens, 2011]. When classes are attributed to objects it is a nominal scale.

In the case of outlier detection, only two classes exist. The evaluation of these binary classifiers is often done by constructing an error matrix, also called confusion matrix or contingency table (see Table 5.2). Given a point cloud, a classification gives a number of TPs (points correctly classified as outlier), TNs (points correctly classified as non-outlier), FPs (points falsely classified as outlier) and FNs (points falsely classified as non-outlier). These are obtained by comparing the class of each point with the ground truth and can be formulated in a 2×2 confusion matrix. From this matrix several statistics obtained from Fawcett [2006] are computed. The *sensitivity* is computed with Equation 5.1, and gives the percentages of outliers that are detected.

$$Sensitivity = \frac{TP}{TP + FN} \tag{5.1}$$

Theoretically, one can classify all point (including good ones) as outliers which results in 100% sensitivity. Obviously, this would not be a satisfactory result. Therefore, more measures are necessary to assess the quality of classification. *Precision* gives the percentage of detected outliers that correspond with the ground truth and is computed with:

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

Furthermore, the *False Positive Rate* (FPR) is used to determine the percentage of points that are classified as outliers, but are actually good points and in this thesis referred to as FPR (Type-I error):

$$False\ Positive\ Rate\ (FPR) = \frac{FP}{TN + FP} \tag{5.3}$$

Finally, the *False Negative Rate* (FNR) determines the percentage of reference outliers that are not detected—also computed with $1 - sensitivity$ and referred to as FNR (Type-II error):

$$False\ Negative\ Rate\ (FNR) = \frac{FN}{TP + FN} \tag{5.4}$$

A good performing classifier has a low FPR and a low FNR. The influence of voxel size is assessed by computing the FPR and FNR for varying voxel sizes.

5.4 RESULTS

This section describes the performance of the cleaning algorithm in terms of outlier detection quality. This is done by evaluating all datasets, introduced in the previous section, against the presented Quality Metrics. The effect of the voxel size on the accuracy is discussed in § 5.4.1 followed by common errors (§ 5.4.3). Subsequently, different parts of the algorithm are discussed separately in § 5.4.4. Finally, § 5.5 evaluates the computation time of the developed prototype.

5.4.1 Outlier detection: overall algorithm performance

First, the overall performance of the proposed algorithm is discussed in terms of the noise types identified in § 4.1. Observe Fig. 5.6 to visually inspect the cleaning of dataset A1. Type-1 noise is removed effectively as is seen in Fig. 5.3. These high and low isolated outliers (type-1) provide a useful example to get intuition about how and why the algorithm works. Consider the voxelization in 5.3b, where connected components are labeled with colors. Its easy to see that outliers (colored voxels) are not connected in the voxel grid with the terrain (black voxels). This is a strong indication for outliers, moreover the same points do not form a plane or have high intensity values. Fig. 5.3c shows the detected outliers and are removed from the point cloud in Fig. 5.3d. In this case, detecting outliers is straightforward when (almost) all operations classify it as such.

For the same reasons this algorithm also handles cluster noise (type-2) well. Different from type-1 noise is that these outliers have a higher local density. As long as these clusters are far enough from the surface, they are classified as outliers.

Type-3 outliers are found to be the most difficult to detect, because they often appear close to the terrain and have high local densities. However, the algorithm still removes most of this noise very well. This is clearly shown by observing the cleaning of dataset A1 in Fig. 5.6.

Dealing with Challenging Situations

The above situations of how this algorithm handles outliers are the most straightforward situations. However, in many cases it is harder to separate outliers from good points. When outliers (clusters or isolated) move closer to the surface, as such that they are connected to the terrain, they are not separable from real objects (e.g. vegetation). Similarly, real objects scanned further from the terrain can be unconnected and possibly seen as outlier. An example is shown in Fig. 5.4a, where tree tops are disconnected from the terrain. Only connected components labeling would classify these tree tops as outliers. However, a closing operation can connect tree voxels with terrain voxels, local point density is not low and LiDAR intensities do not match with outliers, allowing the algorithm to keep most of the points and only falsely classify a minimum of good points as outliers. By maintaining the unconnected trees after cleaning, clearly shows the advantage of using multiple methods.

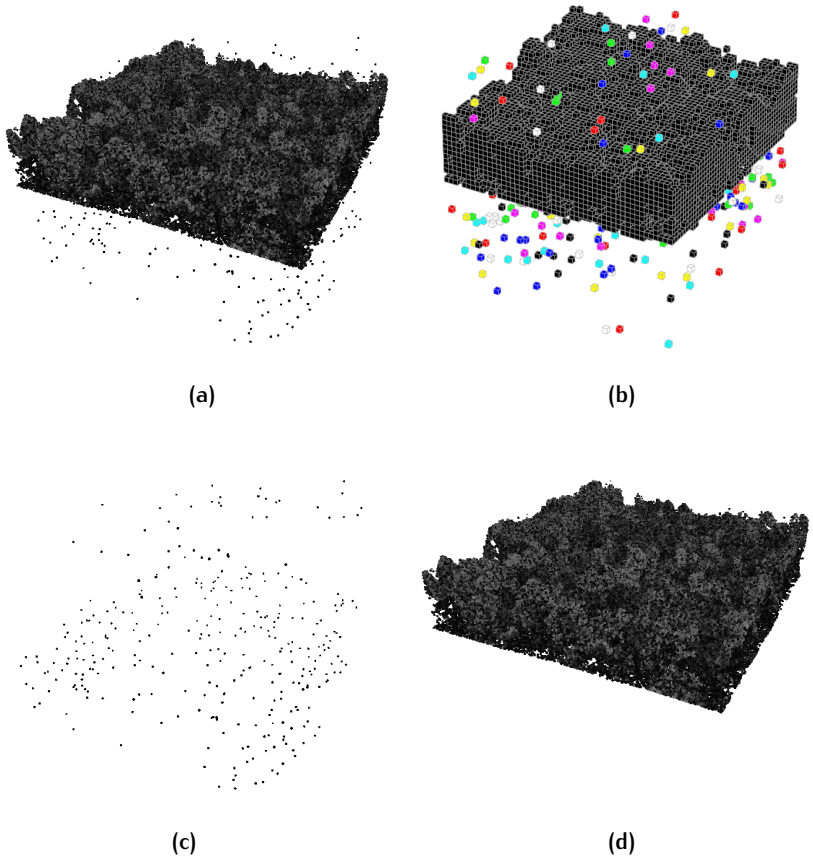


Figure 5.3: Section from point cloud A1. (a) Point cloud with type-1 outliers. (b) Connected components labeling of (a)—every color represents a component (black is the terrain). (c) Outliers that are detected. (d) Cleaned point cloud.

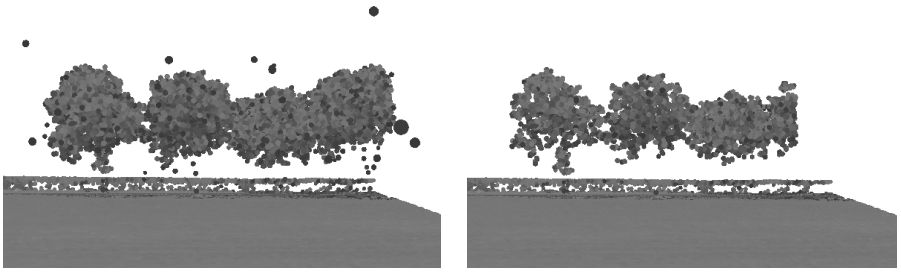


Figure 5.4: (a) Low local point density between treetops and ground (A1). (b) Cleaned version of (a)—all outliers and a few tree points are removed, because they are not connected to the terrain. The use of multiple operations allows the majority of tree points to not be falsely classified.

The proposed method also handles common challenging situations intuitively by design. By constructing a voxel grid and defining connected parts it deals with deviating terrain heights without the need to set any local thresholds. A steep slope in the scene does not affect outlier detection, as long as the terrain is connected. Handling complex terrain without any problems is a huge advantage compared to neighborhood-based methods which use local height thresholds. Similarly, features scanned below the ground surface (e.g. a ramp to a subfloor artefact) causes no problems, whereas existing methods could classify points below ground as outliers.

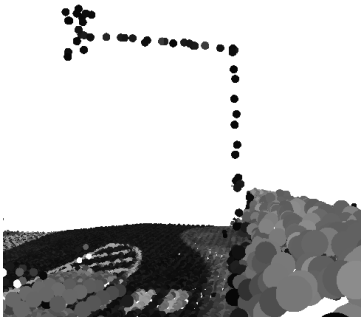
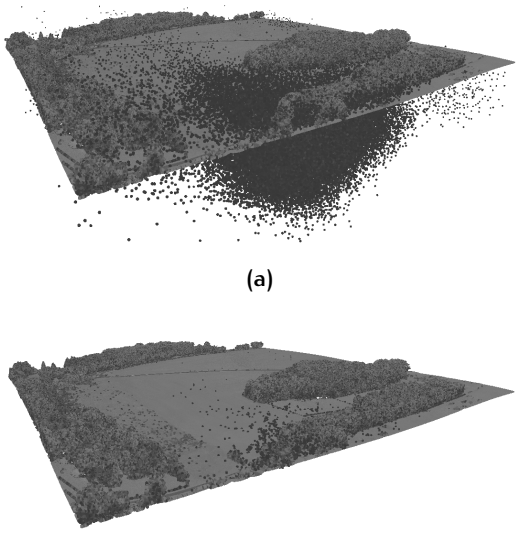


Figure 5.5: Low density object not classified as outliers because its is connected to the surface.

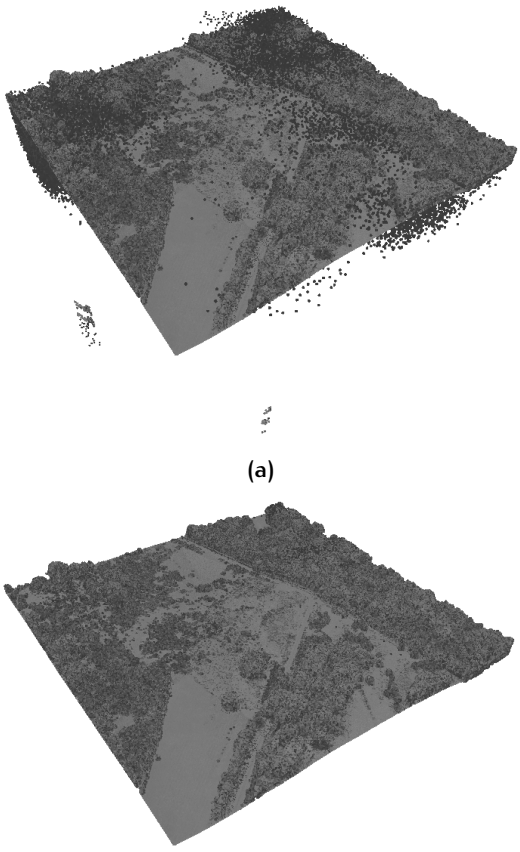
Another complex situation that is handled well by this method is illustrated in Fig. 5.5. Street posts like this are difficult to separate from outliers for two reasons. First, the point density is very low due to a small object surface (density-based methods). Second, together with points scanned significantly higher than its local neighborhood may cause problems for distance-based and morphological methods. However, points on the street post will be part of the same connected part as the surface, because it physically connects to the ground. This enables the proposed method to distinguish real objects from outliers.



(a)

(b)

Figure 5.6: Point cloud A1. (a) Raw. (b) After outlier removal.



(a)

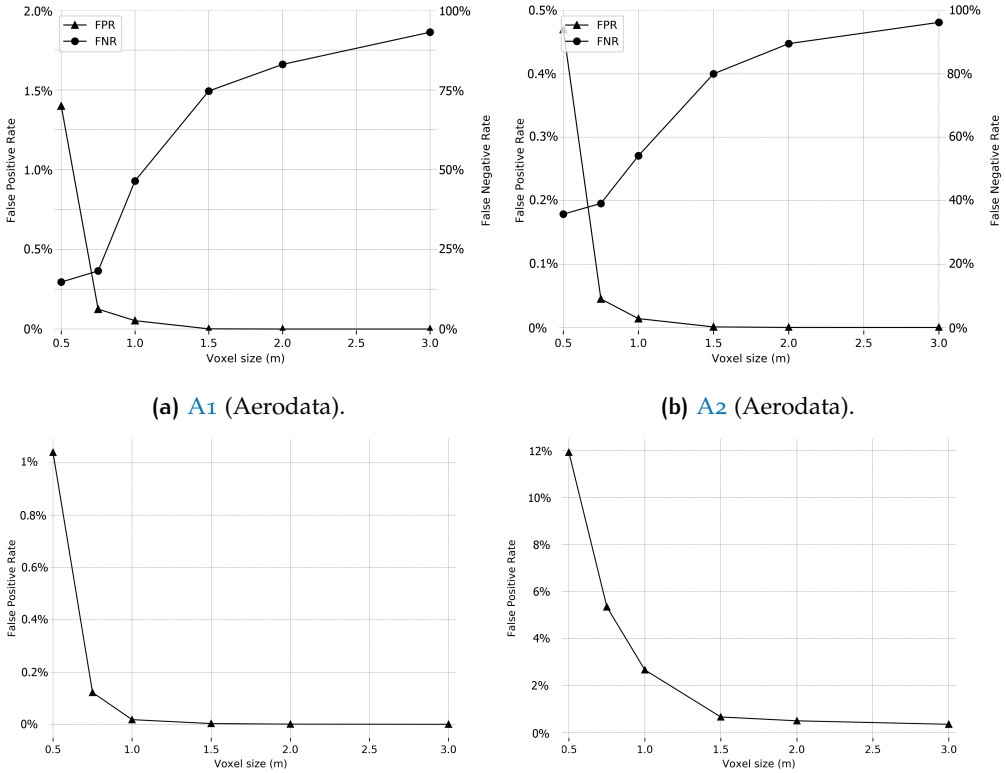
(b)

Figure 5.7: Point cloud A2. (a) Raw. (b) After outlier removal.

5.4.2 Voxel Size Selection and Performance

Now that we have a better understanding and feeling of how and when the algorithm works, we can take a more detailed look into the performance. The voxel size selection obviously has a major influence on the performance in both accuracy and computation time (computation time is discussed in § 5.5).

By studying datasets A1 and A2 the sensitivity of the classifier can be determined for different voxel sizes. For both examples shown in Fig. 5.6 and Fig. 5.7 a voxel size of 0.75 m is used. The performance per voxel size is illustrated in Fig. 5.8. Logically, a smaller voxel size results in a higher sensitivity. However, with a too fine resolution (small voxel size) gaps will appear which causes more features to be unconnected, and thus a high FPR. This trade-off between a low FPR or a low FNR is well illustrated in both Fig. 5.8a and Fig. 5.8b.



(a) A1 (Aerodata). **(b)** A2 (Aerodata).

(c) C (AHN). **(d)** D (Kadaster).

Figure 5.8: Outlier classification differences between ground truth and the automatic voxel-based cleaning. Note left axis: FPR. Right axis: FNR.

		True Condition		
	n = 8,275,821	**Positive**	**Negative**	
Predicted	**Positive**	66,204	3,740	Sensitivity = 60.6
Condition	**Negative**	43,668	8,162,209	Precision = 95.3
		FNR = 39.7	FPR = 0.04	

Table 5.3: Confusion matrix of A2 at 0.75 m resolution.

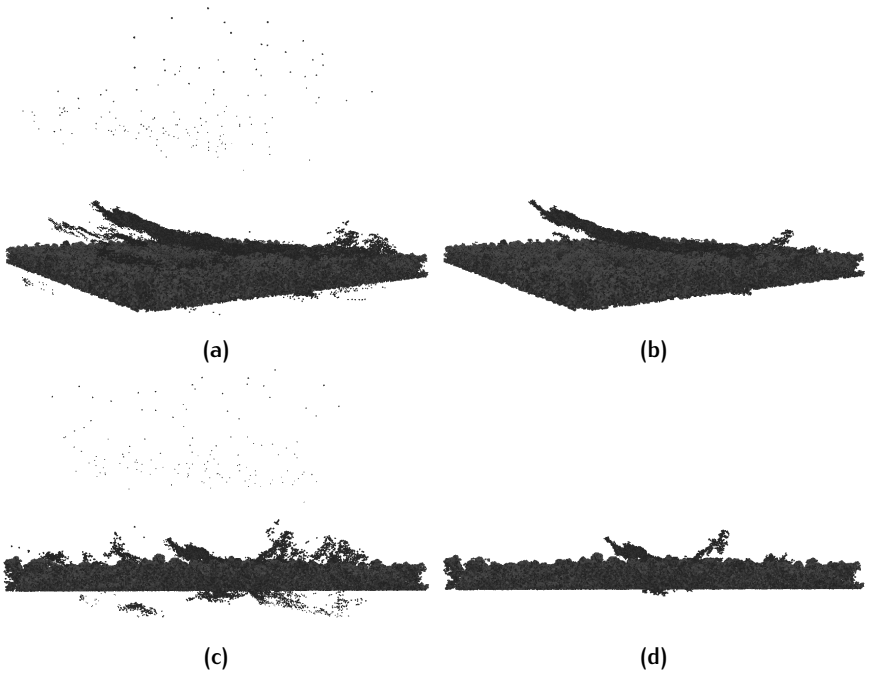


Figure 5.9: Point cloud B before and after outlier removal. (a) Point cloud B before cleaning with high type-1 outliers and type-3 outliers in clusters. (b) Cleaned version of (a). All outliers removed except one cluster penetrating the terrain in the centre. (c) Side view of (a). (d) After outlier removal of (c), clearly showing the remaining cluster noise in the center.

For datasets A1 (Table 5.6) and A2 (Table 5.3), the results are very convincing. At 0.75 m resolution 82.17% and 60.26% of the outliers were removed, while only 0.13% and 0.04% of the points were falsely classified as outliers. Up to 1 m, the cleaning process is very effective. Smaller than 0.75 m and larger than 1 m causes a FPR or FNR that is very high. It is as expected, when considering a larger voxel size the FPR decreases and the FNR increases.

For dataset B there was no manually classified reference dataset available. Nonetheless, visual inspection shows that at voxel size 0.75 m (8.7% of the points are classified as outliers), the method effectively cleans many outliers from a very noisy dataset (Fig. 5.9). All type-1 outliers and most of type-3 outliers are removed (no type-2 outliers in this dataset). However, a large cluster of type-3 outliers penetrates the terrain and is not removed after the cleaning process as seen in Fig. 5.9b and Fig. 5.9d (further explained in § 5.4.3).

Point cloud C is used to test the voxel-based method on a urban landscape. This dataset is outlier-free and can thus not be used to assess the sensitivity of outlier classification. However, its still very useful to evaluate the FPR for different scenes. Fig. 5.8c shows a very convincing FPR of only 0.02% at a voxel size of 1 meter. The FPR increases to 0.12% at 0.75 m and 1.05% at 0.50 m. Also by visual inspection, 0.50 m seems to be a too small resolution for accurate outlier classification, causing complete features (e.g. roofs, trees, lampposts) being removed. There is no difference noticeable between an urban landscape or a natural environment in terms of FPR.

The cleaned dataset of point cloud D (Fig. 5.10a) can be seen in Fig. 5.10b. This point cloud differs from the previous ones in terms of acquisition technique—dense-matching instead of ALS—and no intensity information is

(a) (b)

Figure 5.10: Point cloud D. (a) Raw point cloud D. (b) After outlier removal. Falsely removed features are indicated with circles.

available. The voxel-based method can cope with this by simply removing the intensity analysis from the workflow. The source point cloud is free of outliers so the FNR is not tested.

This experiment shows that a dense-matched point cloud is less suitable for the proposed voxel-based cleaning method. A FPR of 11.9% (voxel size = 0.50 m) is substantial larger than LiDAR point clouds tested in this thesis. Observing Fig. 5.8d shows that the FPR decreases with a larger resolution—5.3% (75 cm) and 2.65% (1 m). The consequence of classifying a large portion of false positives is that complete features are removed from the dataset. By observing the resulting point cloud (Fig. 5.10b), it can be seen that complete roofs and trees are removed (Fig. 5.11d). An acceptable FPR (0.64%) occurs at a 1.5 m resolution. However, we learned from previous tests that the FNR at 1.5 m is rather high, assuming the dataset contains similar types of outliers.

5.4.3 Common Classification Problems

Previous experiments showed several common problems that caused an increased FPR or FNR. Different situations causing these errors are explained here separately.

1. False Positive Rate

An increased FPR means good points are classified as outliers and thus wrongly removed from the dataset. Some findings are listed below:

- In most cases, a small voxel size (< 0.75 m) is not beneficial and results in parts of the surface being disconnected from the rest of the terrain. This is due to low local point densities, as happens near tree trunks. Fig. 5.4a shows several treetops in the dataset with low point densities at the trunks. As a result, several good points are classified as outliers.
- Although a larger resolution (> 0.75 m) definitively decreases the FPR between the optimal range of 0.75–1 m these problems still occur. Fig. 5.11 gives two examples where this happens. The first shows street posts (Fig. 5.11a) that have a small surface and thus (close to) zero

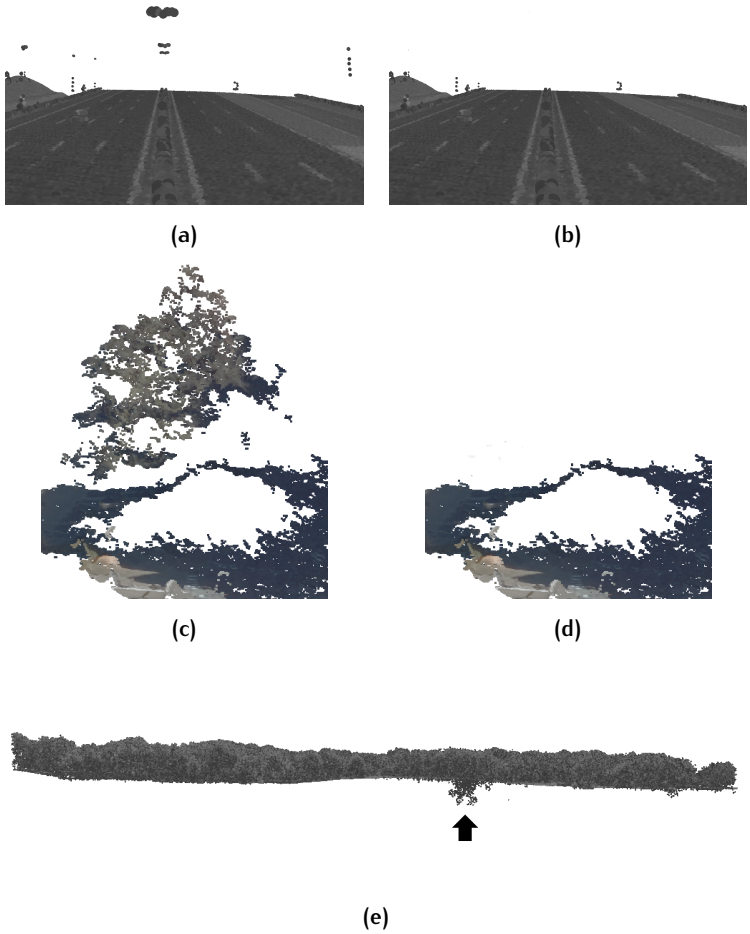


Figure 5.11: Common errors. (a) Street posts captured incomplete causing gaps. (b) Cleaned version of (a)—street posts are removed, because they are not connected to the terrain. (c) Top layer of vegetation is captured leaving empty space below. (d) Vegetation is wrongly removed after outlier detection. (e) Side view of A2; small fraction of type-3 noise remains below the surface after outlier removal.

hits from laser pulses. The surface that does reflect is hard to separate from outliers and filtered out in this case (Fig. 5.11b).

- Similarly, vegetation is wrongly removed from dataset D (Fig. 5.11d). These objects are classified as outliers, because they are not connected to the surface, i.e. the objects are floating. This occurs more often with dense-matched point clouds (D) than LiDAR point clouds. LiDAR has the potential to provide much richer spatial information about canopy characteristics in three dimensions. The reason behind this is that one emitted laser pulse can record the range to multiple targets, whereas a camera is limited to one data point per pixel. This is best demonstrated with an example of a tree. A laser pulse penetrates the tree and reflects on multiple features, including the ground. A camera only captures the top layer of a tree, leaving an empty space below (Fig. 5.11c).

Table 5.4: Performance of each operation for point cloud A1 and A2.

A1	*Method*	TP	FP	FN	FNR	FPR
	Density	47,331	2,688	35,589	42.92	0.05
	CCL	69,008	166,908	13,912	16.78	2.95
	CCL after closing	66,257	5,726	16,663	20.10	0.10
	LiDAR intensity	78,558	2,145,036	4,362	5.26	37.89
	Planarity	76,473	1,369,895	6,447	7.77	24.20
	Overall	68,134	7,109	15,475	18.66	0.12
A2						
	Density	50,142	2,875	59,730	54.36	0.04
	CCL	69,073	11,317	40,799	37.13	0.14
	CCL after closing	64,194	1,730	45,678	41.57	0.02
	LiDAR intensity	75,983	532,129	33,889	31.84	6.52
	Planarity	85,240	2,786,722	24,632	22.42	34.13
	Overall	66,204	3,740	43,668	39.74	0.05

2. False Negative Rate

Ideally, the method removes all outliers from the dataset automatically. In practice is this almost impossible. Even when detecting outliers manually it is not always clear to separate outliers from inliers when they appear very close to real world features. However, these situations have very little effect on the dataset, because usually it concerns just a handful of points.

- In the same context, this voxel-based method does not make the correct decision when outliers appear near the surface, i.e. within less than a voxel size distance away. The outliers are in this case physically connected to the surface and therefore part of the surface. This can cause a high FNR when considering a cluster touching the surface. An example of this can be seen in Fig. 5.11e, where a group of outliers is not filtered out, because they connect to the ground.
- Similarly, Fig. 5.9d shows a smile-shaped noise cluster penetrating the surface and is therefore not classified as outliers.
- Depending on the distance between outliers and inliers, a smaller voxel size can decrease the FNR. However, this would cause a higher FPR, as explained in the previous section.

5.4.4 Method Breakdown—Operation Evaluation

To understand if the method benefits from using multiple methods, each single operation needs to be evaluated. This section gives a detailed look into the performance of each individual operation that is selected for this thesis. As mentioned before, the selection of operations is a choice I made based on several expectation explained in Chapter 4. However, it is possible that an operation is redundant or not effective and thus not beneficial for the overall method. To investigate this, the FPR and FNR is computed for each operation and used to study how this effects the final classification.

Table 5.4 describes the performance of each operation for dataset A1 and A2. It is easy to see that none of the operations have similar performance

results. This indicates that no operation is redundant. Moreover, the performance of the overall method outer performs each single operation when considering both FPR and FNR. Especially the FPR seems to benefit from using multiple methods.

This is in expectation with the initial design motivation. The first three operations are to detect outliers and the last two to minimize false positives.

5.5 COMPUTATION TIME AND SCALABILITY

There are two factors that influence the scalability of the method in terms of computation time and memory allocations: (1) voxel size; (2) size of dataset. This section evaluates both for different sizes. Fig. 5.12 shows computation times for each operation. The experiments shown in these figures are derived on a HP EliteBook 8570w with an Intel Core i7-3630QM CPU @ 2.40GHz and 8 Gb RAM.

5.5.1 Voxel size

The number of voxels m in a 3D raster increase cubically (3rd power) for every time resolution s (i.e. voxel size) halves. Computation time may become unpractical when resolution is chosen too fine. Therefore, algorithmic performance is bounded to $O(n^3)$ for an input n expressed in distance/voxel size. This means n gets twice as big, every the voxel size halves. Fig. 5.12a demonstrates the influence of the voxel size (point cloud has 2.5M points). Operations which are solely dependent on the number of voxels (i.e. CCL, density analysis) entail a $O(n^3)$ time complexity, as is shown in the graph. For voxelization, planarity and intensity analysis, the number of points inside each voxel influences the complexity, i.e. a coarser resolution may result in less voxels, but include more points per voxel.

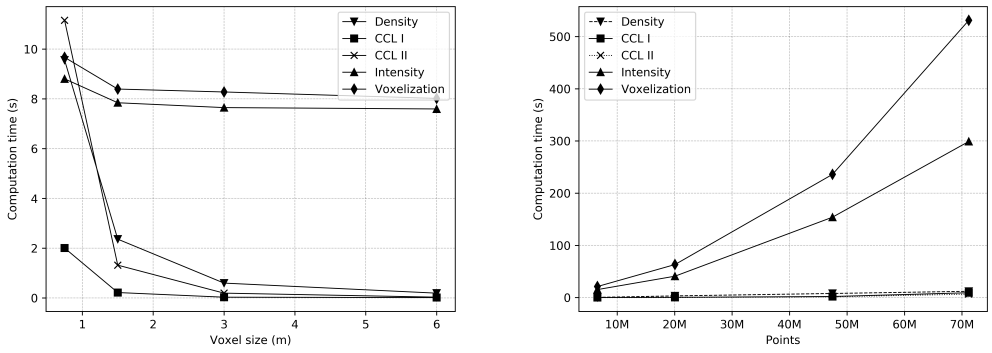


Figure 5.12: Computation times. (a) Time per spatial resolution. (b) Relation between computation time and size of point cloud (resolution of grid = 2 m).

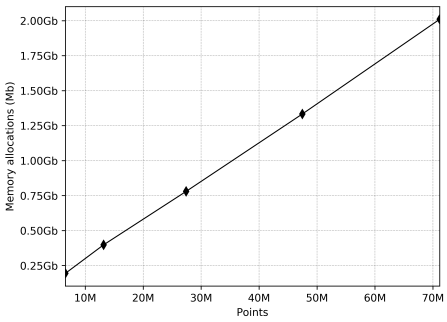


Figure 5.13: Memory allocations for voxelization of different sized point clouds.

5.5.2 Size of dataset

Point cloud size (i.e. number of points) effects computation time in a similar way. Voxelization of a larger point cloud results in more voxels and thus more computations. Computation time of the proposed method grows linear with the number of voxels, i.e. $O(m)$ for m is number of input voxels. However, in practice datasets do not scale per voxel. Instead, increasing the area size from 1 km $\times$ 1 km to 2 km $\times$ 2 km grows the number of voxels exponentially. This experiment is shown in Fig. 5.12b for a resolution of 2 m; it shows how computation time increases for larger datasets (voxel size is fixed). The plotted datasets are related to 0.5 km, 1 km, 1.5 km and 2 km squared tiles, because the number of voxels actually depend on the bounding box of the dataset (and voxel size) and not the number of points the cloud contains.

Both voxel size and increasing the dataset results in more voxels and thus more computations. However, comparing Fig. 5.12b and Fig. 5.12a shows different time complexities. This is simply, because in practice choosing a twice as large point cloud (area), usually results in more voxels in only x and y direction. Therefore is the performance bounded to $O(d^2)$ (when d is expressed in edge length of the bounding box), on the condition that the terrain is flat. This is what can be seen in Fig. 5.12b for operations solely dependent on number of voxels. Whereas voxel size influences the number of voxels in x, y and z direction.

5.5.3 Memory allocations

As seen in the previous section, building the voxel grid structure takes up for most of the computation time. It also accounts for the largest memory allocations. Fig. 5.13 shows a linear relationship between the number of points and memory allocations. At around 70 millions points 2.0 Gigabytes of memory needs to be reserved for the execution of the algorithm. Compared to only a few hundred Kilobytes for CCL on a binary grid is this a lot. Meaning, in this case a computer with 8 Gb of RAM would run out of memory with a point cloud of roughly 280 million points. Because the algorithm is dependent on reading the point cloud into the memory, the effect of voxel size is almost negligible in this matter.

		True Condition		
	n = 5,743,977	**Positive**	**Negative**	
Predicted Condition	**Positive**	62,821	5,699	Sensitivity = 75.7
	Negative	20,099	5,655,358	Precision = 91.7
		FNR = 24.2	FPR = 0.10	

Table 5.5: Cleaning quality with LAStools.

		True Condition		
	n = 5,743,977	**Positive**	**Negative**	
Predicted Condition	**Positive**	68,134	7,109	Sensitivity = 82.2
	Negative	14,786	5,653,948	Precision = 90.6
		FNR = 17.8	FPR = 0.12	

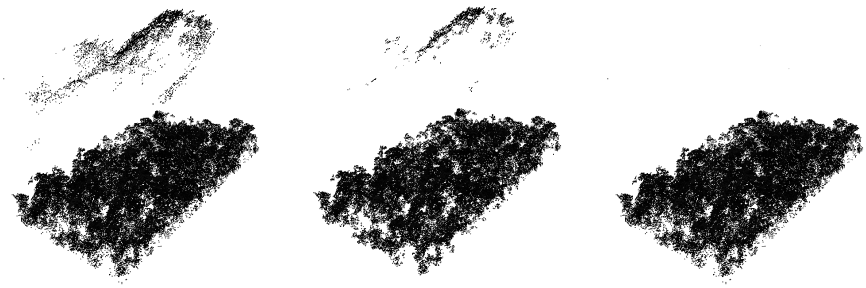
Table 5.6: Cleaning quality of A1 with proposed method.

5.6 COMPARISON TO EXISTING METHOD

There are several outlier filtering tools freely available. Some of the issues these have are discussed in § 3.1. A comparison of methods can highlight these issues and emphasize on where the designed voxel-based method outer performs existing tools.

In this comparison LAStools (see § 3.1) is compared to my method. Both filter tools are used to clean dataset A1. A grid resolution of 0.75 m is used for each case. Furthermore, best results with LAStools were obtained with a density threshold of 3. Table 5.5 and Table 5.6 show the quality performance of LAStools and my method.

In this case, LAStools performs worse in terms of FNR. My method decreases the FNR with 39% with respect to LAStools. Both methods perform similar in terms of FPR.



(a) (b) (c)

Figure 5.14: Comparison cleaning of clustered outliers. (a) Source point cloud with clustered outliers. (b) Cleaned with LAStools. (c) Cleaned with proposed method.

The most striking example of performance difference is for type-2 outliers (clusters), as seen in Fig. 5.14a. The limitations of LAStools are easy noticeable for this situation; clustered outliers are not cleaned well, as is shown in Fig. 5.14b. Many outliers are not detected, because of a high local density in a cluster. However, Fig. 5.14c shows that the binary voxel-based method completely removes type-2 outliers.

6 DISCUSSION & FUTURE WORK

This chapter elaborates on a few possible improvements for the proposed method. For some issues I propose a solution. However, they are not included in the methodology, because the solution (a) needs more research; or (b) is a theoretical solution and is not yet implemented in the prototype version.

An assessment and a comparison to existing tools showed the proposed method is able to detect all types of outliers effectively. However, a few points of discussion can be identified. This chapter dicusses the following: scalability of the method (§ 6.1), separation by water (§ 6.2), confidence of final classification (§ 6.3), binary raster processing vs. point-wise feature extraction (§ 6.4) and manual parameter adjustments (§ 6.5).

6.1 EXPERIMENTS WITH SCALABILITY ISSUES

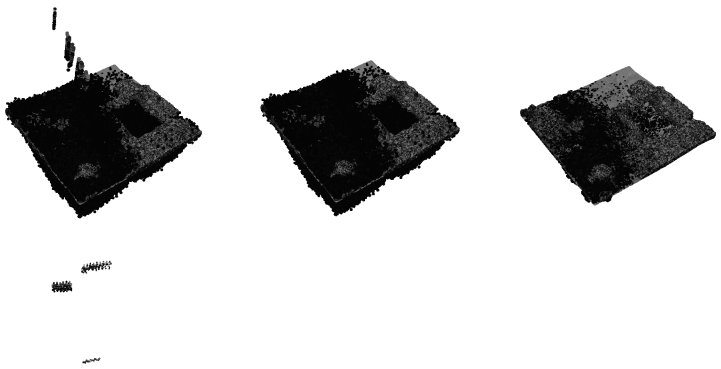
In practice the scalability is limited by the number of voxels (determined by resolution or point cloud size)—in terms of computation time—and size of the point cloud—in terms of memory allocations. Moreover, there are two situations that I came across for which I propose a solution. § 6.1.1 handles very high and low outliers, and § 6.1.2 presents a streaming solution to handle point clouds larger than your RAM allows.

6.1.1 From Coarse To Fine

In some cases, outliers appear very high and/or low in the dataset (Fig. 6.1a). This leaves large areas of empty space between good points and points that need to be removed. All this space is filled with voxels and results in unnecessary computations that are done on these voxels. The point cloud shown in Fig. 6.1a covers a 0.5×0.5 km area and contains very high and low outliers. Therefore, the dimensions of the bounding box are: $0.5 \times 0.5 \times 4.5$ km. A grid with a 1 meter resolution would contain more than 1 billion voxels.

The solution to prevent the algorithm from constructing many unnecessary empty voxels, is to first build a grid with a coarse resolution followed with the desired resolution. A coarse resolution creates less voxels, but still removes all of the very high and low outliers (and thus leaves a much smaller bounding box). After the first iteration, a finer resolution is chosen to run the method with the desired level of detail.

Fig. 6.1c shows much improvement after completing a first iteration with a resolution of 50m. This grid contains only 9,000 voxels, but still removes the very high and low outliers (Fig. 6.1b). The resulting point cloud is input for the second iteration with a resolution of 1m. The gridded point cloud now only contains 9.7M points (instead of 1B). This finer resolution allows to remove the remaining outliers (Fig. 6.1c).



(a) (b) (c)

Figure 6.1: From coarse to fine. (a) Raw point cloud with very high and low outliers. (b) First iteration of outlier removal with 50 m resolution. (c) Second iteration, with 0.75 m resolution.

6.1.2 Streaming

Massive point cloud datasets could easily overload the memory of most commodity computers, as explained in § 5.5.3. Instead of loading the entire dataset into memory, streaming solutions are proposed. Streaming algorithms sequentially read a stream of data, pipe it to a processor and free up the memory after the data is processed.

Usually, streaming can only succeed when a portion of points have spatial coherence. Meaning, when some spatial operation (e.g. constructing a Delaunay triangulation) has to be done on the points there has to be a correlation between their location in space and location (index) in the stream. Therefore, studies propose to first sort (e.g. hilbert sort) the points before streaming.

Isenburg et al. [2006] avoids a heavy sorting step by including *finalization tags* in the stream that indicate when no more points in the stream will fall in specified regions. The finalizer reads a stream of raw points three times from disk. During the first pass it computes and partitions the bounding box into equal cells. During the second pass it counts the number of points for each cell. And a third pass to add a finalization tag into the stream whenever a cell's count is reached. Then, a spatially finalized point stream pipes to a processing step.

The proposed methodology to detect outliers has no streaming solution. Therefore, a theoretical implementation is given here (see Fig. 6.2). This streaming solution is split in two tasks. One for processing the binary grid using Connected Components Labeling CCL. A second for the group-based processing. Both solutions have different requirements, because CCL needs global specifications to compute connected regions, whereas the group-based operations only require local information.

1. Streaming Points into Binary Grid

The first streaming solution benefits from the binary regular grid structure and is not dependent on spatial coherence. Binary rasterization of an input point cloud reduces the storage size to a minimum, because every grid cell only needs 1 bit (to store a 1 or 0). Therefore, massive point clouds that

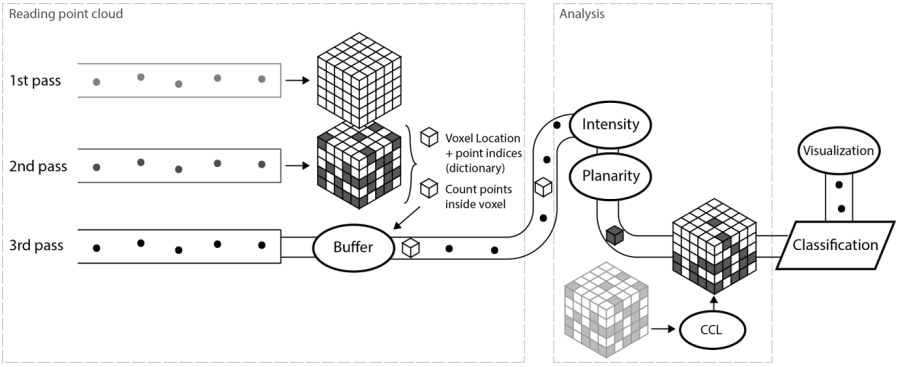


Figure 6.2: The streaming solution reads the points thrice and pipes a spatially finalized voxel stream to the intensity and planarity analysis, which then compares the result with the CCL output, and finally writes out a cleaned point cloud.

are too large for the memory could be completely read into memory when represented in a binary array.

In this case, the streaming algorithm only has to fill this binary grid with points from the stream, and is thus independent of spatial coherence. After streaming is completed, a binary grid remains that fits completely in the memory which then can be used for further processing.

Although, this only requires one pass over the input stream (assuming a bounding box can be created from the point cloud's metadata), all point cloud processing is limited to the raster space and therefore, not comparable to the streaming solution of Isenburg et al. [2006].

This could be implemented as follows:

- The first step is to create an empty (all zeros) 3D binary voxelgrid from the bounding box [1] as explained in § 4.2. Two data structures are built for this, including a binary array representing the voxelgrid and a dictionary to describe which data points each voxel contains.

- Then the stream pipes to the binary grid. For each point four actions are done:
  1. the voxel that contains the point changes from 0 to 1.
  2. the voxel location (i, j, k)—and the point's index it contains—are added to the dictionary, as well as the count.
  3. re-calculate average intensity of point cloud with new point.
  4. remove point from memory before a new point from the stream is read.

- When the stream is complete, the resulting binary grid is of equal size as before the stream.

- This grid is then used for the connected components labeling algorithm and closing operation, which results in the first outlier classifications.

1 According to the American Society for Photogrammetry & Remote Sensing (ASPRS) minimum and maximum x, y, z coordinates can be stored in the public header of a `.las` file. When this information is not available an extra iteration over the points in necessary to find these locations and construct the bounding box.

2. Streaming for Group-Based Processing

This completes all processing in the raster space, and only analysis of the group-based points (i.e. density, intensity and planarity) is required. Group-based analysis only succeeds if the input stream has sufficient spatial coherence. Accordingly, a slightly different streaming method can be used, inspired by the work of Isenburg et al. [2006]. Implementation details are given below:

- This solution continues with the regular grid and dictionary created during the first pass.
- A second pass is necessary, which buffers all the points in each cell until the cell's counter reaches zero. When a cell reaches zero, all inner points are placed in the output stream.
- The output stream is then piped to planarity- and intensity analysis, and is classified as described in § 4.6 - Planarity and § 4.5 - Lidar Point Intensity.
- This classification is then compared with the binary classification from the first pass. As described in § 4.1.4, it decides if the voxel contains outliers or not, writes it to the output file, and frees its memory.

One advantage is the output can be piped to a visualization tool. Data begins to appear in an early stage while still consuming input, by doing all (fast) raster processing first and then piping it to (slow) group-based operations.

However, an observant reader might object that creating an empty binary grid from the point cloud's bounding box—and keep it in memory—sets a limit to the size of the input point cloud. This is a disadvantage, however memory requirements for binary arrays are very limited allowing voxelization of massive point clouds with commodity computers.

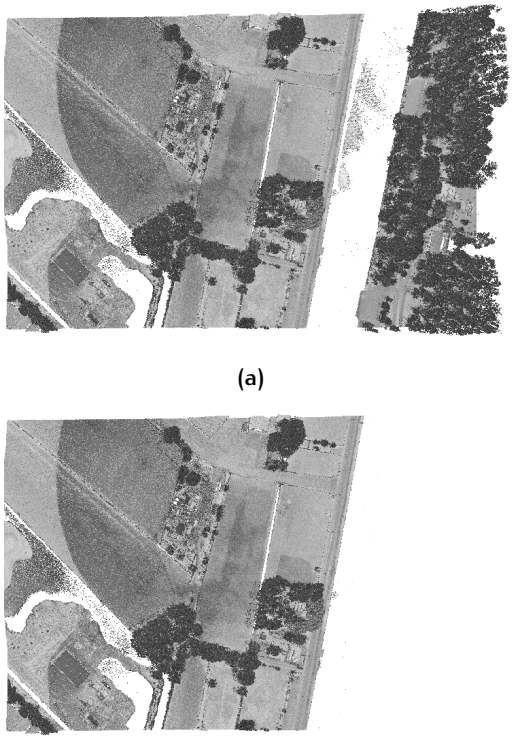
An experiment with dataset A1 shows the memory requirements for a streaming solution. Point cloud A1 has a size of 180 Mb. After voxelization, the binary grid has a size of 5.1 Mb for $500 \times 500 \times 179$ (44.8 M) voxels. A computer with 8 Gb's of RAM is capable to easily handle billions of voxels. Although, this solution allows to process substantially larger datasets, there is a limit. Therefore, a fully streamed solution for computing connected components for over a trillion voxels is given in Isenburg and Shewchuk [2011], which can be addressed in future work.

6.2 SEPARATION BY WATER

Rare cases where rivers separate land in the target area can cause problems for the proposed outlier detection method, because the two land-parts get unconnected.

Most of the aerial lidar sensors use a wavelength in the infrared. Infrared wavelengths get rapidly absorbed by water. Therefore, water surfaces may have no measurements. In the point cloud dataset this can cause large gaps, as is seen in Fig. 6.3a. Gaps and unconnected parts result in an increased FPR–as explained in § 5.4.3—because they are wrongly filtered out (Fig. 6.3b).

In this section I propose a theoretical solution, which is also practical to implement in the proposed method.



(a)

(b)

Figure 6.3: (a) A river separates two land parts in the point cloud. (b) The right half gets filtered out, as it is unconnected to the largest area.

Consider Fig. 6.4, which is a side view of the same point cloud shown in Fig. 6.3a. A river splits the land surface into region 1 and region 2. By computing the incline between the two regions, an assumption about the position related to each other can be made. Regions with negligible incline relative to the ground surface (region 1) will also be considered as ground. This is done with the following steps:

- After finding the largest connected component G as explained in § 4.4, compute its centroid S_G.
- Select all regions with a volume larger than a specified threshold (only consider possible large land areas).
- For each selected region R find the centroid S_R and compute the incline α between S_G and S_R.
- Consider regions with α smaller than a specified threshold as ground surface and not as outlier.

Following these steps, region 2 has a α of 1.1° and is not classified as clustered outlier. Similarly, unconnected higher and lower regions have a larger α and are very likely to contain outliers.

This experiment showed promising results for handling situations where rivers split the target area. However, more research has to be done. Although, measuring incline can be very effective in flat areas, it may be less helpful in sloped areas. This may require to measure the incline in a different way. Instead of finding the centroids, find the two points closest to each

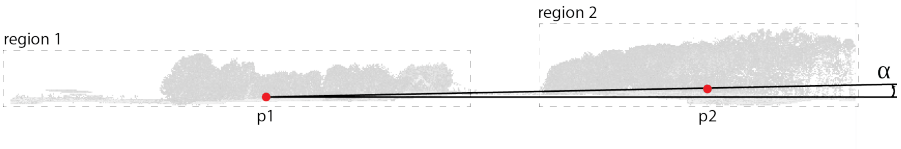


Figure 6.4: The angle α between centroids p1 and p2 is below a specified threshold. Therefore points in region 2 are not classified as outliers.

other on both regions. Because water is also flat in sloped areas, this incline should also be negligible.

6.3 CLASSIFICATION

This section discusses the final outlier classification, which uses map algebra. The method described in § 4.1.4, simply sums all boolean classifications and classifies each voxel where $sum \geq 3$ as outlier. Although it works very well with the selected operations for this study, this threshold is arbitrary and might not hold up when more or less operations are used. Moreover, all operations have equal weight, even though CCL is far more effective to detect outliers than intensity- or planarity analysis.

6.3.1 Voxel Classification Using Machine Learning

Alternatively, supervised learning classifiers can be exploited to detect outliers. Statistical binary classification is widely studied in the field of machine learning. Machine learning methods can predict the probability, given a new observation, to which predefined class it belongs. Plaza-Leiva et al. [2017]; Plaza et al. [2015] successfully used Support Vector Machine (SVM) and Neural Network (NN) to classify voxels based on the spatial distribution of inner points.

Such a classifier measures characteristics, also called features, of the input data. Choosing informative and independent features is a crucial step for effective classification [Bishop, 2006]. The five operations presented in this thesis extract useful features that can be used to train a classifier. The set of features observed in this thesis can be used to describe a feature vector, which is then a five-dimensional vector that represents a voxel. However, training such a classifier requires training data containing classified outliers. Developing a supervised outlier classifier can be addressed in future work, when this data is available.

6.4 BOOLEAN GRID VS. GROUP-BASED

Issues with point cloud processing often include memory requirements and efficiency (speed) (see § 5.5 and § 6.1). Although, it is not a main goal of this study to present a efficient and fast outlier detection tool, it is essential

for commercialized processing tools. Moreover, certain design choices are based on the desire for fast processing, such as the use of a binary grid. Therefore, aspects that are less efficient should not be ignored.

Logically, the choice for a series of operations requires more computations and is thus less efficient. Computation time is even more influenced by operations going back to the point level, instead of processing the binary grid. This is shown in Fig. 5.12, where operations only using the binary grid (i.e. CCL), significantly outperform operations on a group of points (i.e. density, intensity, planarity).

This method makes use of the grid structure to group points prior to analysis. However, a solution where all processing is done in the raster space could speed up the method significantly. Image processing is an active and large research field. Although many raster processing techniques are reviewed for this study, advanced computer vision algorithms were out of scope for this thesis.

6.5 MANUAL PARAMETER ADJUSTMENTS

Manual input from the user is almost unavoidable and can be seen as a drawback. Many point cloud processing tools rely on tuning several parameters in order to achieve better results (also concluded in Chapter 3). Therefore, tools should achieve good results while minimizing user input.

This study uses several thresholds to process the data and are arguably manual parameters. These include:

- Voxel size;
- Density threshold in local neighborhood (number of points);
- Threshold for outlying intensity values;
- Curvature threshold to define planar surfaces;
- Outlier classification threshold with boolean algebra (also discussed in § 6.3).

This study showed that voxel size has a great influence on the outcome. Attempts to automate voxel size selection needs further research. A list of resolution criteria is given in § 4.2. Also, Fig. 5.8 showed good results with a resolution between 0.75 and 1.0 meter for the analyzed datasets. This range holds for point clouds with different densities.

7 CONCLUSIONS & RECOMMENDATIONS

This chapter gives the conclusions from this study. First the research questions are answered in § 7.1.1. Secondly, the main contributions (§ 7.1.2) are given. Finally, the recommendations are summarized in § 7.1.3.

7.1 CONCLUSIONS

This graduation project described an algorithm for removing (clustered) outliers from aerial LiDAR data. The algorithm can distinguish outliers from low density features, as opposed to algorithms that base their decisions only on local neighborhoods. It showed that a voxel structure in combination with raster processing techniques allow for effective and efficient point cloud analysis. However, voxelization of the source point cloud requires a careful selection of the resolution to achieve the desired results. Furthermore, a streaming solution can be developed, which makes the method suitable for processing large areas.

Even though the detection quality is improved compared to existing methods, a few issues need more attention. The major criticism on the voxel-based approach, as developed in this thesis, is the arbitrary selection of analysis techniques and the final classification by combining them, as is stated in § 6.3. Moreover, three operations (density, intensity, planarity) make use of the voxel structure by grouping inner points prior to analysis (group-based analysis), instead of analyzing the binary grid itself (which CCL does). This could be argued as a drawback, because processing a binary grid is far more efficient than processing group-based points.

Nevertheless, this study showed the benefits of combining different operations to classify outliers. It showed substantial improvement by minimizing false positives, while detecting more outliers than existing tools. Additionally, a gridded solution is presented to combine all methods and analyze the data in a coherent way.

7.1.1 Research questions

All research questions are answered here. It starts with answering the sub research questions and concludes with the answer to the main research question.

- **Question 2**. *How can raster processing techniques—in particular Connected Components Labeling—be used on a 3D voxelized point cloud to detect outliers?*

  Outliers can be detected from binary voxelized point clouds in three steps. First, the CCL algorithm separates voxels in different regions. Second, count all regions sizes. Third, classify all voxels not connected to the largest region as outliers.

A preprocessing step to fill gaps proved to enhance the dataset in order to classify less False Positives. By using the closing operator (mathematical morphology) on a bit voxelmap, the dataset is restructured in such a way that gaps between unconnected regions may get filled. Doing this before the CCL algorithm may result in less fragmented regions.

- **Question 3**. *What is the influence of different voxel resolutions for outlier detection algorithms, in terms of accuracy and computation load?*

  Voxel size has a great influence on accuracy of the proposed method. A resolution larger than 1 meter is too coarse to capture the details in the environment. Moreover, a larger voxel size aggregates a bigger set of points into one voxel and may include outliers close to the surface.

  On the other side, a resolution chosen too fine (<0.75 m) causes a significant increase in False Positives due to discontinuities in the voxel model. Experiments showed best results between 0.75 m and 1 m resolution.

  Voxel size also heavily influences computation time. The number of voxels in a 3D raster increases with the 3rd power of the resolution (when expressed in voxels/distance). A too fine resolution (< 0.75 m) may cause unpractical running times. This adds to the argument of not using a voxel size smaller than 0.75 m. Based on accuracy results, a resolution of 0.75–1.0 seems desired.

- **Question 4**. *Can a series of analysis methods within a voxel model achieve higher classification quality?*

  Yes, the False Positive Rate can be decreased significantly.

  Firstly, intensity analysis shows great potential in achieving better classification results. However, more research is required to sustain the claimed correlation between outliers and intensity values. Also, better statistical models could be exploited for outlier separation.

  Secondly, planarity analysis decreases the FPR in specific cases where unconnected man made (flat) objects are classified as outliers

- **Question 5**. *What outlier detection quality is achievable and how to influence the trade-off between True Positives and False Positives?*

  The proposed method can achieve very satisfactory results in terms of both TP and FP. For two datasets 82 % and 60 % of the outliers were successfully removed, while only 0.13 % and 0.04 % of points were falsely classified as outliers. All identified types of outliers are detected, including clusters.

  In some special cases the method is not able to clean outliers well. When outliers are spatially close to the surface its hard to separate them from the good points. Also, when a cluster of outliers is connected to the surface, they are not removed.

  The trade-off of TPs and FPs can be minimized by choosing a suitable voxel size. Moreover, analysis such as intensity and planarity can identify good points from wrongly classified outliers.

  Compared to an existing tool lastools, the proposed method yielded better results in terms of both TPs and FPs.

- **Question 6**. *What is the influence of scaling the dataset on the algorithm in terms of time and memory?*

  The method scales linear in both time and memory when input is defined by voxels and points. This means the computer can run out of memory when the dataset is too large. In this case, a streaming solution can be implemented which allows the algorithm to process massive point clouds.

- **Question 7**. *Can the same outlier detection algorithm be used for both LiDAR (aerial) datasets and DIM datasets?*

  No, Dense Image Matching (DIM) usually has more gaps in the dataset than LiDAR datasets due to occlusion. A fragmented voxel model is not beneficial for the detection of outliers using CCL.

And finally, the main research question is answered:

- **Question 1**. *Is a voxel-based approach a viable option to automatically detect outliers from aerial LiDAR point clouds?*

  Yes, based on the quality metric and experiments presented in this study, it can be concluded that the proposed voxel-based approach performs well in terms of outlier detection accuracy (minimizing both FPR and FNR). A comparison showed higher classification accuracy than existing distance-/density-based methods. Furthermore, the method is suitable for the implementation of a streaming solution, which would allow the method to process massive point clouds.

7.1.2 Contributions

The main contributions of this thesis are:

- Designing a voxel-based solution to detect outliers from aerial LiDAR point clouds. This study extensively explores different processing techniques and integrates them into a voxel-based methodology.

- From the processing techniques used in this thesis, most contributions are made on using connected components to detect outliers. This includes dealing with gaps/holes in the data and selecting an effective resolution for the voxelgrid.

- This study presents a new method to detect clustered outliers, which is an ongoing problem. A literature (and related work) review influenced the design of using image processing techniques on a binary grid. This study showed the effectiveness of using these methods, which can be used for many other point cloud analysis and classification problems in a efficient way.

7.1.3 Recommendations & Remarks

This section gives a few recommendations, that can be helpful when researching this topic.

First of all, it is challenging to detect all outliers without classifying false positives. Extracting more features may be unwanted, as this requires more computations and is possibly redundant. In essence, the goal is to remove

outliers from point clouds as efficient and effective as possible. An advantage of the proposed method is that it is very suitable to be incorporated in a tool, and make the user's workflow more efficient. A user interface could allow the operator to quickly review the result in cluster format, instead of point by point. So even if the tool miss-classifies certain clusters, it can quickly be recovered.

The same can be argued for the problem of rivers separating the land in some areas (see § 6.2). Extra analysis can overcome this problem, but the separated cluster can also simply be reclassified by the user. Therefore, instead of only aiming for a errorless classifier, it should also be considered how the tool will be used. This also requires special attention regarding manual adjustment of parameters (see § 6.5).

Secondly, the datasets used in this study all showed a relation between LiDAR point's intensity and being an outlier. This information is not exploited by other outlier detection algorithms, although it proved to be beneficial. However, a more advanced statistical model is needed and can be addressed in future work. This also requires a study to investigate the correlation between intensity and outliers in aerial point clouds.

Thirdly, statistical binary classification is widely studied in the field of machine learning. Moreover, computer vision research actively studies learning-based methods. These methods can be explored with rasterized point clouds (see § 6.3.1). This thesis already showed that simple image processing techniques can be effective.

BIBLIOGRAPHY

Angluin, D. and Laird, P. (1988). Learning from noisy examples. *Machine Learning*, 2(4):343–370.

Arge, L., Green Larsen, K., Mølhave, T., and van Walderveen, F. (2010). Cleaning massive sonar point clouds. In *Proceedings 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 152–161. ACM.

Barnett, T. L. (1994). *Outliers in Statistical Data*. JOHN WILEY & SONS INC.

Beraldin, J.-A. (2004). Integration of laser scanning and close-range photogrammetry – the last decade and beyond. In *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Science Congresss*, pages 972–983.

Birchfield, S. (2017). *Image Processing and Analysis*. Cengage Learning.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag.

Boulch, A., de La Gorce, M., and Marlet, R. (2014). Piecewise-planar 3d reconstruction with edge and corner regularization. *Computer Graphics Forum*, 33(5):55–64.

Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. (2000). LOF: Identifying Density-Based Local Outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. ACM Press.

Chen, Q., Gong, P., Baldocchi, D., and Xie, G. (2007). Filtering airborne laser scanning data with morphological methods. *Photogrammetric Engineering & Remote Sensing*, 73(2):175–185. includes method for filling missing data (water). Good explenation of filter outliers with morphological operators.

Eisenbeiss, H. (2009). *UAV photogrammetry*. PhD thesis.

El-Hakim, S. F. and Beraldin, J. A. (1994). Integration of range and intensity data to improve vision-based three-dimensional measurements. In El-Hakim, S. F., editor, *Videometrics III*. SPIE.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874.

Fisher, P. (1997). The pixel: A snare and a delusion. *International Journal of Remote Sensing*, 18(3):679–685.

Goodchild, M. F. (1992). Geographical data modeling. *Computers & Geosciences*, 18(4):401–408.

Gorte, B. and Pfeifer, N. (2004). Structuring laser-scanned trees using 3d mathematical morphology. In *International Archives of Photogrammetry and Remote Sensing*, volume 35, pages 929–933.

Han, X.-F., Jin, J. S., Wang, M.-J., and Jiang, W. (2017). Guided 3d point cloud filtering. *Multimedia Tools and Applications*.

Hao, W. and Wang, Y. (2014). Classification-based scene modeling for urban point clouds. *Optical Engineering*, 53(3):033110.

Hawkins, D. (1980). *Identification of Outliers*. Springer Netherlands.

Heinzel, J. and Huber, M. (2016). Detecting tree stems from volumetric TLS data in forest environments with rich understory. *Remote Sensing*, 9(1):9.

Isenburg, M. (2013). Laszip: lossless compression of lidar data. *Photogrammetric Engineering & Remote Sensing*, 79(2):209–217.

Isenburg, M., Liu, Y., Shewchuk, J., and Snoeyink, J. (2006). Streaming computation of delaunay triangulations. *ACM Transactions on Graphics*, 25(3):1049.

Isenburg, M. and Shewchuk, J. R. (2011). Streaming connected component computation for trillion voxel images.

Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323.

Jain, R., Kasturi, R., and Schunck, B. G. (1995). *Machine Vision*.

Jenk, P., Wand, M., Bokeloh, M., Schilling, A., and StraSer (2006). Bayesian point cloud reconstruction. *Comput Graph Forum*, 25(3):379–388.

Johnson, T., Kwok, I., and Ng, R. T. (1998). Fast computation of 2-dimensional depth contours. In *4th International Conference on Knowledge Discovery and Data Mining*, pages 224–228.

Kilian, J., Haala, N., and Englich, M. (1996). Capture andevaluation of airborne laser scanner data. In *International Archives of Photogrammetry and Remote Sensing*, pages 383–388.

Knorr, E. M., Ng, R. T., and Tucakov, V. (2000). Distance-based outliers: algorithms and applications. *The VLDB Journal The International Journal on Very Large Data Bases*, 8(3-4):237–253.

Kobler, A., Pfeifer, N., Ogrinc, P., Todorovski, L., Oštir, K., and Džeroski, S. (2007). Repetitive interpolation: A robust algorithm for DTM generation from aerial laser scanner data in forested terrain. *Remote Sensing of Environment*, 108(1):9–23.

Kumler, M. P. (1994). An intensive comparison of triangulated irregular networks (TINs) and digital elevation models (DEMs). *Cartographica: The International Journal for Geographic Information and Geovisualization*, 31(2):1–99.

Lalonde, J.-F., Vandapel, N., Huber, D. F., and Hebert, M. (2006). Natural terrain classification using three-dimensional ladar data for ground robot mobility. *Journal of Field Robotics*, 23(10):839–861.

Lehtomaki, M., Jaakkola, A., Hyyppa, J., Lampinen, J., Kaartinen, H., Kukko, A., Puttonen, E., and Hyyppa, H. (2016). Object classification and recognition from mobile laser scanning point clouds in a road environment. *IEEE Transactions on Geoscience and Remote Sensing*, 54(2):1226–1239.

Lemmens, M. (2011). *Geo-information*. Springer Netherlands.

Li, X., Zhang, Y., and Yang, Y. (2017a). Outlier detection for reconstructed point clouds based on image. In *2017 First International Conference on Electronics Instrumentation & Information Systems (EIIS)*. IEEE.

Li, Y., Yong, B., van Oosterom, P., Lemmens, M., Wu, H., Ren, L., Zheng, M., and Zhou, J. (2017b). Airborne LiDAR data filtering based on geodesic transformations of mathematical morphology. *Remote Sensing*, 9(11):1104.

Maligo, A. and Lacroix, S. (2017). Classification of outdoor 3d lidar data based on unsupervised gaussian mixture models. *IEEE Transactions on Automation Science and Engineering*, 14(1):5–16.

Matkan, A. A., Hajeb, M., Mirbagheri, B., Sadeghian, S., and Ahmadi, M. (2014). SPATIAL ANALYSIS FOR OUTLIER REMOVAL FROM LIDAR DATA. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-2/W3:187–190. good introduction to lidar and outliers in general.

Meng, X., Wang, L., Silván-Cárdenas, J. L., and Currit, N. (2009). A multi-directional ground filtering algorithm for airborne LIDAR. *ISPRS Journal of Photogrammetry and Remote Sensing*, 64(1):117–124.

Nourian, P., Gonçalves, R., Zlatanova, S., Ohori, K. A., and Vo, A. V. (2016). Voxelization algorithms for geospatial applications: Computational methods for voxelating spatial datasets of 3d city models containing 3d surface, curve and point data models. *MethodsX*, 3:69 – 86.

Nurunnabi, A., West, G., and Belton, D. (2015). Outlier detection and robust normal-curvature estimation in mobile laser scanning 3d point cloud data. *Pattern Recognition*, 48(4):1404–1419.

Papadimitriou, S., Kitagawa, H., Gibbons, P., and Faloutsos, C. (2002). LOCI: fast outlier detection using the local correlation integral. In *Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405)*. IEEE.

Pingel, T. J., Clarke, K. C., and McBride, W. A. (2013). An improved simple morphological filter for the terrain classification of airborne LIDAR data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 77:21–30.

Plaza, V., Gomez-Ruiz, J. A., Mandow, A., and Garcia-Cerezo, A. (2015). Multi-layer perceptrons for voxel-based classification of point clouds from natural environments. In *Advances in Computational Intelligence*, pages 250–261. Springer International Publishing.

Plaza-Leiva, V., Gomez-Ruiz, J., Mandow, A., and García-Cerezo, A. (2017). Voxel-based neighborhood for spatial shape pattern classification of lidar point clouds with supervised learning. *Sensors*, 17(3):594.

Quan, Y., Song, J., Guo, X., Miao, Q., and Yang, Y. (2016). Filtering LiDAR data based on adjacent triangle of triangulated irregular network. *Multimedia Tools and Applications*, 76(8):11051–11063.

Santamaria-Navarro, À., Teniente, E. H., Morta, M., and Andrade-Cetto, J. (2014). Terrain classification in complex three-dimensional outdoor environments. *Journal of Field Robotics*, 32(1):42–60.

Shapiro, L. and Stockman, G. (2001). *Computer Vision*. Prentice Hall.

Shen, J., Liu, J., Zhao, R., and Lin, X. (2011). A kd tree based outlier detection method for airborne lidar point clouds. In *2011 International Symposium on Image and Data Fusion*. IEEE.

Silván-Cárdenas, J. and Wang, L. (2006). A multi-resolution approach for filtering LiDAR altimetry data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 61(1):11–22.

Sithole, G. (2005). *Segmentation and Classification of Airborne Laser Scanner Data*. PhD thesis, TU Delft, Publications on Geodesy, 59. Publication of Netherlands Geodetic Commision.

Sithole, G. and Vosselman, G. (2004). Experimental comparison of filter algorithms for bare-earth extraction from airborne laser scanning point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 59(1-2):85–101.

Sotoodeh, S. (2006). Outlier detection in laser scanner point clouds. In *The international archives of the photogrammetry, remote sensing and spatial information sciences*, volume 36, pages 297–302. ISPRS.

Sotoodeh, S. (2007). Hierarchical clustered outlier detection in laser scanner point clouds. 36.

Sun, S. and Salvaggio, C. (2013). Aerial 3d building detection and modeling from airborne LiDAR point clouds. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 6(3):1440–1449.

Szeliski, R. (2011). *Computer Vision*. Springer London.

Teutsch, C., Trostmann, E., and Berndt, D. (2011). A parallel point cloud clustering algorithm for subset segmentation and outlier detection. In Remondino, F. and Shortis, M. R., editors, *Videometrics, Range Imaging, and Applications XI*. SPIE.

Tian, X., Xu, L., Li, X., Jing, L., and Zhao, Y. (2012). A kernel-density-estimation-based outlier detection for airborne LiDAR point clouds. In *2012 IEEE International Conference on Imaging Systems and Techniques Proceedings*. IEEE.

Tomlin, C. D. (1983). A map algebra. In *Proceedings of the 1983 Harvard Computer Graphics Conference*, pages 127–150.

Vo, A.-V., Truong-Hong, L., Laefer, D. F., and Bertolotto, M. (2015). Octree-based region growing for point cloud segmentation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 104:88–100.

Vosselman, G. and Maas, H.-G. (2010). *Airborne and terrestrial laser scanning*. Whittles Publishing.

Wang, C. and Glenn, N. (2009). Integrating LiDAR intensity and elevation data for terrain characterization in a forested area. *IEEE Geoscience and Remote Sensing Letters*, 6(3):463–466.

Worboys, M. and Duckham, M. (2004). *GIS: a Computing Perspective, Second Edition*. CRC Press.

Wu, B., Yu, B., Yue, W., Shu, S., Tan, W., Hu, C., Huang, Y., Wu, J., and Liu, H. (2013). A voxel-based method for automated identification and morphological parameters estimation of individual street trees from mobile laser scanning data. *Remote Sensing*, 5(2):584–611.

Xiong, X., Munoz, D., Bagnell, J. A., and Hebert, M. (2011). 3-d scene analysis via sequenced predictions over points and regions. In *2011 IEEE International Conference on Robotics and Automation*. IEEE.

Zhang, K., Chen, S.-C., Whitman, D., Shyu, M.-L., Yan, J., and Zhang, C. (2003). A progressive morphological filter for removing nonground measurements from airborne LIDAR data. *IEEE Transactions on Geoscience and Remote Sensing*, 41(4):872–882.

Zhuang, Y., Liu, Y., He, G., and Wang, W. (2015). Contextual classification of 3d laser points with conditional random fields in urban environments. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.

A REFLECTION

This section aims to reflect on this study related to the main aspects of the Geomatics for the Built Environment (MSc.) program from Delft University of Technology. Geomatics is the science concerning (1) data acquisition, (2) data management, (3) data analysis, (4) data visualization, and (5) data quality. This graduation project shows knowledge of all five aspects.

Data acquisition is done with LiDAR. Although, all data is collected and provided by third parties, specifications and details are explained in § 2.1.

Data storage-/management is structured by using `.las` and `.laz` files. Properties are given in § 2.1.2. Julia scripts are used to read and write the data. Moreover, an alternative way of reading the data for better memory requirements is discussed in § 6.1.2 Streaming. Finally, different data models are discussed in § 2.3.

Data analysis covers the largest part of this thesis and is discussed theoretically (§ 2.4) as well as implemented in a workflow (Chapter 4 and Chapter 5).

Data visualization is an important part, because of the visual aspects of point cloud data. Therefore, visualizations are present in many steps.

Finally, data quality is assessed in two ways. First, the input data is discussed in § 5.2. Secondly, an assessment of the processed point clouds is given in § 5.4.

COLOPHON

This document was typeset using LaTeX. The document layout was generated using the `arsclassica` package by Lorenzo Pantieri, which is an adaption of the original `classicthesis` package from André Miede.