Inferring High Resolution Terrain, Vegetation, and Lines of Sight Models from Point Cloud Data

BY

BENEDETTO VITALE
B.S, Politecnico di Milano, Milan, Italy, 2014

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2016

Chicago, Illinois

Defense Committee:

Tanya Berger-Wolf, Chair and Advisor

Brian Ziebart

Pier Luca Lanzi, Politecnico di Milano

To my lovely family...

Without you none of this would have been possible.

ACKNOWLEDGMENTS

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ANOVA   Analysis of variance

ANN   Artificial Neural Networks

BST   Binary Search Tree

CHM   Canopy Height Model

LIDAR   Laser Imaging Detection and Ranging

KDT   K-Dimensional Tree

KNN   K-Nearest Neighbors

TIN   Triangular Irregular Network

TP   True Positive

TN   True Negative

FP   False Positive

FN   False Negative

LOOCV   Leave One Out Cross-Validation

SVM   Support Vector Machine

TPS   Thin-Plate Spline

UTM   Universal Transverse Mercator

SUMMARY

With the diffusion of technologies which enables the acquisition of the 3-Dimensional (3D) structure of different surfaces, there has been a great focus on the information brought and the uses that are possible with this kind of data. One of the field in which this technology has been widely used is ecology, given that 3D data offers the possibility of analyzing trees and their characteristics directly from the point cloud structure by means of techniques aimed at isolating ground points from non-ground ones, procedures with the goal of segmenting individual trees in the cloud and analysis with respect to the trees point structure obtained in order to derive the trees properties.

In this thesis we proposed a framework which infers information on vegetation and terrain starting from raw point cloud data of forested environments, classifying first the points in the point cloud as ground and non-ground by means of a multiscale curvature algorithm proposed in the literature, then segmenting individual trees among the vegetation points by means of a new segmentation algorithm proposed. By leveraging on the information extracted for terrain and vegetation, the framework infers also the visual obstruction potential of the identified vegetation elements by means of a supervised inference approach. The first phase of the supervised inference consists in extracting a suitable set of features from the vegetation points structures, in which we tried to extract features both through a manual approach, meaning that we chose a specific set of features to extract, and through neural networks in an automatic way. In the second phase instead, different machine learning algorithms were used in order to

SUMMARY (continued)

infer the visual obstruction potential of each vegetation element using the features extracted in the previous phase. We also provided a practical example among the possible uses of the information extracted by the framework, which consists in exploiting the framework results to analyze lines of sight among the individuals located in the environment.

Another possible use of the information extracted by the framework could providing support in generating an immersive experience in forested environments, experiencing the habitat exactly as animals living in it. The reason behind the generation of an immersive environment is that interactions among animals of the same or different specie significantly influence their behavior under many valuable aspects, thus being able to track and analyze their movements and actions in a detailed and immersive way would bring useful insight to animal behavior studies.

CHAPTER 1

INTRODUCTION

Thanks to the availability of different technologies which enabled the extraction of the 3D point cloud structure from captured surfaces, there has been a great focus on what can be accomplished with 3D data in various fields. For example, point cloud data of forested surfaces has been used to estimate to estimate fruit-tree leaf area ([1]) or to generate a vine plantation map ([2]). Two are the main technologies used to obtain the 3D point cloud information relative to a certain scanned surface:

- **Laser Imaging Detection and Ranging (LIDAR)** is "an active remote sensing technology that measures properties of reflected light to determine range to a distant object" ([3], [4]). "The range to an object is computed by measuring the time delay between the transmission of a laser pulse and the detection of the reflected signal" ([3], [5]). Thanks to the possibility of generating a 3D structure with high resolution and accuracy, LIDAR has been widely used in fields such as ecology ([1]), geomorphology ([2], [6]) and remote sensing ([7]).

- **Aerial Photography** consists of using flying technologies such as drones to capture images of surfaces from different angles and at different heights. Thanks to the fact that each image covers a different height or angle of the surfaces, they can be assembled in

order to obtain a detailed 3D point cloud structure of the captured surfaces, as successfully done in [8]).

There are many advantages in analyzing the 3D structure of a certain scanned surface, which could be object identifications according to spatial features ([9]) or extracting charcteristics of vegetation elements in forested environments once obtained their point structure ([10]). Another useful information which could be extracted from 3D point cloud data corresponding to forested environment derive from the analysis of the lines of sight among individuals located in the environment and of the interactions among them.

Visibility information is a crucial aspect in forested environments, especially in fields such as animal behavior, given the fact that interactions among animals of the same or different specie significantly influence their general behavior. Given the importance of visibility, using this information to generate an immersive environment would offer a great support to the community of scientists focused on animal behavior. It is quite hard to track animals in some natural environments due to their continuous movements and the fact that they often reach places where they are not easily observable, as with monkeys on very high trees. Having the possibility of experiencing forested environments exactly as animals do during their movements and everyday life would help significantly in understanding which are the causes that make animals behave or react in a specific way in their natural habitat. The objective of this thesis raises many technical challenges however, considering that plenty of information must be extracted from raw point cloud data.

1.1   <u>Thesis Objectives</u>

In this thesis we used the 3D point cloud data to provide support in generating an immersive 3D experience for forest surfaces by means of a framework extracting various information starting from raw point cloud data. We first infer information on the terrain by determining which points belong to the ground elements and which to the non-ground ones. Then we infer information about vegetation by detecting individual vegetation elements among the non-ground points identified. The proposed framework can be used for any ecological study, including those that focus solely on the vegetation (such as following the blooming and growth of trees over time). We showcase the use of this habitat information by applying it to compute visibility information from an animal perspective placed in that landscape. If animal position information is available, we can compute mutual visibility of animals in the habitat, given that their main obstructing features are on the scale of the features we can extract from the point cloud data.

At a high level, there are four main steps that must be performed in order to extract the information necessary to reach the final objective of this thesis:

1. **Ground/non-ground separation:** the point cloud data must be analyzed in order to identify points corresponding to the ground and points corresponding to not ground elements, such as vegetation. This step is crucial, given that detecting the right vegetation is crucial for the functioning of the whole framework and for the correctness of the produced results.

2. **Tree segmentation:** points corresponding to vegetation must be segmented in individual vegetation elements. In doing so, it must be considered that a forest can present a huge variety of vegetation elements, thus it is necessary to perform a parameter space exploration to determine parameter settings that capture desired classes of vegetation most accurately.

3. **Visibility/obstruction characterization of vegetation:** points corresponding to vegetation elements must be analyzed in order to extract a suitable set of characteristics which would allow to infer the visual obstruction potential of the vegetation element to which they correspond. Once those features are extracted, various machine learning classification techniques can be applied in order to infer the visual obstruction potential.

4. **Lines of sight analysis:** Given the location of two individuals, we must check whether some vegetation elements able to obstruct their lines of sight are located between those individuals. We use the results of the obstruction characterization of vegetation from previous step to infer whether two individuals can see each other.

For some of these technical challenges, different approaches had already been proposed in the literature and we will have a deeper look at them in Chapter 2, highlighting advantages and limitations of the existing techniques.

For what concerns the structure of the thesis, we now describe the focus of each chapter: Chapter 2 provides a review of the most recent solutions proposed in the areas of interest covered by our framework. Chapter 3 covers the background topics necessary to understand the work performed in this thesis. In Chapter 4 we describe the data used to perform the experi-

ments in this thesis, their acquisition procedure and the information they provide. Chapter 5 shows in details the first step performed starting from the raw point cloud data, which is the ground/non-ground classification of points in the cloud. In Chapter 6 we explain in a detailed way the segmentation technique proposed in this thesis, the underlying assumptions and the results obtained. Chapter 7 covers the features extraction procedures used and the experiments performed in order to predict the visual obstruction potential of the vegetation elements in the environment. Chapter 8 presents the method used to determine whether an object with visual obstruction potential is standing in the line of sight of two individuals located in the forest environment, thus obstructing their visual. In Chapter 9 we make some conclusions and reasoning on the presented methods, showing also potential applications and variants of the method and potential future developments.

CHAPTER 2

STATE OF THE ART

In this Chapter we survey the existing work in the literature relevant to the objectives of habitat feature extraction from point cloud data and visibility calculations. Before proceeding, given that in our case the vegetation elements are characterized by trees, we will use the term "tree" instead of vegetation element in order to refer to vegetation.

2.1   Ground/Non-Ground Separation Approaches

When analyzing point cloud data, knowing which points correspond to ground elements and which to non-ground elements is fundamental in order to properly process data with the aim of extracting meaningful information. Given the importance of this operation, in the literature it is possible to find many different approaches aimed at properly performing this separation ([11], [12], [13]). In [13] in particular the authors performed the ground/non-ground separation by using active shape models for the ground, which then were matched to the point cloud surface. However, estimating proper shapes for the ground is time consuming and requires prior knowledge of the environment. In order to avoid estimating shapes, for this thesis we used the *multiscale curvature algorithm for Point Cloud Data* proposed in [14], which is described in a detailed way in Chapter 5 and does not require any prior knowledge about the environment ground shapes.

2.2   Tree Segmentation Approaches

Before the availability of laser scanning technologies, a typical method to extract 3D informa-
tion of forested environments and capturing individual trees characteristics was field inventory,
as done in [15]. However, field inventory can be really time consuming and limited by spatial
accessibility, given that not all study areas can be accessed so easily. With the development of
the 3D point cloud extraction technologies, various approaches making use of the 3D structure
have been proposed, which can be mostly grouped into two categories: the Canopy Height
Model (CHM) based algorithms and the raw point cloud based ones.

2.2.1   CHM Based Algorithms

CHM-based approaches have in common the use of the point cloud derived CHM, which is
a "raster image interpolated from the point cloud depicting the top of the vegetation canopy"
([3]). While the CHM model brings a valuable amount of additional information, it may be
affected by inherent errors caused by a various number of sources. For instance, it was shown
that "a spatial error can be introduced during the interpolation process from the point cloud
to the gridded height model, which can decrease the accuracy of tree segmentations and of the
relevant measurements" ([3]). Based on the CHM, various methods were proposed:
In [16] the authors proposed a tree segmentation algorithm with two main steps. First they
adopted a local maxima detection approach to find the tree tops. Then, they used a region
growing procedure to highlight the respective tree crowns, obtaining at the end the full trees.
In [17] instead, the authors developed a multi-scale template matching approach in order to
find individual trees in the environment. More in details, they adopted elliptical and other

shaped templates in order to identify the points corresponding to a specific tree. However, this requires a priori knowledge of the tree shapes characterizing the forest at hand.

2.2.2   Raw Point Cloud Based Algorithms

In order to overcome the inherent errors related to the previous kind of approaches, algorithms operating directly on the raw point cloud data were proposed, making use only of the raw information provided by the point cloud elements.

In [18] the authors applied a clustering algorithm directly to the raw coordinates triple of the point cloud data (the raw coordinate of the three axis x, y, z). The clustering algorithm chosen was the *K-means* ([19]). However, using the *K-means* implies that the results obtained will always be subject to the initialization points chosen for the clusters, which were estimated from the local maxima of a rasterized digital surface model, thus making use of more than just the raw point cloud data. In addition, it must be known which is the number of trees in the forest in order to properly choose the k parameter. In [20] instead, the authors developed a method based on two main steps. During the region growing step they identified subparts of the trees canopies, which then they merged into a full tree canopy by means of an agglomerative clustering approach. The most challenging part in such approach is the choice of the search radius R used to identify the treetops, which must be estimated by means of a trial and error approach. Another example of raw approach is [21], where the authors used an algorithm which identifies one tree at a time. In this method, they first found the point with the highest height in order to consider it as a tree top, then they proceeded with decreasing height to identify the tree points by means of a minimum spacing rule and a spacing threshold. Once a tree is found,

its points are removed by the initial point cloud. However, running such a method for a huge amount of points (typical of point cloud data) would be really time consuming, given that it is necessary to scan all the remaining points in the cloud in order to identify each tree.

In this thesis we used an approach similar to the one presented in [21] to perform the tree segmentation, but with different adopted criteria and thresholds in order adapt the procedure to our setting.

2.3   Visual Obstruction Inference

After having identified the individual trees in the point cloud, a considerable amount of information could be extracted by analyzing the point structure of the trees. In the literature it is possible to find only more general approaches aimed either at identifying objects in the point cloud or inferring trees properties:

In [9] the authors tried to classify object in the point cloud into roads, roof, grasses and trees by means of supervised parametric classification techniques, meaning that they already knew which was the set of object types in the point cloud. Another example is [10], where the authors tried to identify the tree species in the forest relative to the point cloud using the raw points and a rasterized image of the surface, where all the species present in the forest were already known. To the best of our knowledge, there are no studies in which the authors tried to infer visual obstruction potential or other visual aspects of trees, for what concerns the generation of immersive experiences in the literature it is possible to find only one attempt in which the author generated "an immersive tour experience system for cultural sites", presented in [22].

CHAPTER 3

BACKGROUND

Each implemented step of the framework relies on various data structure, procedures and basic concepts combined with the aim of extracting more information from the input data. In order to offer a complete and clear understanding of the whole process, in this chapter we will go through the concepts and methods on which the work done in this thesis is based and further developed.

3.1 Tree Data Structure

A tree is defined as a data structure composed by nodes and edges, not containing any cycle.

A non empty tree is made of a root node and many depth levels containing other nodes that all together constitutes a hierarchy. In this thesis we focus our attention on just two kind of trees data structure, used in order to reach the final objective.

3.1.1 Binary Search Tree (BST)

The Binary Search Tree is a tree-like data structure which respects the property that the key in each node must be greater than all the ones stored in the sub tree on the left and smaller than all the keys in one on the right ([23]). The major advantage of this data structure is that sorting and searching algorithms can be implemented in a easy and efficient way thanks to the specific ordering of the elements in the tree. In Figure 1 we report an example of BST.
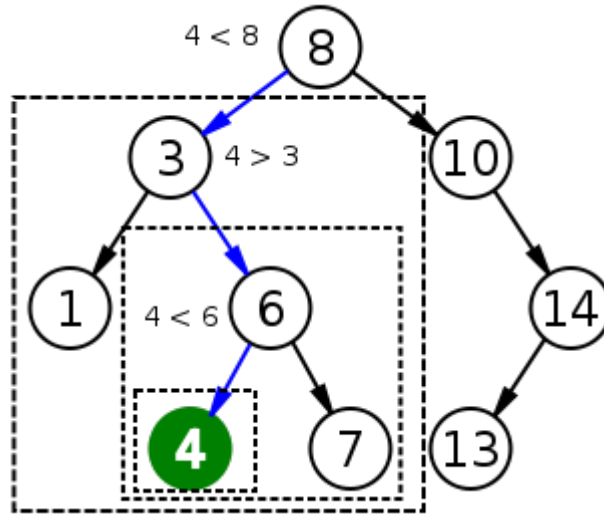
Figure 1: An example of BST. [1]

3.1.2 K-Dimensional Tree (KDT)

At a high level, a KDT is a generalization of a BST that stores points in k-dimensional space ([24]). An example of KDT is reported in Figure 2.

As we can see, a KDT is a k-dimensional binary search tree, where each point defines an hyperplane dividing the k-dimensional space in two parts, where the left part is represented by the left subtree and right part from the right one. The greatest advantage of this separation is given by the speed up obtained in the nearest neighbors lookup, which is performed through the following procedure:

---

[1]https://commons.wikimedia.org/wiki/File:Binary_search_tree_search_4.svg.
Reprinted with permission, see Appendix.
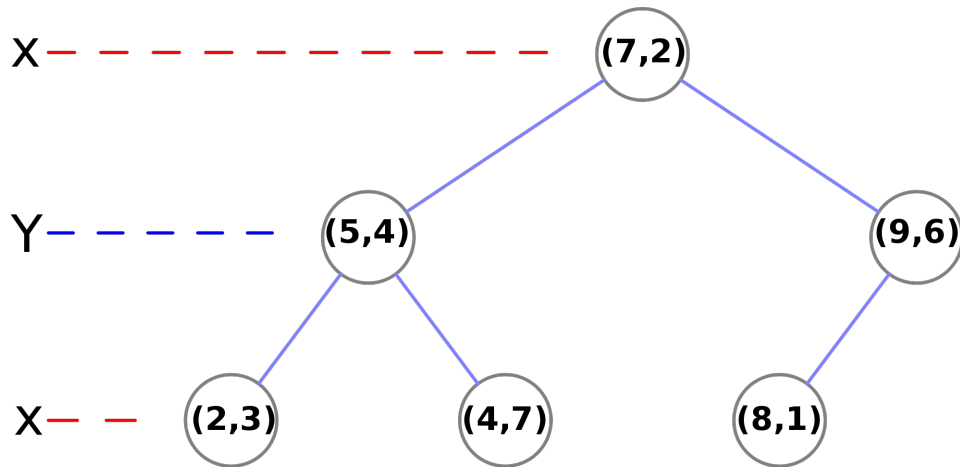
Figure 2: An example of KDT. [1]

I) Take a guess of where the nearest neighbor is located by taking all the points inside a certain hypersphere of radius r.

II) Determine if the candidate hypersphere crosses one of the splitting hyperplanes by checking if $|b_i - a_i| < r$.

III) If the hypersphere is just on one side of the hyperplane, then we need to check only that side, otherwise recursively search both sides of subtrees corresponding to the two parts of the hyperplane.

The speed up is mostly given by the fact that we are guessing where the nearest neighbor is and that our hyperplanes are all axis-aligned, making checking if they an hypersphere crosses

---

[1]https://commons.wikimedia.org/wiki/File:Tree_0001.svg.
Reprinted with permission, see Appendix.

them or not a feasible task, thus guiding towards the right place where to search in a fast and simple way. In this thesis we used the KDT structure in order to perform fast lookup of a point neighbors in the point cloud.

3.2   Kernel Methods

In machine learning kernel methods are techniques which allow to perform a mapping between two different feature spaces without explicitly computing the spaces, thus offering a great computational advantage ([25]).

Kernel methods are mostly used in pattern analysis. The general aim of pattern analysis is to determine a function that relates the input data with the output ones. The great advantage in using them is that it not required to transform the data in raw representation into a feature vector representation, given a specific mapping defined by the user. Instead it suffices to define a user-specified kernel, which is a similarity function over pairs of data instances in their raw representation. In the next sections we will go into details of the kernel methods used in this thesis.

3.2.1   Density Kernel

Let $x_1, ..., x_n$ be an identically independently distributed (i.i.d.) sample taken from a distribution with unknown density f. The objective of the density kernel is estimating the shape of f by using the following kernel density estimator ([26]):

$$\hat{f}_h = \frac{1}{n} \sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - x_i}{h}), \tag{3.1}$$

where K is a kernel function, which is non negative, integrates to one and has zero mean. The h parameter is a smoothing parameter, also known as *bandwidth*, which must be greater than zero. $K_h(x) = \frac{1}{h}K(\frac{x}{h})$ is called scaled kernel with subscript h.

In this thesis we used the density kernel to estimate the density of the points corresponding to a specific tree.

3.2.2   Linear Kernel

In mathematics a linear kernel is defined as a linear mapping between two vector spaces ([25]). Formally, let V and W be two different vector spaces, L be a linear mapping between V and W and v be an element of V, then a linear kernel is defined in :

$$ker(L) = \{v \in V | L(v) = 0\} \tag{3.2}$$

3.2.3   Polynomial Kernel

The polynomial kernel is used to represent the similarity of vectors over polynomials, which are derived from the original variables, thus allowing the use of non-linear models ([11]). Given a degree d polynomial and two input vectors x and y in their original feature space, the polynomial kernel is defined as:

$$K(x, y) = (x^T y + c)^d, \tag{3.3}$$

where c is a free-parameter used to balance the influence of higher-order terms with respect to lower-order ones in the polynomial.

3.2.4 Averaging Kernel

The averaging kernel is a $N \times N$ sliding-window spatial filter, which assigns to the central value of the $N \times N$ matrix in input the average of all points in the matrix ([27]). The matrix in input is referred to as *window* or *kernel*.

3.3 Learning Methods

In machine learning, with the term learning we are referring to the procedure used to infer new information from what it is already known about data. Learning can be achieved in two main ways, known as *supervised learning* and *unsupervised learning*.

Supervised learning is defined as the procedure used to learn a function f such that given an input x, corresponding to a set of characteristics of a certain element, the function is able to infer from x the unknown class y to which the the element belongs, meaning that $f(x) = y$. Unsupervised learning instead consists in looking for meaningful patterns in the data without having any prior knowledge about the class, thus it is more exploration than inference oriented.

In the next section we will see more in details the supervised and unsupervised learning methods, given the fact that they have both been used in this thesis.

3.3.1 Classification

The most common supervised learning method is classification, which consists in training a classifier on some labeled data (for which the belonging classes of data instances are known) in such a way that, when tested on unseen data, the classifier will predict the right label for those data, given their features ([28]). A feature is defined as a measurable property of a certain observed phenomenon. The process consists in two main phases: training and testing.

During training, a certain amount of labeled data is used to learn useful patterns with respect to the class label and the features. During testing instead, the algorithm is used to infer the label of previously unseen data by means of the pattern detected during the training phase. Generally, the labels of data used to test the algorithm are known, thus procedures computing error measures by comparing the label inferred by the algorithm and the real one could be easily applied. Data used to make our classifier learn interesting patterns are called *training set*, while the ones used to test our classifier and verify whether the predictions obtained are correct or not are called *testing set*. A necessary condition is that training and testing set must be *i.i.d.*, given that any kind of correlation between the two would lead to biased predictions.

In this thesis we applied this approach to infer the visual obstruction potential of vegetation elements in the point cloud. In the next sections we will go into details of the supervised classification algorithms we used to perform inference.

3.3.1.1 Decision Trees

A decision tree is a tree-like structure in which internal nodes represents a check on the value of a certain feature (a certain characteristic of an element), each branch represents one of the outcomes of the previous check (what we wanted to verify) and each leaf node represents a specific class (final value assigned to the specific element) ([28]). Each paths from root to leaf is a classification rule, which basically corresponds to a set of values over the checks performed at each internal node which guide toward a specific class. The tree structure is learned maximizing the *information gain* at every possible split.

Formally, let $T = \{x_1, ..., x_n, y\}$ be a set of training examples where x_i is the i-th feature and y is the class label, then, for a certain feature i, information gain is defined as:

$$I(T, i) = H(T) - \sum_{v \in vals(i)} \frac{|\{x \in T | x_i = v\}|}{|T|} \cdot H(\{|\{x \in T | x_i = v\}), \qquad (3.4)$$

where H represents the *entropy*, which, given a discrete random variable X with values $x_1, ..., x_n$ and probability mass function $P(X)$, is defined as:

$$H(T) = -\sum_{i=1}^{n} P(x_i) log_2 P(x_i) \qquad (3.5)$$

3.3.1.2 K-Nearest Neighbors (KNN)

KNN is a memory based supervised classification algorithm, meaning that previously unseen data are classified with the class of the least distant element or elements in the training set, according to the k chosen ([28]). In order to make the algorithm work properly it is necessary to define a suitable distance measure between data instances. The k parameter is used to determine how many neighbors must be considered in distance order. Once identified the k closest neighbors, the final prediction is obtained by performing majority voting on the classes of found neighbors. In case of a tie, a random decision is made. It is immediate to notice that the choice of the k parameter and the distance measure are the crucial factors in such approach.

3.3.1.3 Naive Bayes

The naive bayes classifier is a supervised classification method using probabilities to perform classification, based on the assumption that all the features in the data are independent from

the class ([28]). By exploiting to this assumption, the conditional probability of a training instance $x_1, ..., x_n, y$, where $x_i \in X_j$ is a value of the j-th feature and $y \in Y$ is the class label, could be rewritten in the following way:

$$P(X_1 = x_1, ..., X_n = x_n | Y = y) = P(X_1 = x_1 | Y = y) \cdot ... \cdot P(X_n = x_n | Y = y) \qquad (3.6)$$

During the training phase all the conditional probabilities of the features values conditioned by every class label are computed from the training set. During the test phase instead, given a previously unseen instance $x_1, ..., x_n$, the predicted class label is the $\hat{y} \in Y$ that maximizes the following conditional probability $P(X_1 = x_1, ..., X_n = x_n | Y = \hat{y})$ computed using the probabilities estimated during the training phase.

3.3.1.4   Logistic Regression

Logistic regression is a statistical model that measures the relationship between the class label and one or more independent features by estimating probabilities by means of the *logit function* ([28]). Thus, given a previously unseen instance $x_1, ..., x_n$ where $x_i \in X_j$ represents a value of the j-th feature, the probability of a certain class label is estimated in the following way:

$$P(Y = y) = \frac{e^{w_0 + w_1 \cdot x_1 + ... + w_n \cdot x_n}}{1 + e^{w_0 + w_1 \cdot x_1 + ... + w_n \cdot x_n}}, \qquad (3.7)$$

where $w_0, w_1, ..., w_n$ are the features weight estimated during the training phase by minimizing a logarithmic cost function that penalizes in a logarithmic way each wrong prediction on the training set.

3.3.1.5 Random Forest

Random forest is an supervised classification ensemble method ([28]). The term *ensemble* means that we train the same classifier a certain number of times, each time with different features, and then combine their outcome in some specific way in order to obtain the final prediction.

More in details, the random forest method consists in training an high number of decision trees, each one having as features a random selection of all the ones available in the training set.

At testing time the final prediction is obtained by performing majority voting on all the outcomes of the built trees for the specific features' values of the tested data instance.

3.3.1.6 Support Vector Machine (SVM)

SVM is a supervised learning model which locate the training instances into an high dimensional feature space, meaning that each feature becomes a dimension, and search for the best splitting hyperplane ([28]). A best splitting hyperplane is defined as an hyperplane which separates the instances belonging to one class from the ones belonging to others classes with the best possible margin, meaning that the distance of points belonging to a certain class from the hyperplane is maximum. When the number of dimension is too high, SVM exploits the *kernel trick* in order to avoid computing directly the new feature space in order to lower the

number of dimensions. In order to avoid the explicit computation phase, the vectorial product between two elements is computed instead of the whole space transformation.

3.3.2 The Clustering Method

The most common unsupervised learning method instead is clustering, consisting in grouping elements into clusters, where with the term cluster we are referring to a group of elements which are really similar to each other, according to a chosen similarity measure ([29]). Elements belonging to different clusters instead should be really different from each other, always according to the same chosen similarity. The crucial aspect in such procedure is determining the best similarity measure with respect to data and objectives.

In this thesis we have used an approach similar to clustering in order to identify the points in the point cloud corresponding to a specific tree, leveraging on the fact that distances between point belonging to the same tree is generally different from distance between points belonging to different trees.

3.4 Point Cloud Representation

The analysis on the forested surfaces used in this thesis were performed on the point cloud representation of the surfaces. A point cloud representation is a collection of points bringing 3D information relative to an object or a surface ([30]). It is a very basic discrete representation, essentially specifying the geometry of the object by means of sampling procedures at certain positions. In the geographic field they are mostly used in order to create digital elevation models of the terrain.

3.5 Probability Distributions

Many of the methods already seen leverage on the concept of probability distributions. Probability distributions are used to assign a certain probability value to each subset of a chosen random experiment ([31]). There two main types of probability distributions: *Discrete* probability distributions, where it it possible to assign a probability to each possible outcome of the experiment, and *continuous* probability distributions, where random variables takes value from a continuum, thus probabilities are assigned to intervals and individual values. Two are the main continuous distributions used in this thesis, which we are going to describe in the next sections.

3.5.1 Gaussian Distribution

The Gaussian distribution is one of the main probability distribution widely used in various fields, given that a set of random variables can be approximated to a Gaussian distribution under specific circumstances ([31]):

- The random variables must be i.i.d.

- They must have a well-defined mean

- They must have a well-defined variance

The Gaussian distribution is defined by the following probability density function :

$$f(x|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$ (3.8)

where μ is the mean of the distribution and σ the variance.

3.5.2   Student t-distribution

The Student's t-distribution is a member of the family of continuous probability distributions, which are used in order to estimate the mean of a normally distributed population (it follows a Gaussian distribution) with small sample size and unknown standard deviation ([31]).

Now, let v be the degrees of freedom of the distribution, corresponding to the number of variables in it, then the Student t-distribution is defined by the following probability density function:

$$f(t) = \frac{\Gamma(\frac{v+1}{2})}{\sqrt{v\pi}\Gamma(\frac{v}{2})}(1 + \frac{t^2}{v})^{-\frac{v+1}{2}}, \tag{3.9}$$

where $\Gamma(t)$ is the *Gamma Function*, defined as

$$\Gamma(t) = \int_0^\infty x^{t-1}e^{-}xdx \tag{3.10}$$

3.6   Statistical Significance Tests

Once results of a prediction are obtained, statistical significance tests are performed in order to verify whether the results obtained are significant or not. Statistical significance tests consist in checking if the *p-value* is less than a chosen significance level α ([31]). The *p-value* is defined as the probability of obtaining at least as extreme results as the one observed, given that the *null hypothesis* is true. The null hypothesis is defined as the default condition according to which there is no relationships between the samples that we are considering. The significance level instead is defined as the probability of rejecting the null hypothesis given that it is true.

When the *p-value* is less than α, we can say that our samples are statistically significant at a $1 - \alpha$ level. Statistical test are widely used in order to verify if the improvement in the prediction results obtained is significant with respect to to another result obtained using a different prediction technique, based on the fact that comparisons among different prediction results are meaningful only if the relative samples are significant. For what concerns the tests used in this thesis, we used the *Student t-test* and the *The Analysis of Variance* (ANOVA) test, explained in the following sections.

3.6.1 Student t-test

The Student t-test is a statistical test used to check hypothesis where the test samples are defined by a Student's t-distribution, given that the null hypothesis is satisfied ([31]). This test can be used to determine whether two samples are significantly different from each other given their means.

3.6.2 The ANOVA Test

The ANOVA test is a statistical significance test used to analyze the differences among group means and variances ([31]). More in details, the ANOVA test performs a statistical test with the aim of checking if the means of several groups are equal, thus it generalizes the t-test to more than just two groups.

3.7 Validation Metrics

Validation of results obtained through a classification process is a very important aspect, given that we want to be as sure as possible that the predictions made are reasonable and

correct. There are many techniques that can be used to perform validation, but before going into details we first introduce the nomenclature necessary to understand their definition ([28]):

1. **TP:** are instances which are predicted as positive (class represented as 1 in binary classification) according to the truth label and the classification outcome.

2. **TN:** are instances which are predicted as negative (class represented as 0 in binary classification) according to the truth label and the classification outcome.

3. **FP:** are instances which are predicted as positive by the classifier but they are negative according to the truth label.

4. **FN:** are instances which are predicted as negative by the classifier but they are positive according to the truth label.

These definitions are intuitively represented in the matrix reported in Figure 3, known as *confusion matrix*.

In the next sections we will define the most common validation measures and techniques used in the field.

3.7.1 <u>Accuracy</u>

According to ([28]), accuracy is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.11}$$

Accuracy measures the percentage of correctly classified instances with respect to all the instances in the test set.

prediction outcome

Figure 3: Confusion matrix.

3.7.2 Recall

According to ([28]), recall is defined as:

$$Recall = \frac{TP}{TP + FN} \tag{3.12}$$

Recall measures the percentage of the positively instances correctly classified with respect to all the positive instances in the test set.

3.7.3 Precision

According to ([28]), precision is defined as:

$$Precision = \frac{TP}{TP + FP} \tag{3.13}$$

Precision measures the percentage of the positively instances correctly classified with respect to all instances classified as positive.

3.7.4 Cross-Validation

Cross-validations is a model validation technique with the aim of testing how the results of a certain prediction will generalize with respect to an independent dataset ([28]).

The technique consist in dividing the data available in a certain number of folds, all stratified (with the same number of positive and negative instances). Once created the folds, the model is trained on all the fold except one, which will be used as test data. This is done until all folds have been tested. The final error is given by the average of the errors computed on each fold. When each fold contains only one instance, then we are performing a validation technique called *leave one out cross-validation* (LOOCV), given that we train our model on all the instances available except one. In LOOCV the error is computed as the number of correct predictions on all the folds divided by the number of folds, which corresponds to the size of the dataset.

3.8 Universal Transverse of Mercator (UTM) Coordinates System

In this section we provide the description of the UTM geographical coordinate systems, given that the locations of points in the point cloud are expressed using the UTM system.

The UTM coordinate system is a system commonly used to assign geographical coordinates to elements on Earth ([32]). It uses two coordinates to give locations on the surface of the Earth: the first coordinate is called *Easting* and the second one *Northing*. As with latitude and longitude, it is a horizontal position representation, meaning that the location on Earth is identified without considering any vertical component representing the vertical position of the

point. However, the UTM system cannot be considered a single map projection, given that it divides Earth in sixty zones, where each zone corresponds six-degree band of longitude.

3.9   Interpolation

Given that in this thesis we used the interpolation procedure to generate a new point cloud surface starting from the original one when performing the ground/non-ground separation of points, we now define the relative concepts. When we use the term interpolation we are referring to the method of generating new data points in the range of a already available set of discrete data points [33]). An example of the interpolation procedure applied on mathematical functions is reported in Figure 4. For what concerns the work done in this thesis, we are mostly interested in the *Thin Plate Spline* (TPS) interpolation, presented in the next section.
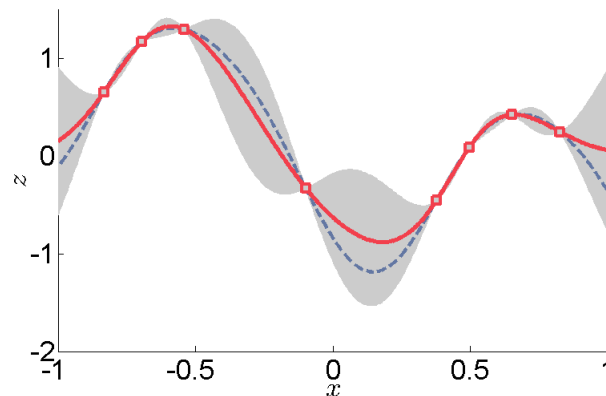
Figure 4: An example of interpolation of a curve (blue dashed) generating an approximate one (red). [1]

28

3.9.1   TPS Interpolation

TPS interpolation is an interpolation technique that, given a set of data points, performs interpolation by means of a weighted combination of thin plate splines, which are numeric functions that are defined piecewise by polynomial functions ([34]). Each spline is centered about each data point and generates the interpolation function passing through the points while at the same time minimizing the following integral:

$$I[f(x,y)] = \int\int_{R^2} (f_{xx}^2 + 2f_{xy}^2 + f_{yy}^2)dxdy, \tag{3.14}$$

where R^2 denotes the two dimensional space and f_{ab} denotes the second derivative computed first with respect to a and then b. The above integral is known with the name of *bending energy*.

3.10   Artificial Neural Networks (ANN)

In this section we go into details of ANN, given the fact that they were used in this thesis to perform features extraction from the trees point structure in an automatic way. In machine learning ANN are models reproducing the biological neural networks and are used to approximate functions depending on generally a huge number of inputs ([35]). An ANN can be depicted as a directed graphic model in which the node of the graph are called *artificial neurons*, which exchange messages with other neurons to which they are connected in the graph.

---

[1]https://en.wikipedia.org/wiki/File:Example_of_kriging_interpolation_in_1D.png. Reprinted with permission, see Appendix.

3.10.1 <u>Artificial Neuron</u>

An artificial neuron is a mathematical function built to model biological neurons. They receive one or more inputs (representing dendrites) and sum them to produce an output (representing a neuron's axon) ([35]). The contribution of each node is weighted and the sum is propagated by means of a non-linear function, known with the term of *activation function*. The first activation function used was sigmoid shaped (output between 0 and 1, defined for all real input values and has a positive derivative at each point), while in the last years non-linear functions have been widely used, such as the *hyperbolic tangent* (antisymmetric function with output between 1 and -1 and defined everywhere on the x-axis). The shape and formula of the sigmoid function are reported in Figure 5, while of the hyperbolic tangent in Figure 6. Activation functions must be monotonically increasing, continuous and preferably differentiable in order to more easily compute the gradient.

Formalizing the definition of an artificial neuron, the output y is obtained through the following formula:

$$y = \phi(\sum_{j=0}^{m} w_j x_j + b), \tag{3.15}$$

where ϕ denotes the activation function, x_j and w_j the j-th input and its weight respectively and b the bias of the specific neuron. In Figure 7 we report the whole structure of an artificial neuron.
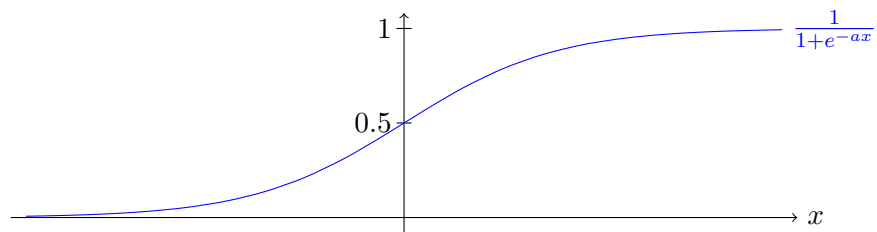
Figure 5: Sigmoid function.
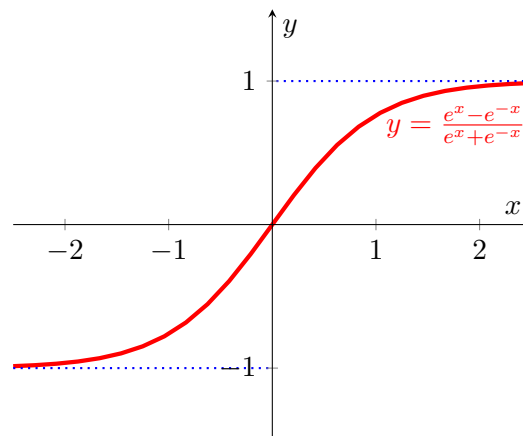
Figure 6: Hyperbolic tangent function.
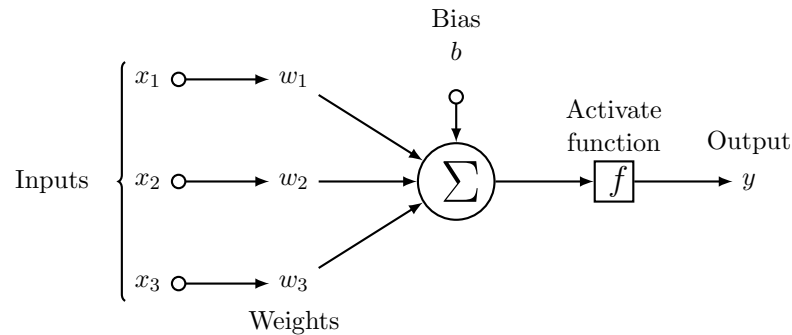
Figure 7: The structure of an artificial neuron.

3.10.2 Structure

Each connection between different neurons has a numeric weight, which is tuned according to experience, thus making neural networks a model adaptive to inputs and suitable for learning.

Generally neurons are grouped in *layers*, which are level of the network corresponding to a certain number of neurons. The first layer is called *input layer*, the ones in the middle are called *hidden layers*, given that we don't really see the inputs passed to them due to their internal position in the network, while the last one is called *output layer* ([35]). In Figure 8 we report an example of ANN with one hidden layer, five neurons in the input layer, three in the hidden layer and one in the output layer.

ANN propagating input in successive layers are known as *feed-forward ANN*.
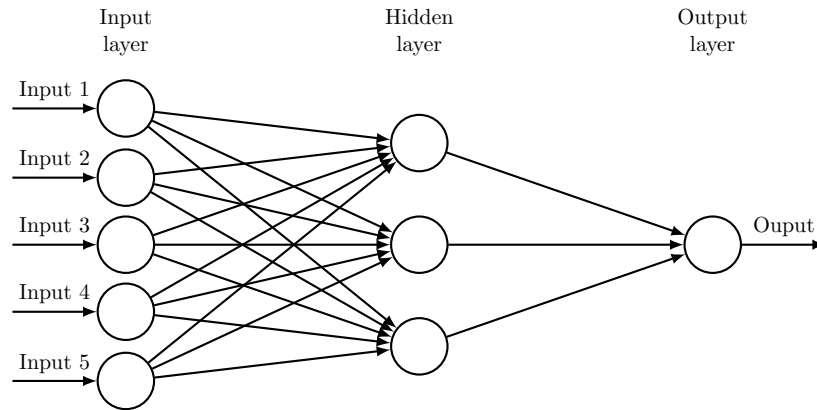
Figure 8: An example of ANN with one hidden layer.

3.10.3 Learning Phase

During the learning phase, the algorithm most commonly used is the *backpropagation algorithm* ([36]). The algorithm consists of two phases, which we are going to describe in details through the steps performed at each one:

- **Propagation:**

  1. Forward propagate the inputs in order to activate the successive layers until we obtain the output.

  2. Backward propagate the gradient of the cost function with respect to parameters in the network until reaching the input layer.

- **Weight Update:** for each connections the following steps are performed:

1. Multiply the gradient computed during the propagation phase for a certain percentage.

2. Update the weight of the connection by subtracting to the old value of the one computed in the previous step.

The two phases are performed until we reach a certain desired error or for a certain number of cycles, also known as *epochs*, while the percentage of the gradient considered is called *learning rate*. The last greatly influences the speed and quality of learning, given that the greater is the ratio, then the faster the neurons are trained, while the lower is the ratio, the slower is the training.

3.10.4   Autoencoder

We now introduce a particular ANN of interest with respect to the feature extraction objective of this thesis. Autoencoders are mostly used to perform a *feature learning* task, which is the case of the third step performed in this thesis. With the term *feature learning* we are referring to the procedure in which we extract from raw input data interesting properties that could be used in a machine learning task ([37]).

The difference between autoencoders and usual multilayer feedforward neural networks resides in the fact that autoencoders are trained to reconstruct the received input and not to perform a prediction task. The structure of an autoencoder is made of two main parts, *the encoder* and *the decoder*, which could be seen as two transition functions ϕ and υ, defined in the following way:

$$\phi: \quad X \to F \tag{3.16}$$

$$\upsilon: \quad F \to X \tag{3.17}$$

$$argmin_{\phi,\upsilon}||X - (\phi \circ \upsilon)X||^2, \tag{3.18}$$

where X represents the input space and F the features space.

In the simplest case, which is when we have just one hidden layer, an autoencoder maps the input to a set of *latent variables* representing the initial input, where the adjective latent refers to the fact that this variable are estimated in the hidden part of the network. After the latent representation of the initial input x is generated, this representation is mapped to an output x' of the same shape of x, given that the objective is reconstructing the initial input. In Figure 9 we report the structure of an autoencoder ANN.

During the training phase of an autoencoder, for each input x, the following steps are performed:

- Compute activations at all the hidden layers and in the end at the output layer in a feedforward way in order to obtain an output x'.

- Measure the deviation between x and x' by means of a chosen error metric.

- Backpropagate the error through the entire network and update the weights
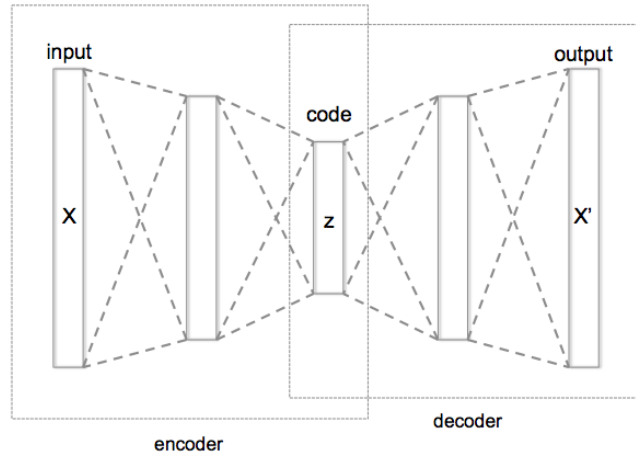
Figure 9: Structure of an autoencoder. [1]

These steps are performed for a chosen number of epochs, using at each epoch techniques with the aim of avoiding overfitting.

With the term overfitting we are referring to the situation in which our model will perform very good on the data used for the training phase but poorly on the ones used in the testing phase. Even if using backpropagation generally provides good results, complications may arise when training networks with many hidden layers. The reason why complication arises is the so called *gradient vanishing problem*, which occurs when errors are backpropagated to previous layers and the gradient gradually becomes smaller until its contribution is insignificant.

---

[1]https://commons.wikimedia.org/wiki/File:Autoencoder_structure.png.
Reprinted with permission, see Appendix.

This could probably lead to the network reconstructing the average of the input data, thus not updating the weights of the connection anymore, meaning that no useful information is extracted anymore. The gradient vanishing problem is sometimes solved by means of more complex training techniques, event if they too often result in a very slow learning process and uneffective solutions, in particular with respect to the trade-off between time and results, critical in neural networks. A reliable solution to this problem was proposed in [38], where the *pretraining technique* is introduced. This technique consists in using a set of initial weights that approximate the final result. More in details, the pretraining technique proposed in [38] involves training in sets of two neighboring layers the network and then fine-tuning the results using standard backpropagation. Different successful experiments using autoencoders in order to perform features extraction can be found in [39], [40] and [41].

3.10.5 Denoising Autoencoder

Denoising autoencoders are autoencoders that take a partially corrupted input (they introduce a noise component) while training to recover the original undistorted input. This technique was introduced with a specific approach based on the concept of *good representation* ([42]).

A good representation is defined as a representation that can be obtained in a robust way starting from a corrupted input and thought to be of good use in recovering the corresponding original clean input. In the definition of this concept an implicit assumption is made: Representations at a higher level must be stable and robust with respect to the corruption of the input. The previous condition must hold in order to extract relevant features for the representation of the input. Furthermore, in order to train an autoencoder to denoise data, a

preliminary stochastic mapping is necessary to perform $x \rightarrow \tilde{x}$, such that the data are corrupted but $\tilde{x}$ can still be used as input for a normal autoencoder, thus allowing the reconstruction of the initial input x in an effective way.

3.10.6 Stacked Denoising Autoencoder

A stacked denoising autoencoder neural network is simply obtained by creating a deep neural network, meaning that we use an high number of hidden layers, where each hidden layer is a denoising autoencoder neural network. This means that the results computed to the first autoencoder layer are passed to the successive autoencoder layer until the output layers is reached. An explanatory representation of the stacked denoising autoencoder neural network is reported in Figure 10.
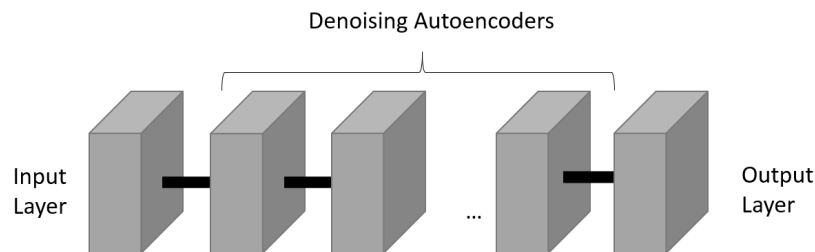
Figure 10: Structure of a stacked denoising autoencoder neural network.

CHAPTER 4

POINT CLOUD DATA

Before going into details of how we reached the thesis objective, we first describe the data used to test the methods proposed in this thesis, their acquisition process and the relative ground truth together with its estimation process.

The point cloud data used are relative to many sites of the Mpala Research Centre in Laikipia region of Kenya (covering an area of 2.5 by 1.2 Km), where a drone *senseFly*[1] was deployed in order to properly capture the area. Drones are a very common technology used to capture surfaces, thanks to the information that could be extracted from *the ultra light images* of the surface ([43]). The drone flew 11 days in January 2015, acquiring data on a different surface each day according to the drone range limitations. From each captured area, which was characterized by various photos taken at different height and from different angles, we obtained the point cloud 3D structure through storing the 3D structure as "las" ([44]) data format by means of the *senseFly* software on the drone. The derived point cloud has a resolution of 1.5 cm. However, there are two main issues related to the data acquisition method chosen. The first issue is that even if with various images different parts of the trees structure are properly captured, the inner parts of the same structure are often partially missing. Inner parts are missing because they are not properly detected due to the fact that they are covered by dense

---

[1] *More information on the company can be found at https://www.sensefly.com/home.html*

external parts of the same tree structure, thus not being captured by means of images only. The second issue is related to the fact that images capturing the surface included also the shadows of objects. When combining the images into the 3D point cloud structure some assembled shadows formed amounts of points with low height but slightly over the ground, thus being classified as non-ground even if they actually are part of the ground.

For each point in the cloud, the following features are available:

- **X:** coordinate corresponding to the *Easting* coordinate of the UTM system.

- **Y:** coordinate corresponding to the *Northing* coordinate of the UTM system.

- **Z:** coordinate representing the height of points in the 3D space.

- **Red:** red color component of the point.

- **Green:** green color component of the point.

- **Blue:** blue color component of the point.

- **Classification**: scalar field that is used by algorithms classifying points in ground and non-ground in order to mark them. According to the standards, a value of 2 for the classification field means that the point is part of the ground, while every other value different from 2 means that it is part of the non-ground elements.

In order to give a general idea of the forest environment in which we are going to perform the segmentation, Figure 11 shows an example of the Kenya savannah, which is the area captured by the photos with the aim of studying the behavior of baboon in their natural habitat.

As we can see in Figure 11, there is no spotted dominance of one kind of tree over the

others, but instead there is a high variety of vegetation (trees and bushes of variable height). Due to the high variability in height, an effective segmentation algorithm should take vegetation variety into account by means of some flexible approach.

Figure 11: An example of the Kenya savannah vegetation.

4.1   Ground Truth Data

In this section we give information about the ground truth availability and acquisition. The areas that we are going to consider for the experiments are four, given the fact that in order to execute all the steps in the framework we must have the ground truth for the trees, thus having

the possibility to verify the correctness of the results obtained.

The ground truth is available only for 119 trees, of which a measure of the height and diameter in two directions was taken, as well as the note about their species and conditions. A figure corresponding to the ground truth collection process is reported in Figure 12. By performing a visual analysis of the ground truth photos we estimated whether the corresponding trees could represent potential obstruction elements in the environment. Due to the fact that we need the ground truth in order to properly validate the results, we restricted the tests of the framework only to the four areas where the trees of which we have the ground truth are located.

Figure 12: Ground truth collection process.

CHAPTER 5

GROUND/NON-GROUND SEPARATION

The first operation that must be performed in order to reach the final objective proposed in this thesis is identifying which points in the point cloud correspond to the ground elements and which to the non-ground ones. In this chapter we will first go through the details of the algorithm used to distinguish between *ground* and *non-ground* points and its tuning procedure, then we will look at the results obtained using the algorithm with the tuned parameters on the areas of interest.

The algorithm used is the *multiscale curvature algorithm for Point Cloud Data* proposed in [14], which was implemented in the C++ freeware command line tool *MCC-LIDAR*[1].

The program requires in input two parameters:

- **Scale (post spacing):** cell resolution of the surface given in input to the program.

- **Curvature threshold:** distance expressed in meters used to compare the interpolated surface with the original one.

In the following sections we will provide a detailed description of the multiscale curvature algorithm and a sequence of all the operations performed by means of suitable pseudo-code.

---

[1] *The software can be found at https://sourceforge.net/p/mcclidar/wiki/Home/*

5.1   <u>Method Description</u>

The first step performed by the algorithm is looking for points with the same x and y coordinate, classifying them as non-ground and then proceed to their removal from the initial point cloud U_0. This is a reasonable choice, considering that the interpolation of a raster surface requires that no two points have the same x and y coordinates. The reason for the points removal is that if points at the same location have the same value, they are seen duplicates and have they will not affect the output. If they have different values instead, they are seen as coincident. Interpolation functions could handle this kind of data in different ways, using in some settings the first point of the coincident points for the calculation, while in other settings the last. Because of this ambiguity, some locations in the output raster might have a value which is different from the expected one. This issue could be easily solved by properly preparing the data by means of the removal of these coincident points, as done in the first step of the algorithm. About the choice of classifying those points as non-ground, it suffices to notice that if two or more points have the same x and y location, then all the points except for the lowest point (i.e. the one with the minimum z coordinate) must be non-ground.

Once dealt with the coordinates issue, for three different scale domains (different surface resolutions), the following steps are performed:

I) A new raster surface is interpolated using the TPS interpolation technique ([34]). Using TPS interpolation offer the possibility to adjust tension between points, fitting input data and handling the distance at which point samples influence the estimate of the surface ([14]).

II) A 3x3 averaging kernel is applied on the computed surface in order to regularize the interpolated surface.

III) For each point, if its height in the original surface is greater than the one in the interpolated surface plus a curvature tolerance threshold for the specific scale domain, then it's classified as non-ground and removed from the original point cloud.

For each scale domain, these steps are performed until we reach a certain convergence threshold. According to the experiments performed in [14], the best convergence criteria was continuing iterating until the number of classified points during the current loop iteration is less than 0.1% of the points still unclassified in the original point cloud. Once processed the surface for the three different scale domains chosen, the points which still remain unclassified are all classified as ground.

5.2 Pseudo-code

In this subsection we first introduce the nomenclature that we are going to use in the pseudo-code and then the detailed pseudcode of the algorithm:

- **SD**: scale domain, integer contained in [1,3] which represents the resolution at which the surface is scanned at the current iteration

- $t_{i \in SD}$**:** curvature threshold for scale domain i, contained in the set of scale domain used. The algorithm scans the surface with three different scales domain, incrementing t of 0.1 meters every time that we increase the scale domain, starting from the initial value given in input by the user.

- **$CR_{i \in SD}$:** post spacing for scale domain i, contained in the set of scale domain used. For the first scale domain, the algorithm uses half of the value given as input by the user, for the second the exact value received in input and for the last the initial value incremented of half its value.

- **f:** tension parameter (invariant across scale domains). The value assigned is 1.5, according to the analysis performed in [14].

- **U:** vector of points that remain unclassified $(P_1, ..., P_n)$.

- **n:** number of points in U (at the beginning of each loop iteration).

- **P_j:** single point in the cloud having coordinates x_j, y_j, z_j.

- **U_0:** initial point cloud given in input by the user.

Once we introduced the nomenclature used, we report the pseudo-code of the main algorithm procedure in Algorithm 1 while the one of the subroutine handling points with same x and y is reported in Algorithm 2.

For the ground/non-ground classification step we adopted the LIDAR classification standard, meaning that when a point is classified as non-ground the value of its classification field is set to any value different from 2 while when a point is classified as ground the classification field is set to 2.

5.3 Scale and Curvature Threshold Tuning

In order to assign the best values to the curvature threshold and the scale parameters, a few experiments with different settings were necessary. More in details, experiments showed

---

Algorithm 1 Multiscale Curvature Algorithm

1: **procedure** MULTISCALECURVATUREALGORITHM
2:     $U \leftarrow$ filterOutPointsSameXY(U_0)
3:    **for** scale domain SD $= 1$ to 3 **do**
4:       **repeat**
5:          $S \leftarrow$ interpolate new raster surface using $TPS(U, CR_{SD}, f)$
6:          $S' \leftarrow$ surface resulting from passing 3x3 averaging kernel over S
7:          **for** each $P_j \in P$ **do**
8:             **if** $z_j > S'(x_j, y_j) + t_{SD}$ **then**
9:                classify P_j as non-ground and remove it from U
10:             $n_C \leftarrow$ number of points classified and removed from U during
11:                 current iteration of the current loop
12:       **until** $n_C < 0.1\% \cdot n$
13:    classify all the points remaining in U as ground

Algorithm 2 Procedure to filter points with same X and Y

1: **procedure** FILTEROUTSAMEXY(U_0)
2:    **for** each x,y location in U_0 with two or more points **do**
3:       $Z_{lowest} \leftarrow$ minimum z coordinate of the points at x,y
4:       **for** each P_j at x,y **do**
5:          **if** $z_j > z_{lowest}$ **then**
6:             classify P_j as non-ground and remove it from U_0

---

that changing the scale parameter does not really affect much the classification outcome on our data, given that the only noticeable changes in the ground/non-ground separation were only at a ground points density level while trees that were classified as ground would remain the same even by running experiments with different values for the parameter. For this reason we changed only the curvature threshold parameter and left the other fixed at 1.5 meters, as suggested by

the developers of the application on the software page. The results of the performed tuning experiments are shown in figs. Figure 13 and Figure 14.
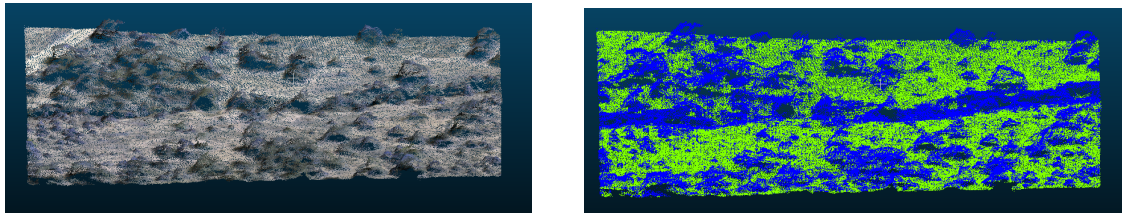
Figure 13: Example of a point cloud surface corresponding to a Kenya surface (on the left) and the same surface classified using a curvature threshold of 0.1 (on the right).
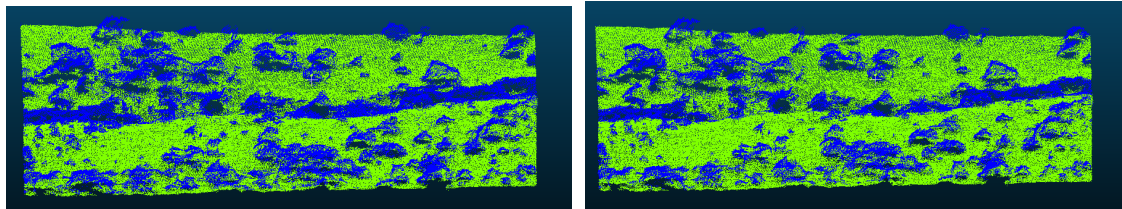
Figure 14: Surface in Figure 13 classified using a curvature threshold of 0.2 (on the left) and using a curvature threshold of 0.3 (on the right).

During the analysis of the results shown in figs. Figure 13 and Figure 14 by means of *CloudCompare*[1], a software that allows to explore the point cloud by zooming and rotating it, we discovered that a curvature threshold of 0.1 would result in classifying many ground points as vegetation (many points of the road in the upper left corner of the surface), while using a curvature threshold of 0.3 would classify many lower trees as ground. Given that 0.2 represent a good compromise between overclassifying and underclassifying, we chose it as best value for the algorithm parameter.

5.4   Separation Result

In this section we show the results of the ground/non-ground separation of the points in the point cloud for the four areas of interest. The results of the experiments on the areas of interest are reported in figs. Figure 15 to Figure 22.
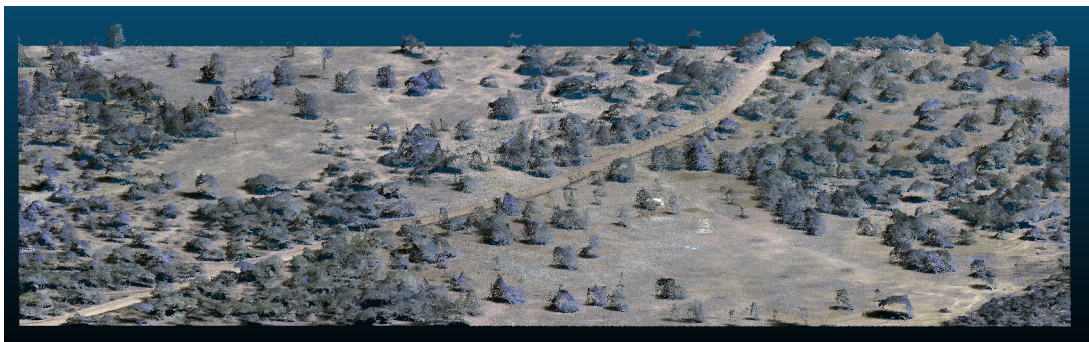
Figure 15: Area 1 of the ones of interest.

---

[1]The software can be found at http://www.danielgm.net/cc/
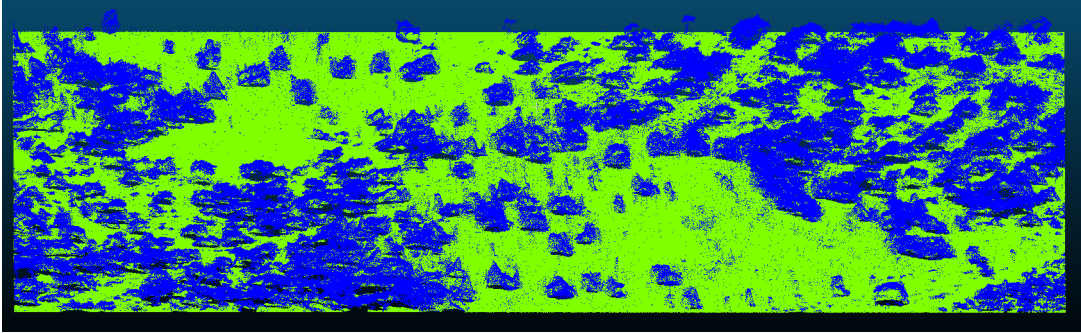
Figure 16: Area 1 of the ones of interest classified (the blue points correspond to the vegetation).

By looking at the results shown in figs. Figure 15 to Figure 22 we can easily notice that almost all the vegetation points are correctly classified but the ground/non-ground separation procedure introduced a strong noise component, partially due to the shadows in the images used to build the point cloud, which were often classified as non-ground elements. In order to weaken the noise component we used the *LasTools*[1] software suite, which contains a set of tools implemented in $C++$ aimed at processing "las" files. More in details, we used the *"lasnoise"* tool, which scans the point cloud data looking for isolated or excessively distant points and remove them from the point cloud. By using this software, thanks to the fact that the points relative to shadows were not much dense and relatively sparse, we were able to remove a great percentage of noisy point from the classified point cloud.

---

[1]The software can be found at http://www.cs.unc.edu/ isenburg/lastools/, it could be used freely
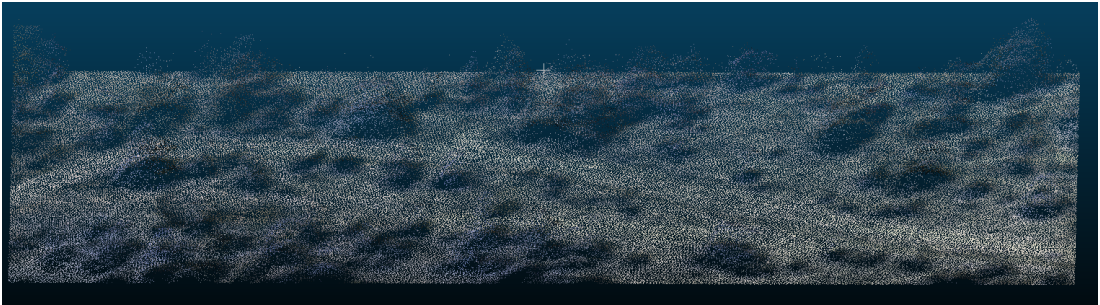
Figure 17: Area 2 of the ones of interest.
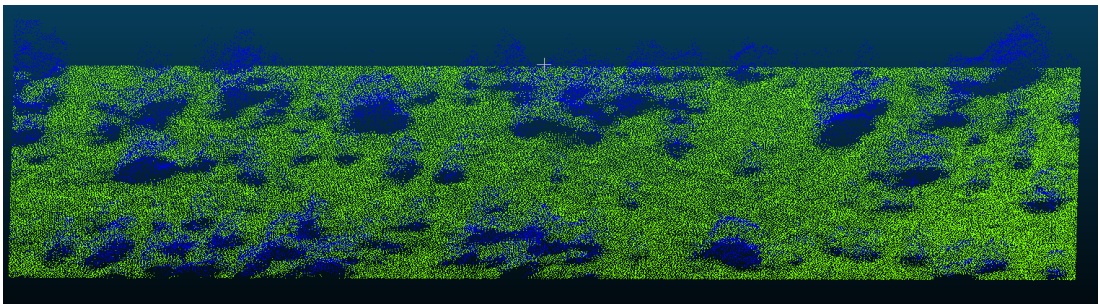
Figure 18: Area 2 of the ones of interest classified (the blue points correspond to the vegetation).
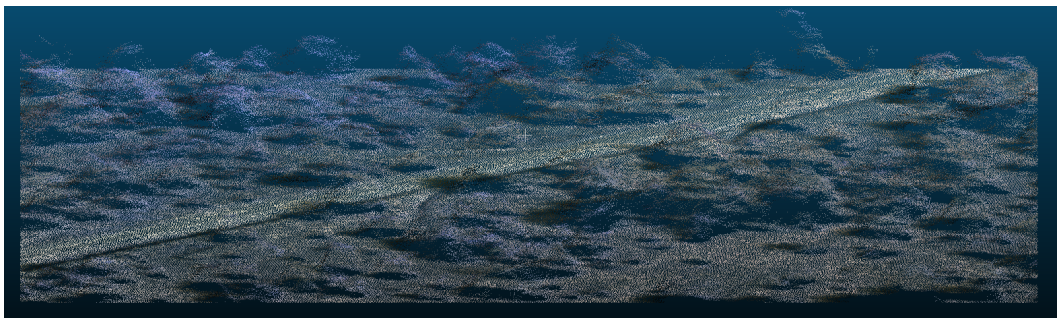
Figure 19: Area 3 of the ones of interest.
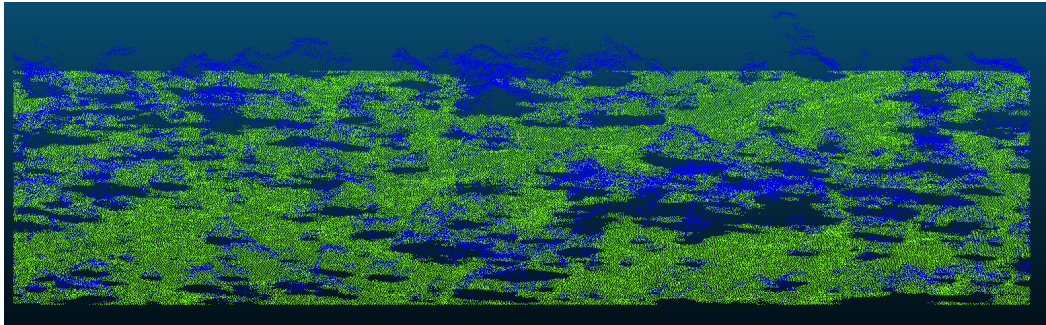
Figure 20: Area 3 of the ones of interest classified (the blue points correspond to the vegetation).
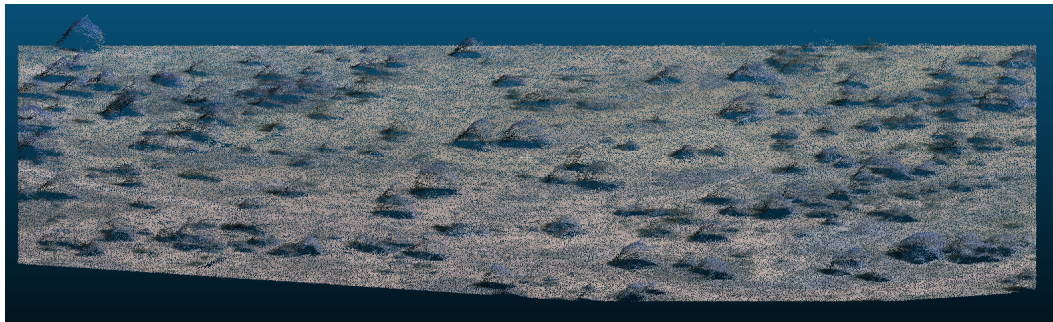
Figure 21: Area 4 of the ones of interest.
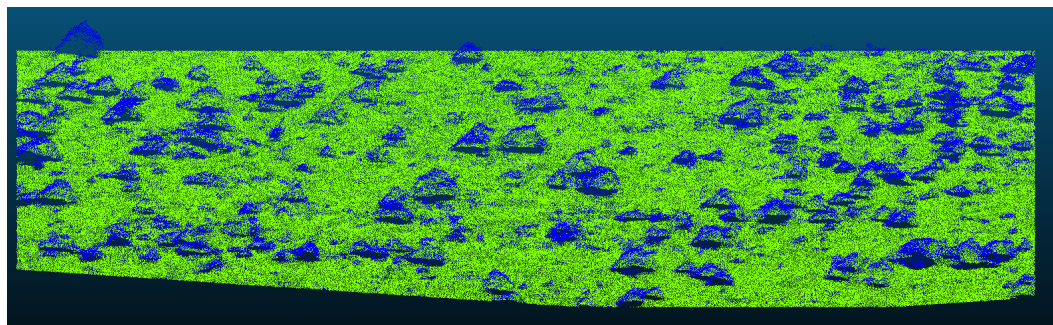
Figure 22: Area 4 of the ones of interest classified (the blue points correspond to the vegetation).

CHAPTER 6

TREE SEGMENTATION

With the information on which points belongs to the ground and which to the non-ground, we can proceed with applying an effective procedure aimed at identifying individual vegetation elements among all vegetation points identified in the previous chapter. In this chapter we will go into details of the *segmentation algorithm* used on vegetation. After having classified the points into ground and non-ground ones, we first computed the real height of the vegetation points with respect to the ground ones in order to remove noisy points with negative or excessively positive height and level the ground. In order to do this, we used the *"lasheight"* tool of the *lasTools* suite, which uses the points classified as part of the ground to compute a *Triangular Irregular Network* (TIN) and then calculates the elevation of each point with respect to the computed TIN. Once we performed this operation, we applied the segmentation algorithm to the surface where height of vegetation points is expressed with respect to the previously identified ground.

In the following sections we will provide a detailed description of the segmentation algorithm used and of the reasons behind its implementation. We will also provide a sequence of all the operations performed by the algorithm by means of suitable pseudo-code.

6.1   Method Description

The proposed method processes the whole cloud finding one tree at each iteration, as in [21]. For the segmentation of each tree the main idea is separating the points into two groups, the ones which actually belong to the tree structure and the ones belonging to other trees. Once we completed the process for one tree, the points corresponding to it are removed from the cloud and the process continue until there are no points left in the initial point cloud. In order to speed up the process, we avoided considering the whole cloud at every iteration, but instead we picked a certain ball of points around the tree top of the considered tree, such to include all the points belonging to it. The estimation of the right radius requires a trial and error process, given that for each kind of forest it is necessary to estimate how many points can belong to a certain tree at maximum in order to avoid losing part of the tree structure. In order to find the target tree top, we look for the highest point in the cloud, which represents the top of the tree that we want to identify during this iteration.

The next step performed is getting all the neighbors' points around the tree top within a chosen radius such to consider the whole tree structure. Once we identified this subset, we keep scanning the point from top to bottom and assign each point to one of the groups according to the minimum distance of each candidate point with respect to the two groups and to an adaptive distance threshold. Proceeding from top to bottom is the best way to identify the candidate tree, given that trees are better delineated at higher levels, while close to the bottom it's much harder to determine to which tree the points belong, given possible overlapping among part of the trees. This is the reason why adopting a top down scanning direction would also

ease the analysis of lower points, given that they will be analyzed once the tree structure is already partially identified.

Once we identified the subset relative to the target tree top, the algorithm is initialized by putting in the group of points relative to the target tree the found tree top, while in the group of the discarded points we put the point in the subset with greatest 3D euclidian distance from the target tree top. After the initialization step , each point subsequent to the tree top is assigned to the right group by means of the following checks:

I) First we check if the minimum distance of the candidate point from the points in the cluster is greater than an *adaptive distance threshold*, determined according to the height of the candidate point. In case of a positive result for this check, we add the point to the discarded ones, otherwise we proceed to step **II**.

II) We check if the minimum distance of the candidate point from the points in the cluster is smaller than the minimum distance of the candidate point from the points discarded during the segmentation of the tree. In case of a positive outcome, the point is added to the cluster points, otherwise it is added to the discarded ones.

We remind that the scanning direction for the points is from top to bottom. Every time that a point is added to one of the two groups, we remove the point from the subset that we are considering, so that by taking the maximum each time we are automatically scanning the points left in the subset in height order. The set of operations performed to assign each point in the subset into the right group is summarized in the flow chart in Figure 23.
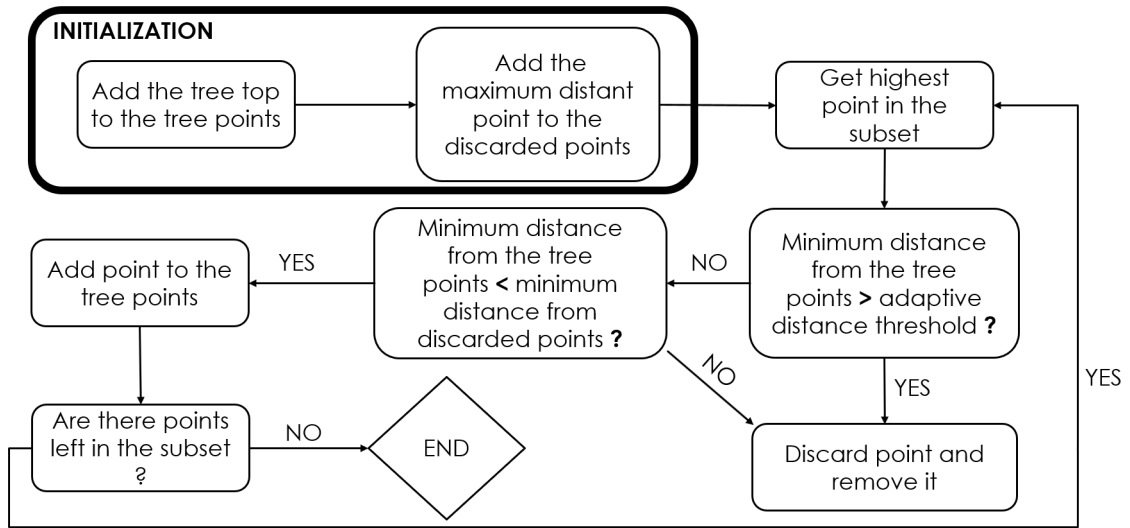
Figure 23: Flow chart summarizing the set of operations performed to identify a specific tree in the subset of points.

From the analysis of the clustering rules listed above, it is clear that the choice of the distance threshold is the critical one. An extensive search of the parameter space in a subset of the trees used as validation followed by a human visual comparison of the produced results between the trees detected and the identified vegetation in the point cloud showed that the best fixed value for the distance threshold is 2.3 meters for the Kenya data.

In Figure 24 we report a line plot with the number of founds trees among the 19 in the validation set of which we have the ground truth on the y axis and the distance threshold value used for the experiments on the x axis.

As shown in the plot, using a distance threshold value of 2.3 meters detected the highest number of trees with ground truth. In addition, this threshold value is also the one producing
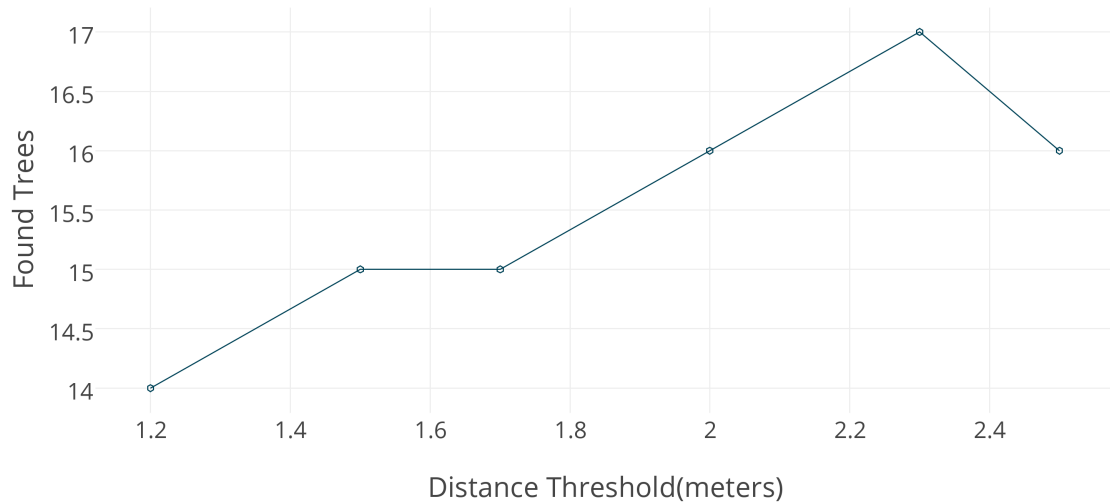
Figure 24: Line plot of the distance threshold values experimented (on the x axis) and of the number of trees with ground truth detected among the 19 in the validation set (on the y axis).

the best result with respect to the visual comparison performed among the original point cloud and the segmented trees.

6.2 Pseudo-code

Now we introduce the nomenclature used in the pseudo-code of the segmentation algorithm and the psuedo-code itself in order give a clearer idea of the steps performed:

- **BS**: the radius of the ball where to search for neighbors in order to obtain the subset.

- **CT**: threshold on the number of points necessary to form a cluster. If the identified tree has less than CP points, then the cluster is discarded, given that the points in it are considered as noisy points generated by the ground/non ground classification. By means

of an empirical analysis we set this threshold to 30, given that we checked that no trees with less than 30 points were present in the point cloud.

- **HT**: height threshold used to determine the value of the adaptive distance threshold. This threshold was set to the 90% of the maximum height of points in the cloud after having analyzed the variation in trees' height in the areas.

- $\boldsymbol{MD_C}$: the minimum distance of a given point from all the point in the cluster during the detection of a specific tree.

- $\boldsymbol{MD_{NC}}$: the minimum distance of a given point from all the points discarded during the detection of a specific tree.

- **DT**: initial value chosen for the adaptive distance threshold.

- **NGP:** non ground points (points corresponding to vegetation).

- **CP:** list containing the points considered as part of the current tree.

- **NCP:** list containing the points discarded during the detection of a specific tree.

- **HP:** Highest Point.

- **TT:** Tree Top.

Before presenting the full pseudo-code of the algorithm, we first introduce the auxiliary functions called by the algorithm:

- **max_height_point(NPG):** returns the point with maximum height among all the points in NGP.

- **get_points(TT, NGP, BS)**: this function builds a KDT with the NGP using the python implementation available in the library *sklearn*[1]. It then returns all the neighbors points within distance BS from the point TT in the created KDT.

- **compute_distance(point_a, point_b, n_dimensions):** computes the euclidian distance between point a and point b in a number of dimensions equal to $n\_dimensions$, which must be smaller or equal than the number of coordinates of the points.

- **get_most_distant_point(point, points):** returns the point in the list of points passed to the function with maximum distance from the one given in input to the function using 3D euclidian distance.

- **compute_minimum_distance(point, points)** returns the minimum distance of the given point from all the points in the set passed to the function using 2D euclidian distance.

- **determine_threshold(HP, HT, DT)**: computes the adaptive distance threshold by returning DT if the height of HP is greater than HT, otherwise it returns DT-0.5 meters, given that we could have some trees which are partially overlapping or which are lower than other trees to which they are close.

In Figure 25 we report a schematic representation of the different spacing between trees at different heights. As we can easily notice in Figure 25, it is clear that the spacing between the points of tree #2 and tree #1 is lower at a certain height, due to the fact that tree #2 is has a

---

[1]*implementation details at http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html*

lower height than our target tree and thus it makes sense to lower the distance threshold of a certain delta meters when the height of the candidate point is below the height threshold. The full pseudo-code of the segmentation algorithm is reported in Algorithm 3.
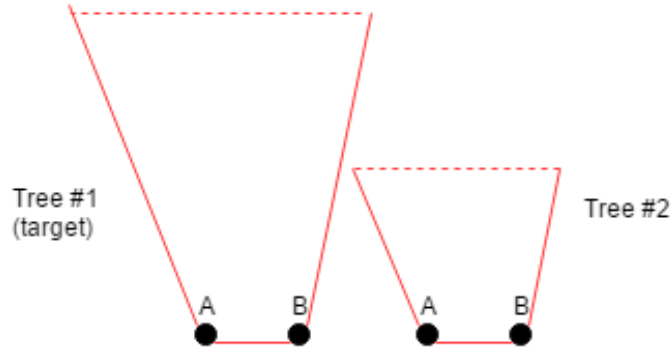
Figure 25: Example of the different spacing between trees at different heights.

6.3 Segmentation Results

In this section we show the experiments performed and the results obtained using the segmentation algorithm on the point cloud data relative to the four areas where trees with ground truth are located. In the results obtained, each tree identified has been marked with a different color in order to differentiate its points from the others.

The segmentation result for Area 1 is reported in figs. Figure 26 and Figure 27, where the algorithm correctly detected 11/12 trees of which we had the ground truth. By analyzing the original point cloud and the one obtained after the ground/non-ground separation we discovered

---
Algorithm 3 Segmentation Algorithm

1: **procedure** TREESSEGMENTATION
2:     $NGP \leftarrow$ filter the point cloud in order to keep only the points with classification $\neq 2$
3:     $DT \leftarrow 2.3$ meters
4:     $CP \leftarrow \emptyset$
5:     $NCP \leftarrow \emptyset$
6:     **while** $size(NGP) \neq 0$ **do**
7:       $TT \leftarrow max\_height\_point(NGP)$
8:       $CP.add(TreeTop)$
9:       $NGP.remove(TT)$
10:       $Subset \leftarrow get\_points(TT, NGP, BS)$
11:       $NCP.add(get\_most\_distant\_point(TT, Subset)$
12:       **while** $size(Subset) \neq 0$ **do**
13:         $HP \leftarrow max\_height\_point(Subset)$
14:         $MD_C \leftarrow compute\_minimum\_distance\_cluster(HP, CP)$
15:         $MD_{NC} \leftarrow compute\_minimum\_distance\_not\_cluster(HP, NCP)$
16:         **if** $MD_C > determine\_threshold(HP, HT, DT)$ **then**
17:           $NCP.add(HP)$
18:         **else if** $MD_C < MD_{NC}$ **then**
19:           $CP.add(HP)$
20:         **else**
21:           $NCP.add(HP)$
22:         $Subset.remove(HP)$
23:       **if** $size(CP) > CT$ **then**
24:         Save CP as an identified tree
25:       $NGP.remove(CP)$
26:       $CP \leftarrow \emptyset$
27:       $NCP \leftarrow \emptyset$

Figure 26: Area 1 original point cloud (trees with ground truth are highlighted by red ellipses).

that the missing tree (highlighted by a red ellipse) was classified as ground by the separation algorithm, probably due to its low height.

The segmentation result for Area 2 is reported in figs. Figure 28 and Figure 29, where the algorithm identified 18/20 trees. By performing the same analysis on the point cloud relative to area 2 before and after the separation we discovered that the two missing trees (highlighted by red ellipses) were not detected because the points corresponding to their structure were classified as ground during the previous step of the framework.

After having encountered the same problem in area 1 and 2, it is reasonable assuming that the separation algorithm has some issues in correctly classifying as vegetation the trees which have too low height. A positive aspect highlighted by the results obtained in area 2 is that the separation algorithm is able to correctly discern between overlapping trees, as shown by the partial overlap of different colors which then proceed highlighting separated trees structures.
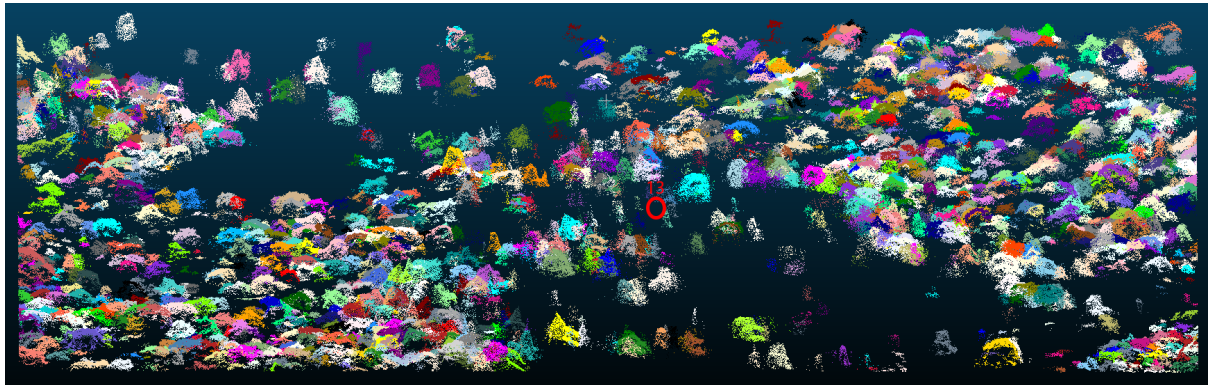
Figure 27: Trees identified by the segmentation algorithm in area 1 (missclassified trees among the ones with ground truth are highlighted by red ellipses).
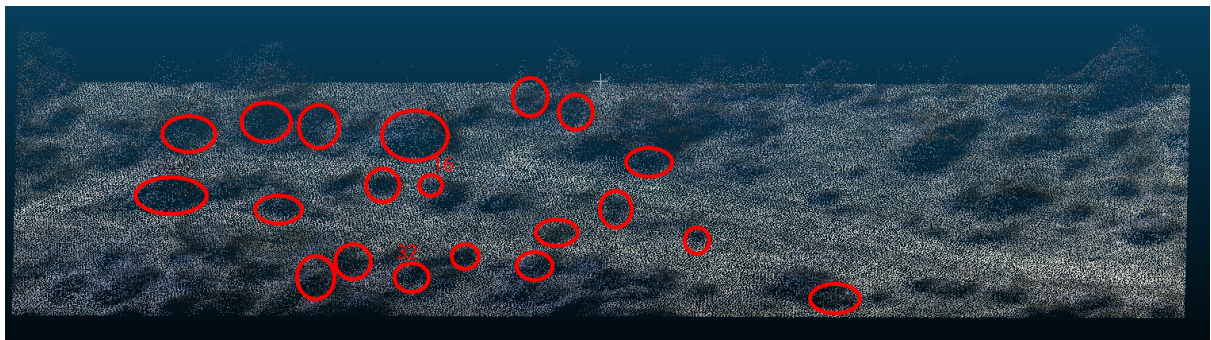
Figure 28: Area 2 original point cloud (trees with ground truth are highlighted by red ellipses).

The segmentation result for Area 3 is reported in figs. Figure 30 and Figure 31, where the algorithm detected 28/32 trees of which we had the ground truth. Many missing ones were not detected due to the same issue of the segmentation algorithm (trees highlighted by the red ellipses with ids 59, 73, 75), which we already discussed with respect to the segmentation results
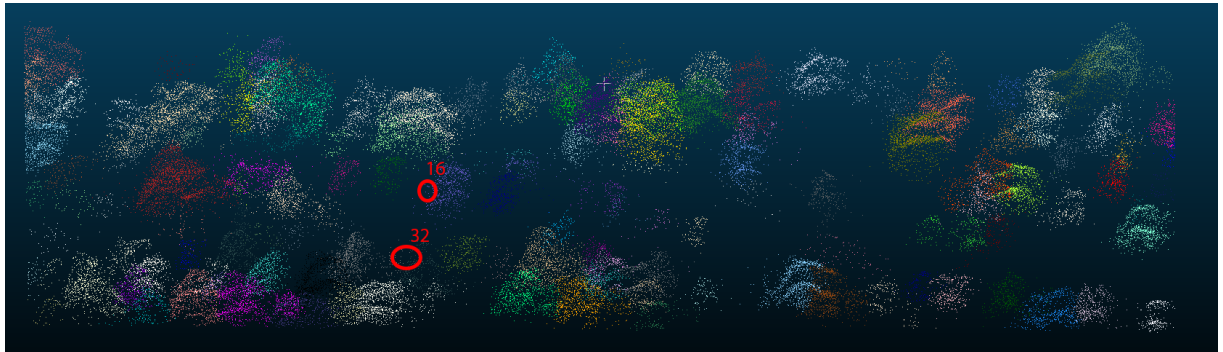
Figure 29: Trees identified by the segmentation algorithm in area 2 (missclassified trees among the ones with ground truth are highlighted by red ellipses).

Figure 30: Area 3 original point cloud (trees with ground truth are highlighted by red ellipses).

in area 2. For what concerns the tree highlighted by the red ellipse with id 99 instead, it was not detected due to the fact that it was too close to another bigger tree, thus the algorithm classified it as part of the bigger adjacent tree.

The segmentation result for Area 4 is reported in figs. Figure 32 and Figure 33, where the algorithm identified 48/55 trees. The majority of the missing trees (highlighted by the red

Figure 31: Trees identified by the segmentation algorithm in area 3 (missclassified trees among the ones with ground truth are highlighted by red ellipses).

Figure 32: Area 4 original point cloud (trees with ground truth are highlighted by red ellipses).

ellipses with ids 179, 202, 203, 211) were classified as ground due to their low height. The fact that the segmentation issue related to low trees affects more Area 4 is predictable, given that Area 4 is the area with the highest percentage of bushes among the areas of interest. The remaining missing trees (highlighted by the red ellipses with ids 170, 171, 219) instead were not detected because they were too close to a bigger tree, thus being classified as part of the
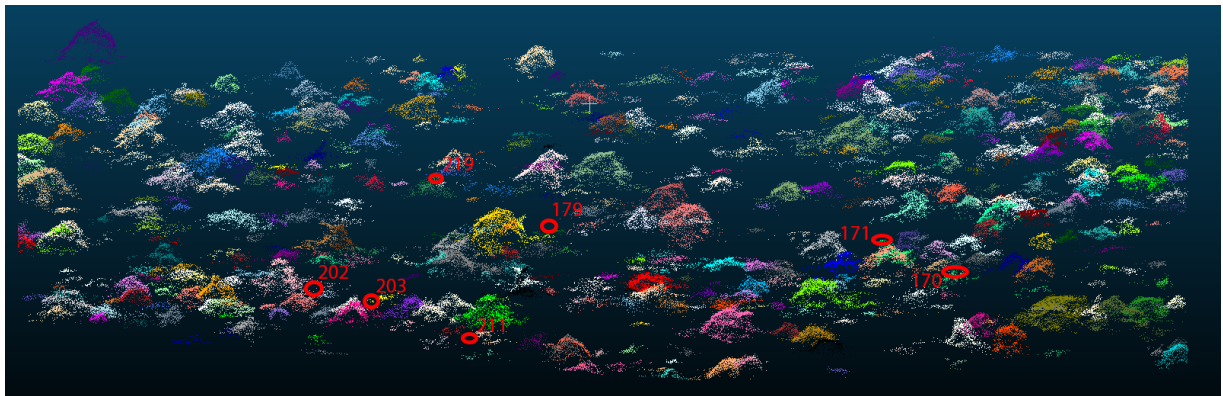
Figure 33: Trees identified by the segmentation algorithm in area 4 (missclassified trees among the ones with ground truth are highlighted by red ellipses).

bigger one. Correctly separating trees which overlap too much or which are too close to each other is a hard task also for the human eye, thus it is reasonable that the algorithm commits some mistakes in the hardest cases.

In order to perform a more detailed analysis of the results we chose in each area a subarea in which it was possible to discern through a visual approach the various trees using Cloud-Compare, thus we were able to compare the trees segmented by the algorithm with the ones in the original point cloud. More in details, we estimated the TP by counting the trees which were correctly detected and the FP by counting the trees which were detected as individual ones but in the original point cloud they were part of bigger tree. The precision of the algorithm with respect to the trees correctly identified in the four subareas is reported in TABLE I.

By analyzing the results reported in TABLE I, we can notice that the segmentation algorithm correctly detected a very high percentage of the trees located in the subareas chosen.

TABLE I: RESULTS OF THE VISUAL VALIDATION PROCESS OVER THE FOUR DIF-FERENT SUB AREAS CHOSEN

Subarea	TP	FN	Precision
Subarea 1	43	12	78%
Subarea 2	36	7	83%
Subarea 3	30	8	79%
Subarea 4	55	13	80%

However, it must be said that this validation approach is not enough detailed to properly validate the results, but given that we don't have enough information about the number of trees in the forest and their ground truth, due to the fact that the ones with ground truth are sparsed across the areas, we cannot perform a more detailed validation procedure. This does not represent a factor influencing in a significant way our final goal, given that even if a single tree is misdetected, as more than one tree or as part of a bigger tree, we are interested in inferring if it is going to represent a potential obstruction factor in the environment and this could be inferred correctly in both cases. About the trees classified as ground, this issue cannot be solved by the segmentation algorithm, considering that those trees are lost during the ground/non-ground separation step.

CHAPTER 7

VISIBILITY FEATURES EXTRACTION

After having obtained the individual vegetation elements among the whole vegetation points, we have to extract features embedding useful information from the trees point structure. In this chapter we will first go through the two main approaches chosen to extract features from the trees point structure and then we will analyze and compare the result obtained by performing inference with the extracted features. We followed two different approached for the features extraction procedure:

- In the first approach we engineered a certain number of chosen features considered suitable for the task at hand and extracted them from the trees' points structure. We will refer to this approach as a *manual extraction* of features, given that we chose before which features we want to extract from the data and then performed the extraction.

- In the second one we used the *stacked denoising autoencoders* neural network to automatically extract features from the trees points structure,.

Before entering into the details of the features extraction process, we first provide a detailed description of the inference task that must be performed.

7.1   Addressed Problem

Given the individually segmented trees through their points structure, our aim is extracting useful features that could be used to infer the visual obstruction potential of a specific tree.

In this setting we are interested in two classes: the "see" class marked as 1 (positive), corresponding to when the tree does not have visual obstruction potential, and the "notSee" class marked as 0 (negative), corresponding to the opposite case. For what concerns the mapping between the ground truth and the individual trees we exploited the fact that the UTM locations of the trees in the photos were known, thus they were easily mapped to the UTM locations of groups of points corresponding to specific trees.

7.2   Manual Features Extraction

In this section we describe in details each one of the features we considered suitable for the inference task at hand and the reasons behind their choice:

- **Height**: height is simply computed by taking the height coordinate of the point with maximum height in the cluster corresponding to the tree. The height is considered a useful feature given that the highest a tree is, the most probable is that it is going to be a potential source of obstruction.

- **Trunk Width**: the trunk width is computed by taking all the points at minimum height and computing the maximum distance between them. This feature is quite significant w.r.t the task, given the fact that the greater the trunk width is, the harder it will be to see through the tree.

- **Density**: density of the tree points is computed applying a *Gaussian Density Kernel* on all the points and then taking the mean of densities of all points in order to obtain one density value for the tree. Density is a crucial aspect with respect to our task, given

that the more points corresponding to vegetation are dense, the harder it will be to see through the tree.

- **Delta X**: represents the maximum variation in the x coordinate of the points corresponding to the tree. The reason why we chose to compute this feature is because including the coordinates interval as features could help the classifier better understand which are the relevant coordinates intervals w.r.t to the inference task.

- **Delta Y**: represents the maximum variation in the y coordinate of the points corresponding to the tree. The reason why we included it is the same for the delta x variation.

- **Average Height**: it is simply computed as the mean of the height of all the points corresponding to the tree. This feature could bring useful insights to estimate how the height of each point influences the outcome.

- **Red Intensity:** the red intensity is computed as the variance of the red values of all points corresponding to the tree. Adding such feature could be useful in order to get the intensity of the tree color, which could be related to visibility issues among animals.

- **Green Intensity:** the green intensity is computed as the variance of the green value of all points corresponding to the tree. This feature could be useful for the same reason of the *Red Intensity* feature.

- **Blue Intensity:** the blue intensity is computed as the variance of the blue value of all points corresponding to the tree.This feature could be useful for the same reason of the *Red Intensity* feature.

7.3   Automatic Features Extraction

In order to automatically extract features from the trees points structure we used a *stacked denoised autoencoder* ANN, where we had to make some modifications to the input in order to make it fit to the ANN requirements:

- Given that neural networks expect a fixed size for the input but each tree is made of a variable number of points, we first found the *minimum size* tree and then sampled without replacement trees with size greater than the minimum one for a number of times equal to the minimum size found before, thus obtaining a fixed number of points for each tree.

- In order to make each input given to the neural network correspond to one tree, we concatenated the points picked from the tree structure into a unique array of size $minimum\_size \cdot features\_number\_of\_point$.

- All the train and test data were normalized between 0 and 1 by means of a *min-max normalization*, given that neural networks are able to learn much more information from normalized data than not normalized ones.

Once prepared the input, we modified the code[1] already provided in the deep learning tutorials[2] based on the *Theano*[3] library.

---

[1]The original code can be found at http://deeplearning.net/tutorial/code/SdA.py

[2]The tutorials can be found at http://deeplearning.net/tutorial/

[3]More information at http://deeplearning.net/software/theano/

The following modifications were applied to the original *stacked denoising autonecoders* source code:

- We modified the network in order to make it return the output of the last hidden layer instead of the output layer, given that it is where the most advanced features are extracted.

- The fine-tuning part was removed because it we were not interested in using the neural network to perform a full inference task but just to extract features in an unsupervised fashion.

After the completion of all the preliminary steps, the network was trained with all the trees available in the four areas except the ones for which we had the ground truth, given that we used them as test set. Once the network was trained and tested, we used the obtained features to perform the classification task using different machine learning classification techniques in order to find the ones performing best for this specific task.

7.4   Experiments and Results

Before performing the experiments, we first had to stratify our dataset, given that there was a significant dominance of the 'notSee' class. The reason why data should be stratified is that otherwise the classifier could be biased towards inferring the class appearing the most (majority class). To perform stratification, we checked which was the majority class and then we took all the instances corresponding to the minority class and an equal amount of instances from the majority class.

Once the dataset was balanced, we performed various experiments, which we are going to describe in the following sections together with the analysis of the result obtained.

7.4.1   Inferring With Manually Extracted Features

The first experiment performed was trying to apply different machine learning techniques once we manually extracted the features presented in Section 7.2.

In order to properly evaluate the results we performed a LOOCV of which we report the results in TABLE II.

TABLE II: ACCURACY, RECALL AND PRECISION RESULTS OF THE INFERENCE TASK WITH MANUALLY EXTRACTED FEATURES

Method	Accuracy	Recall	Precision
Random Forest	0.623 ± 0.017	0.463 ± 0.018	0.636 ± 0.023
Decision Tree	0.617 ± 0.023	0.542 ± 0.019	0.607 ± 0.022
Logistic Regression	0.703	0.378	0.809
K-Nearest Neighbors	0.641	0.512	0.636
Naive Bayes	0.609	0.641	0.581
SVM Polynomial Kernel	0.5	1	0.5

By analyzing the results in TABLE II we notice that all the tested classifiers performed better than the baseline, which is 0.5 given that we have stratified the dataset.

More in details, the best results for accuracy and precision are 70% and 80%, obtained using respectively Decision Tree and Logistic Regression, while for recall the best result is obtained when using Naive Bayes with 61%. The worst result instead is obtained when using SVM with

a polynomial kernel, which inferred exactly as the majority class.

The results obtained are all statistically significant according to an ANOVA test performed, which yielded a p-value of $1.65 \cdot e^{-14}$.

7.4.2 RGB Features Ablation

The next experiment performed was removing the RGB related features from the ones considered in order to check if extracting features relative to the RGB value was actually bringing additional information or not.

In order to validate the results obtained we performed again a LOOCV, obtaining the results reported in TABLE III.

TABLE III: ACCURACY, RECALL AND PRECISION RESULTS OF THE INFERENCE TASK WITH MANUALLY EXTRACTED FEATURES EXCEPT THE RGB RELATED FEATURES

Method	Accuracy	Recall	Precision
Random Forest	0.592 ± 0.025	0.449 ± 0.016	0.606 ± 0.031
Decision Tree	0.621 ± 0.034	0.523 ± 0.015	0.616 ± 0.029
Logistic Regression	0.578	0.459	0.586
K-Nearest Neighbors	0.656	0.452	0.678
Naive Bayes	0.625	0.475	0.633
SVM Polynomial Kernel	0.547	0.514	0.545

From TABLE III we can immediately notice that removing the RGB related features causes a drop in the validation metrics computed, meaning that with the RGB features we are extracting more valuable information for the task at hand.

We can notice that many of the tested classifiers perform worse or in the same way with respect to the previous experiment, exception made for Naive Bayes and SVM with polynomial kernel, which performs considerably better but still worse than the results obtained using also the RGB features in the previous experiment. In addition, results are not statistically significant according to an ANOVA test with p-value 0.967.

7.4.3   Inferring With Automatically Extracted Features

The last experiment performed was inferring using the features automatically extracted with the stacked denoising autoencoders neural network. The main problem using neural networks is finding the best network configuration for the task. The only way to achieve this goal is testing different configurations and analyzing the drops or improvements in the results obtained and acting consequently. Generally speaking, one way to obtain an improvement is incrementing gradually the number of hidden layers and the neurons in each layer, based on the fact that a gradual deeper elaboration of the information at each layer could produce more valuable insights in output. By adopting this approach the improvement will be null or insignificant at a certain point, especially with respect to the trade-off between results and the time needed in order to properly train the network. We again performed a LOOCV in order to properly evaluate the results obtained using each different configuration.

The first configuration tried consists in using all the trees in the four areas as input which

does not correspond to one of the photos and in giving them in input to an autoencoder neural network with 3 hidden layers, the first with 256 neurons, the second with 512 neurons and a target number of features to be extracted equal to 1000 (corresponding to the number of neurons in the last hidden layer of the neural network). In TABLE IV we report the results obtained with the first chosen configuration.

TABLE IV: ACCURACY, RECALL AND PRECISION RESULTS OF THE INFERENCE TASK WITH AUTOMATICALLY EXTRACTED FEATURES USING 3 HIDDEN LAYERS

Method	Accuracy	Recall	Precision
Random Forest	0.753 ± 0.014	0.487 ± 0.007	0.764 ± 0.019
Decision Tree	0.676 ± 0.028	0.531 ± 0.022	0.663 ± 0.032
Logistic Regression	0.75	0.33	1.0
K-Nearest Neighbors	0.734	0.404	0.826
Naive Bayes	0.75	0.354	0.944
SVM Linear Kernel	0.685	0.363	0.8

As we can easily notice, TABLE IV reports an overall significant improvement with respect to the results relative to the manually extracted features according to an ANOVA test with p-value 0.0021. In the best case we obtain 75% accuracy using Logistic Regression, Naive Bayes and Random Forest against the 68% relative to the manually extracted features case, 48% recall using Naive Bayes and 100% precision using Logistic Regression.

The result of this first experiment proves that using neural networks to extract deeper features is indeed promising and worth experimenting more in order to improve the results.

The second configuration tried consists in using again all the trees in the four areas which does not correspond to one of the photos as input and in giving them in input to an autoencoder neural network with 3 hidden layers, the first with 256 neurons, the second with 512 neurons, the third with 1024 neurons and a target number of features to be extracted equal to 2000.

TABLE V: ACCURACY, RECALL AND PRECISION RESULTS OF THE INFERENCE TASK WITH AUTOMATICALLY EXTRACTED FEATURES USING 4 HIDDEN LAYERS

Method	Accuracy	Recall	Precision
Random Forest	0.75 ± 0.015	0.0069 ± 0.008	0.784 ± 0.019
Decision Tree	0.665 ± 0.030	0.512 ± 0.018	0.662 ± 0.032
Logistic Regression	0.75	0.33	1.0
K-Nearest Neighbors	0.75	0.416	0.833
Naive Bayes	0.734	0.361	0.895
SVM Linear Kernel	0.671	0.372	0.761

The result of the inference task with the second configuration adopted are reported in TABLE V, where we can notice that performances are almost the same for Random Forest, Decision Tree, Logistic Regression and SVM with linear kernel with respect to the previous configuration, while we notice a significant improvement for KNN and Naive Bayes, according

to an ANOVA test with p-value 0.0079.

The third configuration tried consists in using again all the trees in the four areas which does not correspond to one of the photos and in giving them in input to an autoencoder neural network with 4 hidden layers, the first with 256 neurons, the second with 512 neurons, the third with 1024 neurons, the fourth with 2048 neurons and a target number of features to be extracted equal to 3000. The results of the third configuration tried are reported in TABLE VI.

TABLE VI: ACCURACY, RECALL AND PRECISION RESULTS OF THE INFERENCE TASK WITH AUTOMATICALLY EXTRACTED FEATURES USING 5 HIDDEN LAYERS

Method	Accuracy	Recall	Precision
Random Forest	0.775 ± 0.012	0.492 ± 0.008	0.782 ± 0.002
Decision Tree	0.753 ± 0.029	0.504 ± 0.017	0.750 ± 0.041
Logistic Regression	0.734	0.240	0.941
K-Nearest Neighbors	0.828	0.453	0.889
Naive Bayes	0.75	0.354	0.944
SVM Linear Kernel	0.75	0.33	1.0

The result shown in TABLE VI confirms the theory that incrementing the numbers of layers and its neurons in a gradual way lead to a significative improvement, confirmed by an ANOVA test with p-value of 0.0026 for this last attempt.

More in details, the above table reports an improvement in accuracy and precision respectively with a 83% obtained with KNN and a 100% obtained with Logistic Regression, while we observe a slight drop in the recall with an highest result of 48% obtained by Random Forest against the 51% obtained by Decision Tree when using the 4 hidden layers network configuration.

Even if we managed to a considerable improvement in the validation metrics used with these last attempts, when using neural networks a trade off between training time and results obtained must be always taken into account, especially when incrementing the number of hidden layers used and the neurons in them.

Experiments with more hidden layers than in the last configuration presented yielded results which did not bring any relevant improvement in any of the metrics used while the training time increased considerably. From the results of experiments with more hidden layers we concluded that incrementing the number of hidden layers and their neurons was not improving performances anymore. An explanation to this limitation could be that not enough trees are given in input to the neural network, thus maybe increasing too much the number of hidden layers with too few trees in input does not allow to extract enough information at too deep levels of the network, thus leaving the result unchanged. For this reason we decided to avoid testing configuration with even more hidden layers, given the fact that we already used all the available trees as input.

7.4.4   Results Comparison

In order to make a comparison among the results obtained in the different experiments and configurations used, in Figure 34 we report a bar chart with the highest results obtained in each performed experiment.
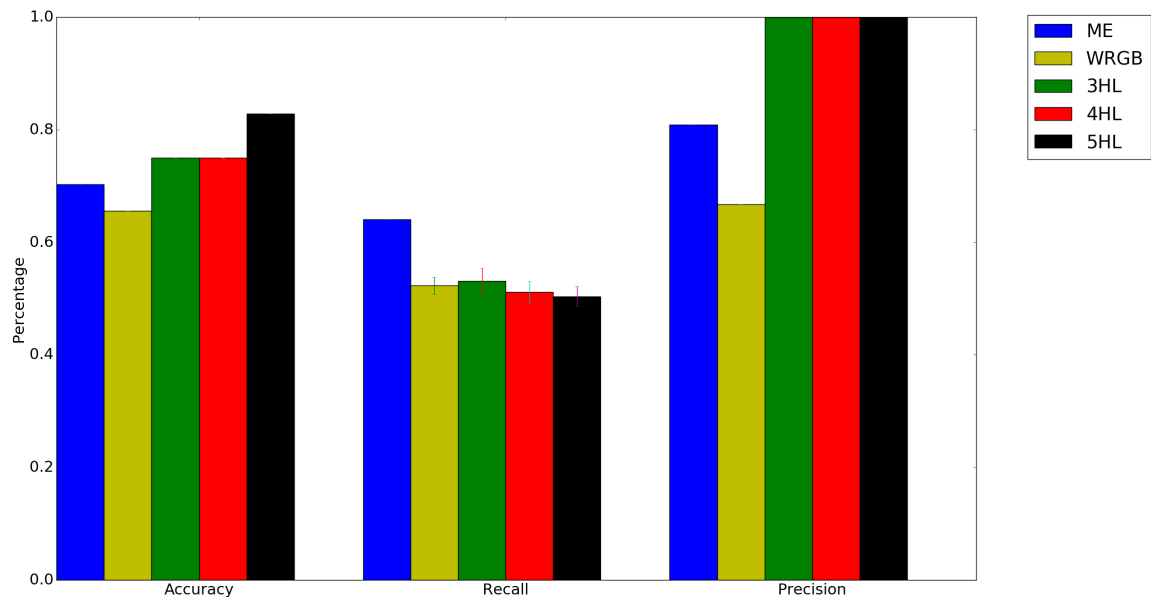
Figure 34: Highest results obtained for each metric during the various experiments where error bars are reported only for results with standard deviation different from zero (legend is ME: *manually extracted features*, WRGB: *RGB features ablation test*, nHL: *configuration with n hidden layers*.

As reported in Figure 34, the best results for accuracy is obtained by the 5HL neural network with KNN, while for recall with the 3HL one by using Random Forest, excluding the

100% obtained when SVM was behaving exactly like the majority class, and for precision with all the neural network configurations by using Logistic Regression and SVM.

This last comparison highlights that the best results with respect to the evaluation metrics used were obtained by means of neural networks, confirming the deep learning effectiveness for the task at hand. From the bar graph we can also notice that overall we obtained low results for recall with a highest value for the metric of 60%, meaning that the classification algorithms used are biased toward the negative class with respect to the dataset used, considering that we have an high number of FN which lower the recall percentage.

CHAPTER 8

LINE OF SIGHT ANALYSIS

Once we obtained the visual obstruction potential labels, we can use the information obtained to analyze the lines of sight among individuals located in the environment. This chapter is focused on going into the details of the procedure used to evaluate the line of sight of two animals and to check whether it is obstructed by some elements with obstruction potential or not. In the following sections we will give a detailed description of the method together with suitable pseudo-code, then we will show the results obtained in the experiments performed.

8.1   Method Description

In order to properly check for the obstruction of the lines of sight, the method requires in input the position of two individuals expressed in the UTM system and their heights.

We approximated the problem to a 2D world in order to speed up the analysis and the whole processing operations. More in details, the idea is approximating each tree with an ellipse built using only two of the three available coordinates. After having tested the results obtained with different sets of coordinates, we chose the Easting and Northing coordinate to build the ellipse, given the fact that they gave us the best approximation according to the results obtained.

In order to model the geometric entities necessary to analyze the lines of sight we used the *python* library *"simpy"*[1].

---

[1] *The library can be found at http://docs.sympy.org/latest/index.html*

81

Once received the trees points structure and the location of the two individuals with their respective heights, the following steps are performed:

I) Compute the 2D line passing between the two individuals using the Easting and Northing coordinate.

II) Collect the trees with obstruction potential using the inferred labels and keep only the ones in range of the Easting and Northing coordinates of the individuals.

III) For each of the trees identified, perform the following checks:

    i) Check if the height of the tree is greater than the one of both individuals. In case of positive outcome, proceed to step **(b)**, otherwise skip the tree given that either it is too low or out of range.

    ii) Compute an ellipse approximating the tree shape using the Easting and Northing coordinate. This is done by using as center for the ellipse the centroid (average of all points for each coordinate) of the tree points for the chosen coordinates, as horizontal radius the value corresponding to half of the maximum variation in the Easting coordinate and as vertical radius the value corresponding to half of the maximum variation in the Northing coordinate.

    iii) Check if the line connecting the two individuals intersects the ellipse approximating the tree. In case of positive outcome, then the line of sight is considered obstructed.

IV) If no tree with visual obstruction potential crossed the line of sight among the two individuals, then the line of sight is considered not obstructed.

The method returns *True* if the line of sight of individuals is not obstructed by a tree with visual obstruction potential, *False* if otherwise.

8.2   <u>Pseudo-code</u>

For completeness we provide also the pseudo-code of the algorithm used to analyze the line of sight among two individuals. Before showing the complete pesudocode, we first describe a set of auxiliary functions called by the algorithm:

- **compute_line(a, b):** this function uses the 'simpy' library to construct a 2D line passing between the two individual locations a and b using the Easting and Northing coordinates.

- **get_trees_with_obstruction_potential_in_range(a, b):** this function returns the trees which were inferred to be visual obstruction potential elements and which are also in range of the Easting and Northing coordinates of the two individuals.

- **height(point):** returns the height of the point.

- **compute_ellipse(center, hradius, vradius):** construct an ellipse having as center the one passed to the function, as horizontal radius and vertical radius the ones specified in input.

- **check_intersection(line, ellipse):** checks if the line in input intersects the ellipse given in input to the function.

The set of operations described in Section 8.1 corresponds to the pseudo-code reported in Algorithm 4.

Algorithm 4 Line of Sight Analysis

1: **procedure** CHECKVISIBILITY(A, B)
2:     $line \leftarrow compute\_line(a, b)$
3:     $trees \leftarrow get\_trees\_with\_obstruction\_potential_{\in}\_range(a, b)$
4:     **for** each $tree$ in trees **do**
5:         **if** $height(tree) < height(a)$ $\wedge$
6:             $height(tree) < height(b)$ **then**
7:             $continue$
8:         **else**
9:             $ellipse \leftarrow compute\_ellipse(tree\_centorid, \frac{delta\_x}{2}, \frac{delta\_y}{2})$
10:             **if** $check\_intersection(line, ellipse)$ **then**
11:                 return $False$
12:     return $True$

8.3   Results and Experiments

In order to properly test the method proposed we labeled all the trees in area 2 by performing an analysis of the point cloud representation of each tree using CloudCompare. Once we obtained the labels, we called the method a certain number of times on random locations in Area 2 range. The outcome was then compared with the result obtained by performing a manual analysis of the point cloud with respect to the lines of the sight corresponding to the random locations generated for the experiment.

By performing 15 calls of the procedure with random locations we obtained a 86% of accuracy, analyzing correctly 13/15 lines of sights.

The visual check used to validate the results obtained does not have an high reliability however, given that labeling in a proper way the trees just by looking at their points structure in the point cloud is a very long shot. Anyway, given the low availability of ground truth for the trees in the four areas, we did not have a different option to perform a more reliable validation procedure for the method proposed.
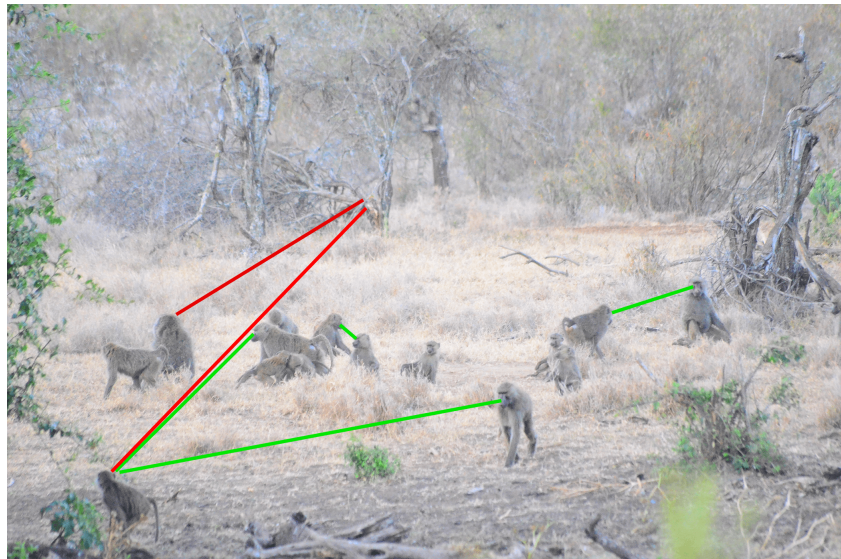

Figure 35: Baboons habitat with not obstructed lines of sight in green and obstructed lines of sight in red (baboons behind the tree pointed by the red lines cannot be seen).

CHAPTER 9

CONCLUSION

9.1   Final Considerations

In this thesis we proposed a framework capable of extracting information about terrain and vegetation starting from raw point cloud data corresponding to forested surfaces. From the vegetation data, each individual vegetation element is then identified by grouping together the corresponding points through a clustering approach. The terrain and vegetation data are then used to infer the visual obstruction potential of the vegetation elements identified by extracting features from their points structure and by performing inference using the extracted features. We provided also a practical application of the data extracted by properly analyzing the lines of sight among individuals located in the environment using the visual obstruction potential inferred and the point structure of the vegetation elements identified. The lines of sight analysis however is just one example of how the information extracted could be used, many other uses are feasible, such supporting the creation of a fully immersive experience of the animals movements and actions in the environment, which would help understanding how animals perceive their habitat during their everyday life, thus providing really useful information for studies related to animal behavior. To identify vegetation elements we described a fast and effective segmentation algorithm that could be used to segment trees in heterogeneously shaped forests, thus providing information on vegetation that could be used to infer even more on the environment itself or

on elements linked to it. To conclude, even if the final objective of this thesis is very domain specific, single steps of this framework could be easily used and integrated in frameworks with different final objectives. For instance, with the ground truth of the specific task that one wishes to perform, the automatic unsupervised features extraction approach making use of neural networks could be easily applied to obtain features for another specific inference task.

9.2   Future Work

The work done in this thesis is extendable in many feasible ways. One of the feasible future improvement could be estimating the threshold value used during the tree segmentation phase in a dynamic way, directly from the characteristics of the points in the point cloud surface, thus making the procedure applicable to any kind of forested environment without the time-consuming tuning procedure. For what concerns the visual obstruction potential inference, training the neural network with an even higher number of inputs, if available, and using an higher number of hidden layers and neurons in the configuration could be an interesting experiment. This experiment could be used to verify whether the limit to the deep learning improvement was just the limited number of training instances or was due to some other aspect of the training procedure, given that in this thesis we were limited to the trees located in the four areas of interest. Another improvement could be obtained in the analysis of the lines of sighs by modeling the relative forested environment by means of 3D technologies. With the 3D information on the environment, it would be possible to perform a more detailed analysis of the lines of sight among individuals, exploiting the orientation at which the lines of sight intersect the vegetation elements and using the height of individuals in a more effective way.

APPENDIX

FIGURES COPYRIGHT

- Figure 1 is subject to a public domain license. The license notice is available at `https://commons.wikimedia.org/wiki/File:Binary_search_tree_search_4.svg`, which allow the reuse.

- Figure 2 is subject to a public domain license. The license notice is available at `https://commons.wikimedia.org/wiki/File:Tree_0001.svg`, which allow the reuse.

- Figure 4 is subject to the GNU Free Documentation License, available at `https://en.wikipedia.org/wiki/Wikipedia:Text_of_the_GNU_Free_Documentation_License`, which allow the reuse. The license notice and the author information are available at `https://en.wikipedia.org/wiki/File:Example_of_kriging_interpolation_in_1D.png`.

- Figure 9 is subject to the Creative Commons Attribution-Share Alike 4.0 International license, available at `https://creativecommons.org/licenses/by-sa/4.0/legalcode`, which allow the reuse. The license notice and the author information are available at `https://commons.wikimedia.org/wiki/File:Autoencoder_structure.png`.

CITED LITERATURE

1. Sanz-Cortiella, R., Llorens-Calveras, J., Arnó-Satorra, J., Ribes-Dasi, M., Masip-Vilalta, J., Camp, F., Gràcia-Aguilá, F., Solanelles-Batlle, F., Planas-DeMartí, S., Pallejà-Cabré, T., et al.: Innovative lidar 3d dynamic measurement system to estimate fruit-tree leaf area. Sensors, 11(6):5769–5791, 2011.

2. Llorens, J., Gil, E., Llop, J., and Queraltó, M.: Georeferenced lidar 3d vine plantation map generation. Sensors, 11(6):6237–6256, 2011.

3. Guo, Q., Li, W., Yu, H., and Alvarez, O.: Effects of topographic variability and lidar sampling density on several dem interpolation methods. Photogrammetric Engineering & Remote Sensing, 76(6):701–712, 2010.

4. Lidar, A. and canopies, O. p.: Lidar remote sensing for ecosystem studies.

5. Wehr, A. and Lohr, U.: Airborne laser scanningan introduction and overview. ISPRS Journal of Photogrammetry and Remote Sensing, 54(2):68–82, 1999.

6. Notebaert, B., Verstraeten, G., Govers, G., and Poesen, J.: Qualitative and quantitative applications of lidar imagery in fluvial geomorphology. Earth Surface Processes and Landforms, 34(2):217–231, 2009.

7. Lim, K., Treitz, P., Wulder, M., St-Onge, B., and Flood, M.: Lidar remote sensing of forest structure. Progress in physical geography, 27(1):88–106, 2003.

8. Früh, C. and Zakhor, A.: 3d model generation for cities using aerial photographs and ground level laser scans. In Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, volume 2, pages II–31. IEEE, 2001.

9. Rutzinger, M., Höfle, B., Hollaus, M., and Pfeifer, N.: Object-based point cloud analysis of full-waveform airborne laser scanning data for urban vegetation classification. Sensors, 8(8):4505–4528, 2008.

CITED LITERATURE (continued)

10. Heinzel, J. and Koch, B.: Exploring full-waveform lidar parameters for tree species classification. International Journal of Applied Earth Observation and Geoinformation, 13(1):152–160, 2011.

11. Chang, Y., Habib, A., Lee, D., and Yom, J.: Automatic classification of lidar data into ground and non-ground points. International archives of Photogrammetry and Remote Sensing, 37:463–468, 2008.

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

13. Elmqvist, M.: Ground surface estimation from airborne laser scanner data using active shape models. International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences, 34(3/A):114–118, 2002.

14. Evans, J. S. and Hudak, A. T.: A multiscale curvature algorithm for classifying discrete return lidar in forested environments. Geoscience and Remote Sensing, IEEE Transactions on, 45(4):1029–1038, 2007.

15. Mäkelä, H. and Pekkarinen, A.: Estimation of forest stand volumes by landsat tm imagery and stand-level field-inventory data. Forest ecology and management, 196(2):245–255, 2004.

16. Tiede, D., Hochleitner, G., and Blaschke, T.: A full gis-based workflow for tree identification and tree crown delineation using laser scanning. In ISPRS Workshop CMRT, volume 5, page 2005, 2005.

17. Korpela, I., Dahlin, B., Schäfer, H., Bruun, E., Haapaniemi, F., Honkasalo, J., Ilvesniemi, S., Kuutti, V., Linkosalmi, M., Mustonen, J., et al.: Single-tree forest inventory using lidar and aerial images for 3d treetop positioning, species recognition, height and crown width estimation. In Proceedings of ISPRS workshop on laser scanning, pages 227–233, 2007.

18. Morsdorf, F., Meier, E., Allgöwer, B., and Nüesch, D.: Clustering in airborne laser scanning raw data for segmentation of single trees. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 34(part 3):W13, 2003.

CITED LITERATURE (continued)

19. MacQueen, J. et al.: Some methods for classification and analysis of multivariate observations. In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, volume 1, pages 281–297. Oakland, CA, USA., 1967.

20. Lee, H., Slatton, K. C., Roth, B., and Cropper Jr, W.: Adaptive clustering of airborne lidar data to segment individual tree crowns in managed pine forests. International Journal of Remote Sensing, 31(1):117–139, 2010.

21. Li, W., Guo, Q., Jakubowski, M. K., and Kelly, M.: A new method for segmenting individual trees from the lidar point cloud. Photogrammetric Engineering & Remote Sensing, 78(1):75–84, 2012.

22. Park, D., Nam, T.-J., and Shi, C.-K.: Designing an immersive tour experience system for cultural tour sites. In CHI'06 extended abstracts on Human factors in computing systems, pages 1193–1198. ACM, 2006.

23. Cormen, T. H.: Introduction to algorithms. MIT press, 2009.

24. Bentley, J. L.: Multidimensional binary search trees used for associative searching. Communications of the ACM, 18(9):509–517, 1975.

25. Aizerman, A., Braverman, E. M., and Rozoner, L.: Theoretical foundations of the potential function method in pattern recognition learning. Automation and remote control, 25:821–837, 1964.

26. Rosenblatt, M. et al.: Remarks on some nonparametric estimates of a density function. The Annals of Mathematical Statistics, 27(3):832–837, 1956.

27. Li, Q. and Racine, J. S.: Nonparametric econometrics: theory and practice. Princeton University Press, 2007.

28. Mohri, M., Rostamizadeh, A., and Talwalkar, A.: Foundations of machine learning. MIT press, 2012.

29. Hastie, T., Tibshirani, R., and Friedman, J.: Unsupervised learning. Springer, 2009.

30. Rusinkiewicz, S. and Levoy, M.: Qsplat: A multiresolution point rendering system for large meshes. In Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pages 343–352. ACM Press/Addison-Wesley Publishing Co., 2000.

CITED LITERATURE (continued)

31. Everitt, B. S.: The cambridge dictionary of statistics. The Indian Journal of Statistics, 69(4):887–888, 2007.

32. Dracup, J. F.: Geodetic Surveying 1940-1990. US Department of Commerce, National Oceanic and Atmospheric Administration, National Ocean Service, 1995.

33. Rabiner, L. R.: Multirate digital signal processing. Prentice Hall PTR, 1996.

34. Briggs, I. C.: Machine contouring using minimum curvature. Geophysics, 39(1):39–48, 1974.

35. Wasserman, P. D.: Advanced methods in neural computing. John Wiley & Sons, Inc., 1993.

36. Rojas, R.: The backpropagation algorithm. In Neural networks, pages 149–182. Springer, 1996.

37. Bengio, Y., Courville, A., and Vincent, P.: Representation learning: A review and new perspectives. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 35(8):1798–1828, 2013.

38. Hinton, G. E. and Salakhutdinov, R. R.: Reducing the dimensionality of data with neural networks. Science, 313(5786):504–507, 2006.

39. Bengio, Y.: Learning deep architectures for ai. Foundations and trends® in Machine Learning, 2(1):1–127, 2009.

40. Liou, C.-Y., Huang, J.-C., and Yang, W.-C.: Modeling word perception using the elman network. Neurocomputing, 71(16):3150–3157, 2008.

41. Liou, C.-Y., Cheng, W.-C., Liou, J.-W., and Liou, D.-R.: Autoencoder for words. Neurocomputing, 139:84–96, 2014.

42. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. The Journal of Machine Learning Research, 11:3371–3408, 2010.

43. Küng, O., Strecha, C., Beyeler, A., Zufferey, J.-C., Floreano, D., Fua, P., and Gervaix, F.: The accuracy of automatic photogrammetric techniques on ultra-light uav

imagery. In UAV-g 2011-Unmanned Aerial Vehicle in Geomatics, number EPFL-CONF-168806, 2011.

44. Chen, Q.: Airborne lidar data processing and information extraction. Photogrammetric engineering and remote sensing, 73(2):109, 2007.

45. Jawak, S. D., Panditrao, S. N., and Luis, A. J.: Validation of high-density airborne lidar-based feature extraction using very high resolution optical remote sensing data. Advances in Remote Sensing, 2013, 2013.

46. Tarsha-Kurdi, F., Landes, T., Grussenmeyer, P., et al.: Hough-transform and extended ransac algorithms for automatic detection of 3d building roof planes from lidar data. In Proceedings of the ISPRS Workshop on Laser Scanning, volume 36, pages 407–412, 2007.

47. Meng, X., Currit, N., and Zhao, K.: Ground filtering algorithms for airborne lidar data: A review of critical issues. Remote Sensing, 2(3):833–860, 2010.

48. Chang, Y.-W., Hsieh, C.-J., Chang, K.-W., Ringgaard, M., and Lin, C.-J.: Training and testing low-degree polynomial data mappings via linear svm. The Journal of Machine Learning Research, 11:1471–1490, 2010.

VITA

NAME	Benedetto Vitale
EDUCATION	
	Master of Science in Computer Science, University of Illinois at Chicago, May 2016, USA
	Master Degree in Computer Engineering , Sep 2016, Polytechnic of Milan, Italy
	Bachelor's Degree in Computer Engineering, Sep 2014, Polytechnic of Milan, Italy
LANGUAGE SKILLS	
Italian	Native speaker
English	Full working proficiency
	2014 - TOEFL examination (96/110)
	A.Y. 2015/16 One Year of study abroad in Chicago, Illinois
	A.Y. 2014/15. Lessons and exams attended exclusively in English