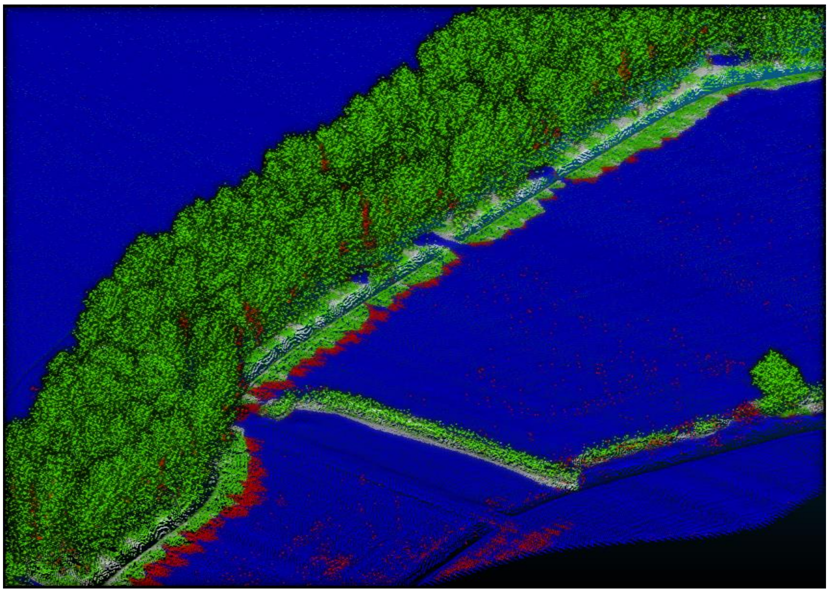
9/6/2018

Towards identifying 3D properties from LiDAR point clouds at a continental scale

Master thesis



Berend-Christiaan Wijers
UNIVERSITY OF AMSTERDAM
10008659
Supervisor: Prof. Dr. Ir. W. Bouten
Co-Supervisor: Dr. A.C. Seijmonsbergen

Summary

The loss of habitat and alteration of the environment has caused a multitude of species to go extinct, and corresponding ecosystems are declining at an alarming rate. Predictive biodiversity distribution modelling (PBDM) is a technique to provide insight in the distribution of organisms, which can be beneficial in the preservation of species. Currently, biodiversity distribution modelling is severely impeded by the lack of high resolution measurements of 3D ecosystem structures. Structure tensors derived from LiDAR point clouds turn out to be an effective measure on describing the shape of groups of points. However, calculating these structure tensors proves to be a computational heavy approach. As such, upscaling to a continental scale is unfeasible. The research on the effectiveness of structure tensors has resided in the field of Earth sciences without using the development of Informatics which might be the missing link for continental scale processing. By using an interdisciplinary approach, the combination of Informatics and Earth sciences is shown to be a viable solution for large scale structure tensor calculations. Through writing a dynamic code which adjusts itself based upon hardware and data, previous computational limitations were overcome. This thesis shows that using dynamic software improves the calculation speed by 1190 times. Continental scale structure tensor calculations now become feasible. Furthermore, classification accuracy of extreme low point densities LiDAR tiles (<5 points/m^2) have gone up from 35% to 78% by applying an adaptive neighborhood search. Considering the value of structure tensors derived from point clouds, large scale calculation can provide a wealth of data for future research in multiple scientific fields.

Keywords: Continental scale, LiDAR, Multi-processing, Parallelization, Vectorization

Acknowledgements

Throughout this thesis, I have received a tremendous amount of support from my supervisors, research group, friends and family for which I want to thank them. First and foremost, I want to thank my supervisor Willem Bouten for guiding me through the process of the development of a research proposal and thesis and my co-supervisor A.C. Seijmonsbergen for providing invaluable feedback. You have given me room to develop while making sure I stay on track. Furthermore, the cooperation with Zsófia Koma has been nothing short of amazing and has been invaluable in diving deeper in the tools and concepts of LiDAR processing. I would also like to thank Daniel Kissling for generously giving me access to the eEcoLiDAR servers which has been the integral key to developing this thesis. Chris Lucas, thank you for bringing me up to speed about your research even though you were finalizing your thesis. Friends and family, you have been very supportive throughout these past 10 months and have always been a listening ear, your support has kept me going even when things seemed grim. Vetle Brænd, thank you for the late-night discord sessions on computing concepts and being a listening ear. Hugo Pineda Hernández and Morten Andersen, you have been an amazing help in the acquisition of foreign datasets. Pieter Zitman, thank you for being a great and supportive friend throughout my education, this thesis was no exception.

Table of Contents

1 Introduction

The loss of habitat and alteration of the environment has caused a multitude of species to go extinct, and corresponding ecosystems are declining at an alarming rate (Hoekstra, et al., 2005; Ceballos, et al., 2015; Cardinale, et al., 2012; Newbold, et al., 2015).

Many ecosystems provide services which are essential to us. As such, national and international programs such as Group on Earth Observations Biodiversity Observation Network (GEO BON), the United Nations (UN) Convention on Biological Diversity (CBD), or the Intergovernmental Science Policy Platform on Biodiversity and Ecosystem Services (IPBES) acknowledge the need to be able to quantify ecosystem structures. By quantifying ecosystem structures, land use changes and their impacts on biodiversity can be assessed. The assessment of these structures and their change is an invaluable tool in the determination of management policies world-wide such as Aichi Biodiversity Targets for 2020 set by the CBD (Pereira, et al., 2013; Skidmore & Pettorelli, 2015; Kissling, et al., 2015).

Predictive biodiversity distribution modelling (PBDM) is a technique to provide insight in the distribution of organisms (Rodríguez, et al., 2007). Humanity has severely altered their surroundings and ecosystems have had to adapt accordingly. Animals depend on a vertical and horizontal distribution of vegetation on different spatial scales (Buler, et al., 2007; Cody, 1985; Wiens, 1989; Fuller, 2012; Kissling, et al., 2008). Currently, biodiversity distribution modelling is severely impeded by the lack of high resolution measurements of 3D ecosystem structures (Lausch, et al., 2016; Davies & Asner, 2014; Paula, et al., 2016). Traditionally, biodiversity distribution modelling uses passive remote sensing techniques in order to compute probabilities of where an organism would prefer to reside (Ferrier, et al., 2002; Stockwell & Peterson, 2002). Grid cell data, which is the current dominant data type for Predictive Biodiversity Modelling (PBDM), usually has no information on the vertical composition of vegetation. This generally means that within a single grid cell, the most dominant land-cover type is assigned. Information is lost when grid cells are large (low resolution > 1-30m) (Turner, et al., 2003).

Furthermore, current species distribution modelling techniques are often required to make a choice between the lack of spatial extent but high resolution or a large spatial extent but low resolution. For instance, the Corine land-cover database provides information on various types of land cover, worldwide (Corine, 2016). However, the data distinguishes 44 different land cover types resulting in only very few forest types (3) and wetland types (5). Small and scattered habitats are then lost and or not included, which might be relevant in distribution modelling.

LIght Detection And Ranging (LiDAR) is a method of retrieving data about distances between the scanner and object(s). Raw laser data is processed onboard after which the resulting data is often referred to as a point cloud. A point in a point cloud has a location in three dimensions. For example, a single point reflected off a surface has a corresponding x, y, z and additional attributes such as intensity, number of returns etc. By collecting a large amount of points per surface area, detailed analysis can be performed. A common practice is to create a raster model, based on the point cloud. There are various platforms of retrieving LiDAR data: Airborne (ALS), Spacebourne (SLS) and Terrestrial Laser Scanning (TLS). ALS, utilizing either unmanned air vehicles (UAV'S), airplanes or helicopters, is a common practice. A typical range of point cloud densities for ALS is between 1-20 points/m^2. Occasionally, TLS and ALS are combined to retrieve very highly detailed information. However, area coverage by TLS is rather limited and SLS is promising, but not yet wildly available (Patrick McCormick, 2005; Popescu, et al., 2011).

A point cloud is a set of points which hold x, y, z information. The amount of points within a point cloud differs greatly between datasets and a way of describing the difference between point clouds is for example the amount of points per square meter. These point clouds can be enriched with extra information, such as: Number of returns, return number and intensity. A common issue in the use of LiDAR data is that the data is too heavy to handle. A LiDAR tile can be comprised of many millions of measurements all with a unique x,y,z value. Processing LiDAR data is a computational challenge due to the sheer file sizes and number of points to process. Two major software solutions have been developed to process LiDAR data across large spatial extents: Orientation and Processing of Airborne Laser Scanning Data (OPALS) and LAStools. However, both of these software solutions are not open source. Furthermore, LAStools is restricted in deriving grid-based features as opposed to OPALS full point cloud feature calculations (Isenburg, 2018; Pfeifer, et al., 2014). Therefore, multiple strategies are employed to reduce the computational strain. A common strategy is to transform LiDAR data into a x-y grid with an average value of z. This means that the information LiDAR holds becomes 2.5 dimensional, due to aggregating multiple points together as one cell. Another strategy is to subsample a point cloud, also referred to as thinning. Thinning can be an effective strategy to reduce computational strain, as you retain the 3 dimensionalities of the data but reduce the number of points, reducing the point density. The reduction in point density can imply that certain small objects won't be described as these points might be filtered out during thinning. However, the data retains the possibility to represent 3 dimensional objects, which is in certain practices, such as PBDM a welcome addition. The reduction of the number of points is a strategy which can be used to cover a certain area and still be able to process it within an acceptable timeframe. Another choice is to reduce the area of interest, which is also an effective methodology to retain the high resolution obtainable by LiDAR while also being able to process the data within a reasonable time. The decision of which strategy to use is dependent on the research aims.

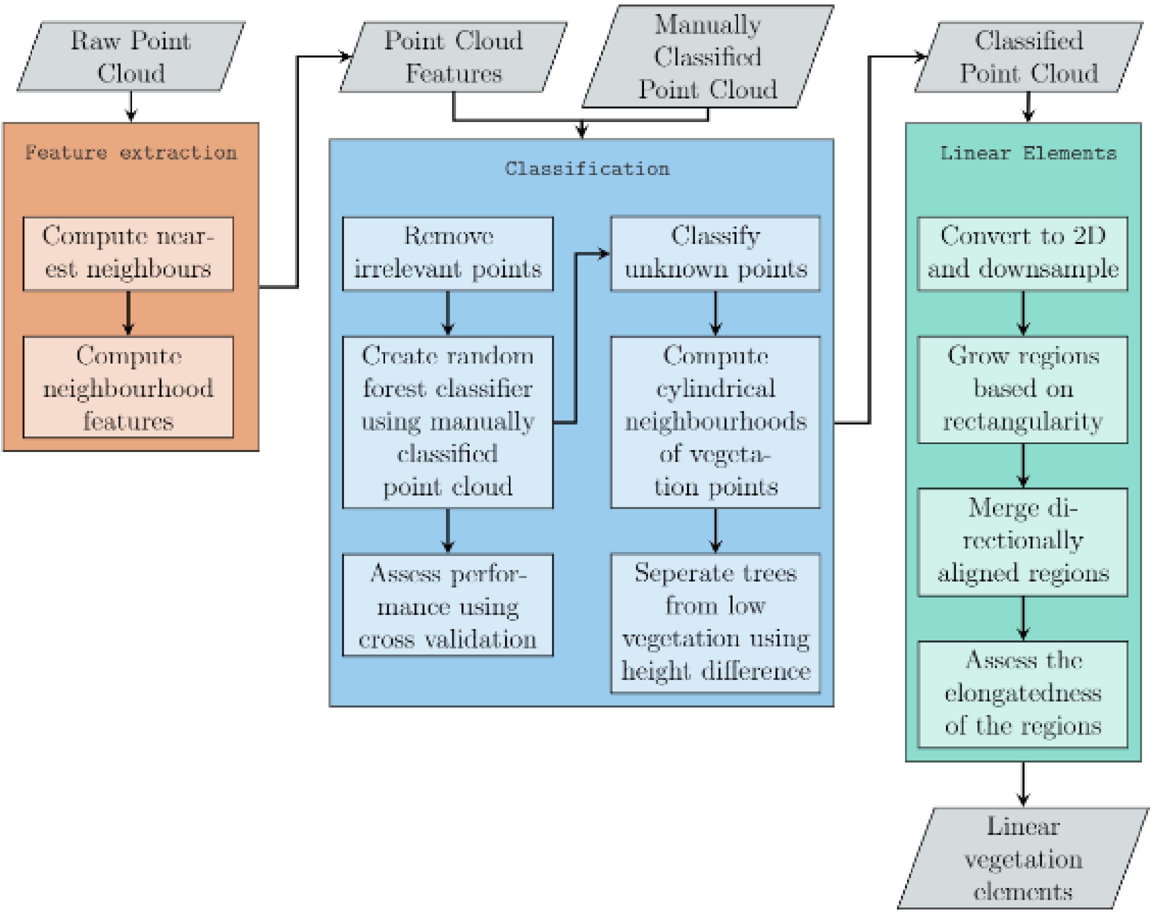

Raw Point Cloud
Feature extraction
Compute nearest neighbours
Compute neighbourhood features
Point Cloud Features
Manually Classified Point Cloud
Classification
Remove irrelevant points
Create random forest classifier using manually classified point cloud
Assess performance using cross validation
Classify unknown points
Compute cylindrical neighbourhoods of vegetation points
Seperate trees from low vegetation using height difference
Classified Point Cloud
Linear Elements
Convert to 2D and downsample
Grow regions based on rectangularity
Merge directionally aligned regions
Assess the elongatedness of the regions
Linear vegetation elements

Figure 1:Original workflow (flowchart) for linear vegetation element delineation. (Lucas, 2017)

A novel processing approach, shown in figure 1, has been developed which uses four geometric (Appendix A, figure 24) properties and eight structure tensors (Appendix A, figure 25) as point cloud features to be used in a Balanced Random Forest classifier to classify points in to a vegetation or non-vegetation class (Lucas, 2017). Structure tensors describe the directions of a neighborhood of a point, based upon the eigen values and eigenvectors of the covariance matrix of the x,y,z values for a group of points (Lucas, 2017). The root of the workflow lies in the description of points and their neighbors. By describing the shape of a point and its neighbors a Random Forest classifier can be used to determine to which class a point belongs. The description of points is done by retrieving the spatial location of a (variable) number of neighbors around a point to be assessed. A covariance matrix is calculated for the group of points after which eigenvalues and vectors are derived. Based upon these eigenvalues and vectors 12 spatial-descriptive features are calculated. Multiple spatial-descriptive features are used to classify points as vegetation and non-vegetation. These features, shown in Appendix A figure 24 & 25 are used in a Random Forest classifier.

Currently, the usage of structure tensors is restricted mostly to a local scale due to data availability and processing speed. As deriving structure tensors from point clouds is a relative novel approach – there is currently no open-source framework capable of handling the required datasets at an acceptable speed. This, for example, limits the possibility of upscaling to continental scales.

Furthermore, upscaling to a continental scale implies using varying LiDAR datasets which all have their own unique properties. The eEcoLiDAR is a project which aims to fill the gap of high-resolution data for PBDM by using LiDAR data to characterize vertical and horizontal complexity of vegetation with high-resolution across continental scales (Kissling, et al., 2017). Considering the goals of the eEcoLiDAR project, the delineation approach should be made applicable for a continental scale as opposed to a local scale. There currently is no continental scale LiDAR data set. This means that generating output for a continental scale, different data suppliers need to be contacted and data needs to be retrieved. The issue with this is that there is a lot of variability between data sets. These differences result from varying operator decisions and varying LiDAR system settings.

Therefore, the aim of this research is to develop a scalable computational framework which scales based upon input data and (processing) machine hardware. The secondary aim is to ensure that the computational framework is capable of classifying LiDAR data from varying datasets.

Two major bottlenecks were identified: 1) Computational performance and 2) data handling. By applying current computing practices to a research-oriented result, computational performance is gained while maintain high accuracy across multiple national datasets. This causes the continental scale calculation to be feasible and might prove to be a good basis for further development.

2 Data

The datasets used for this research are displayed in table XX. For the computational performance analysis, a subset of 10 million points from a tile from the Algemeen Hoogtebestand Nederland (AHN3) dataset will be used. This equates to a surface area of 0.625 km^2 and is from the region Beesd Geldermalsen. The data sources for the LiDAR data presented in table 1 are available in appendix D, data sources.

Table 1: Data used for the computational performance and classification assessment.

Country	Dataset	Coverage	Points / m^2	L,W (km)	Attributes
Denmark	-	Complete	20	1,1	xyz, returns, intensity, RGB classification
The Netherlands	AHN3	Partial	16	18.75, 18.75	xyz, returns, intensity, classification
Belgium	DHMVII	Complete	10-20	0.5, 0.5	xyz, returns, intensity, RGB classification
Belgium	DHMVI	Complete	1-2	0.5, 0.5	xyz
Spain	-	Complete	0.5	2,2	xyz, returns, intensity, RGB classification

The vegetation classification is done on a single tile per dataset. However, for AHN3 (The Netherlands) a subset has been chosen of 6,8 million points with a surface area of 0,428 km^2. Furthermore, all of the study areas have an agricultural landscape with a variation of vegetation elements such as: shrubbery, hedges, single trees and sometimes small forests.

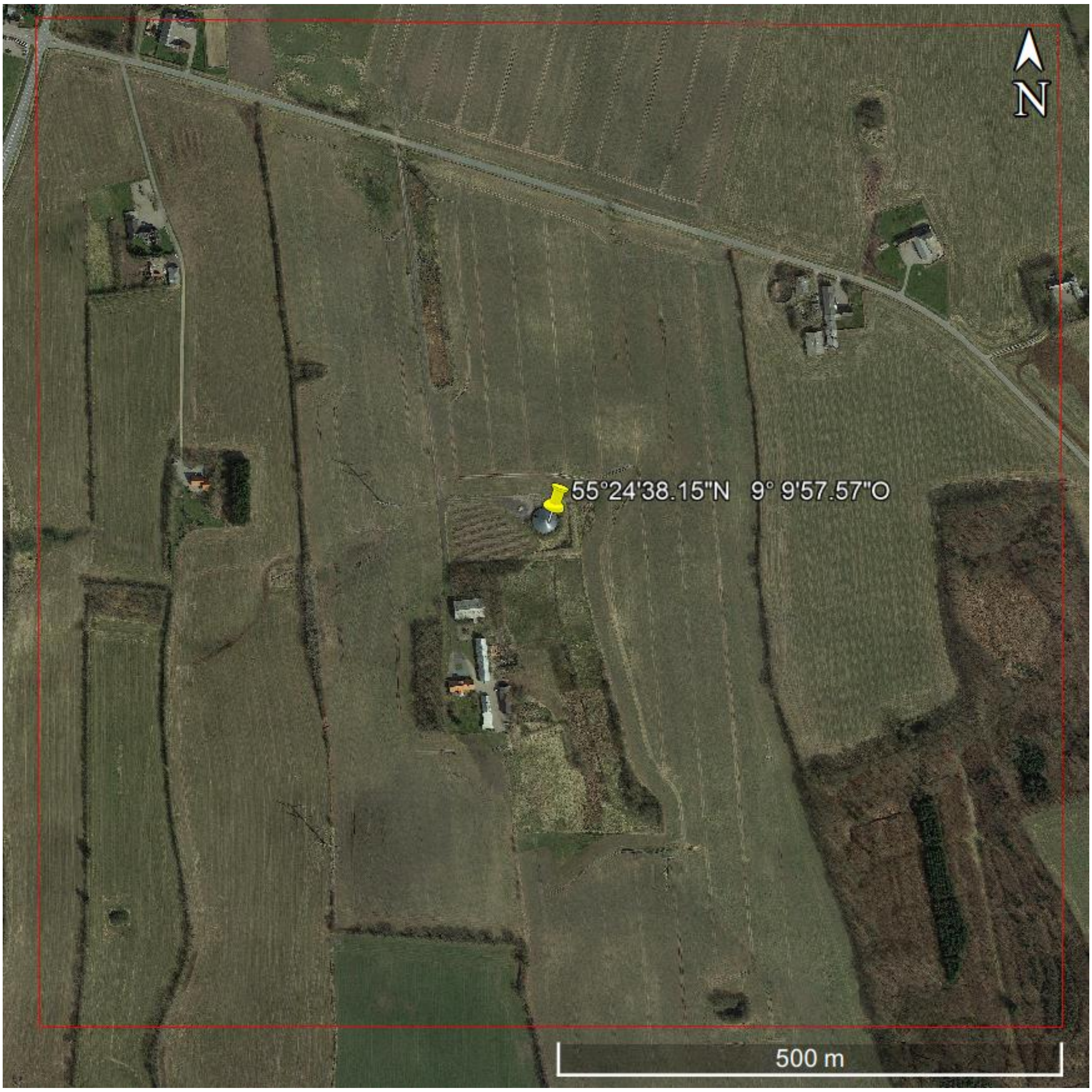


Figure 2: Study area in South West Denmark, between Vejen and Vamdrup, 1 km².

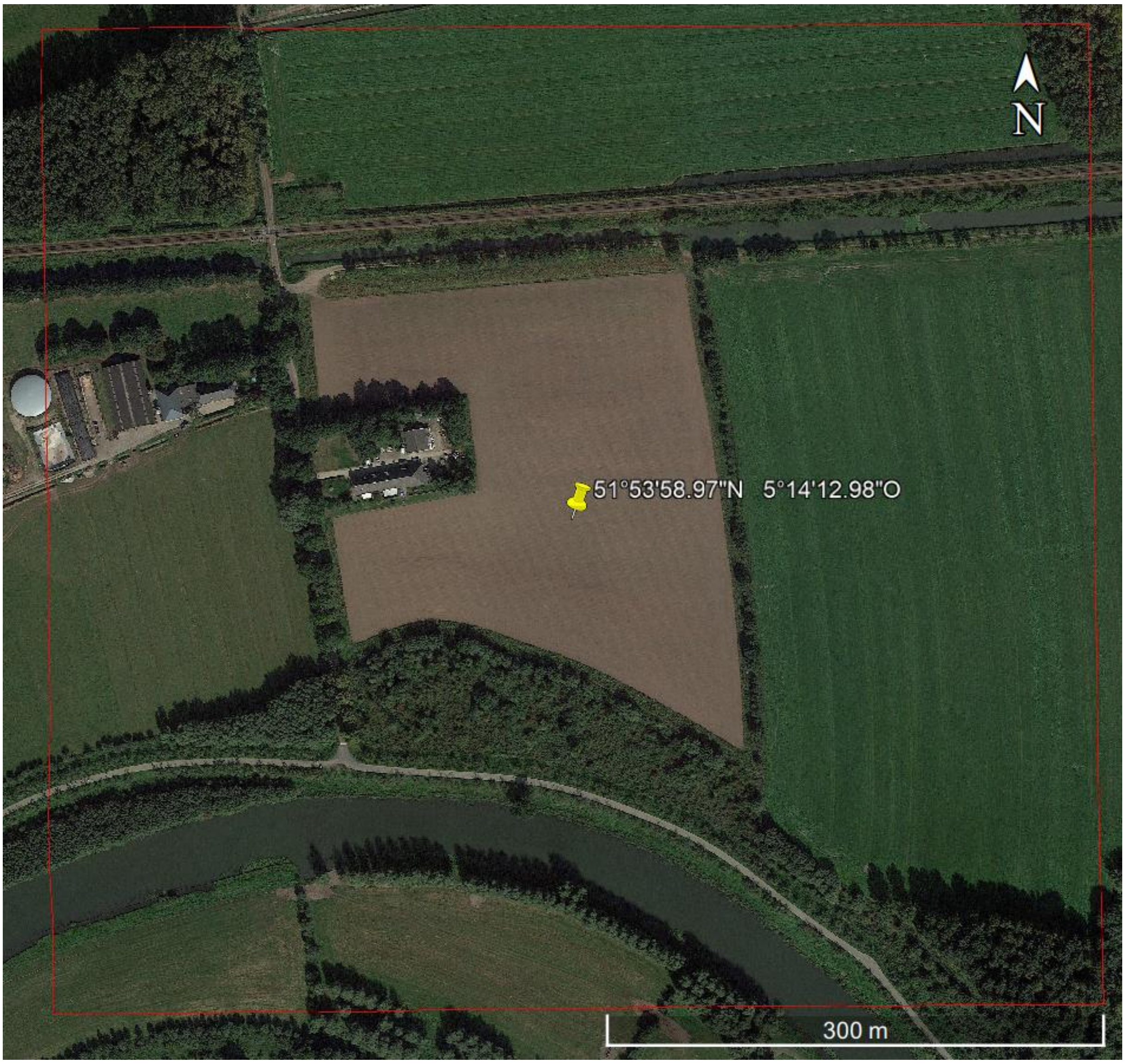


Figure 3: Study area in the South of The Netherlands, between Beesd and Geldermalsen, 0,428 km²

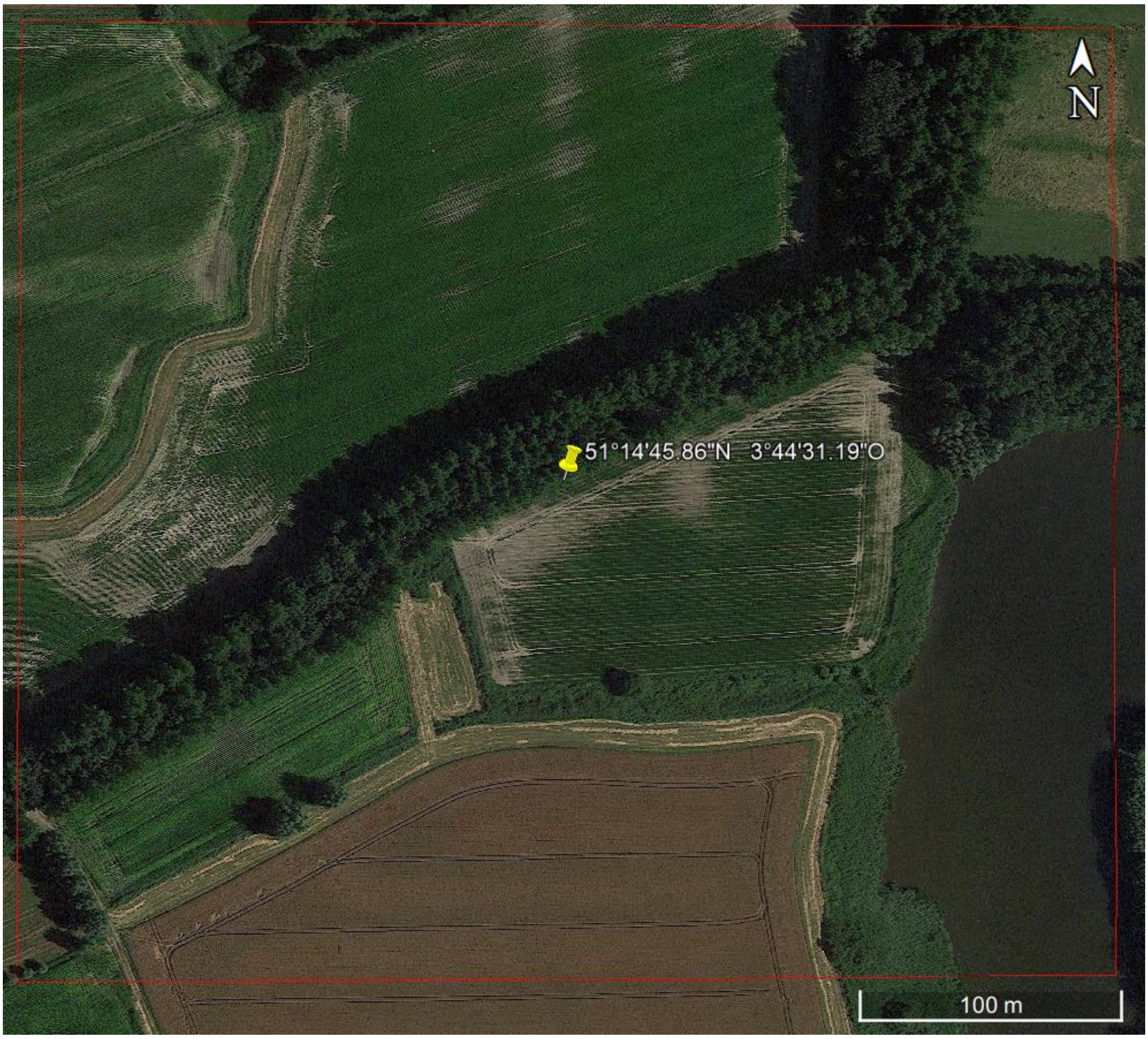


Figure 4: Study area in the North West of Belgium, near Boekenhoute and Assenede, 0,25 km². Both datasets (DHMVI & DHMVII) are on the same location.

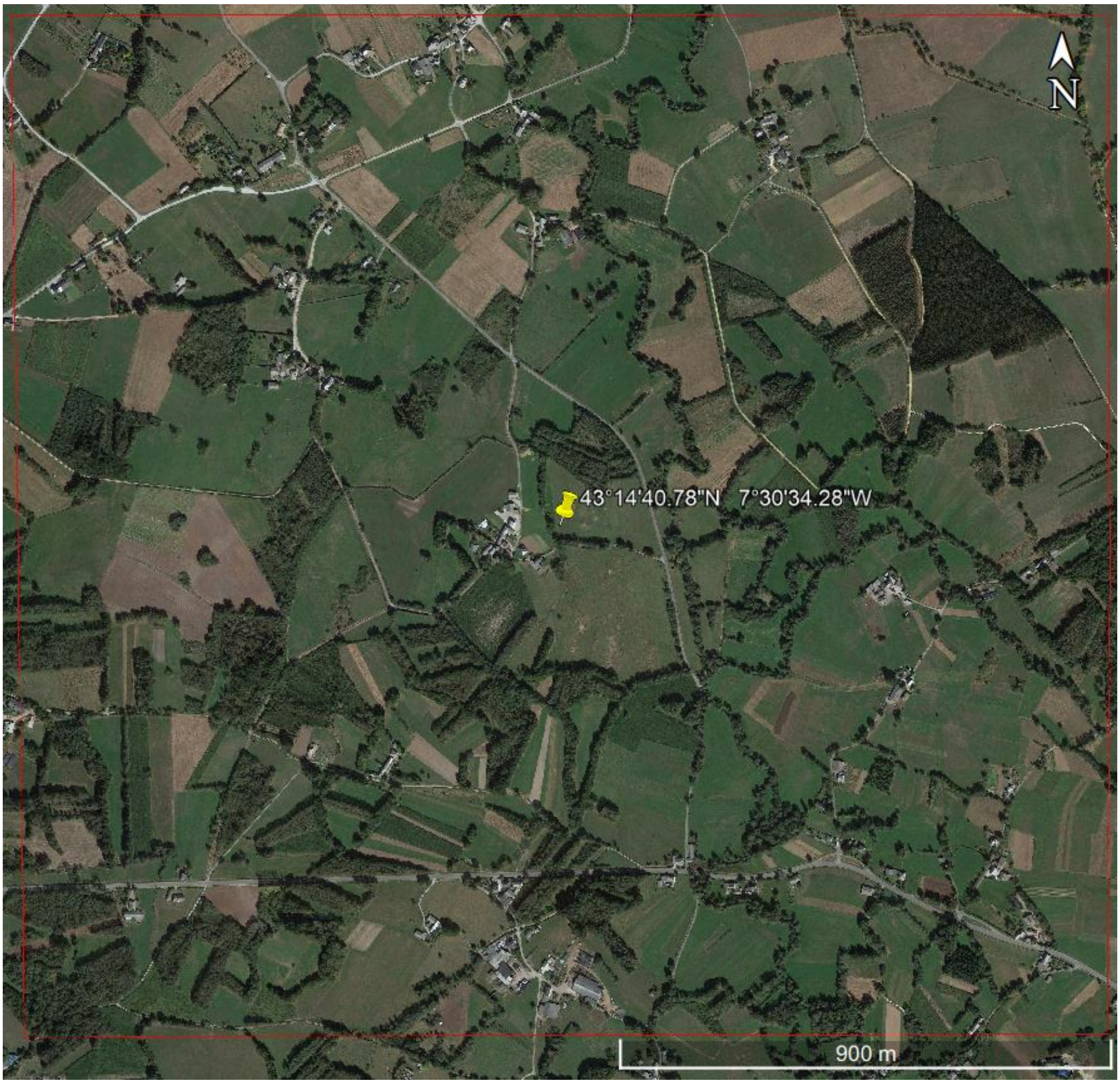


Figure 5: Study area in the North West of Spain, near O Santo, 4 km².

3 Methodology

To calculate spatial features at a continental scale, the linear vegetation element delineation workflow (Lucas, 2017) has been analyzed to determine where bottlenecks may lie (Wescott, 2013). Firstly, a division has been made between computational and data variability bottlenecks. Computational bottlenecks entail concepts which inhibit the general workflow of the methodology due to long waiting times and or resource depletion. With this, calculation speed and resource management are central. The identification of data variability bottlenecks is done by assessing the accuracy of results of the methodology and apply changes where necessary.

Furthermore, the decision has been made to not analyze the full workflow which also includes linear structure identification by region growing. Since the region growing is reliant on the classification performance (Lucas, 2017), effort has been made to optimize the feature calculation and point cloud classification in order to provide a basis for further development, see figure 1.

3.1 Identification of bottlenecks

To identify computational bottlenecks, the original workflow has been summarized in to a technical flow chart instead of a functional workflow to identify key components and to be able to optimize parts of the workflow rather than designing a new workflow. A technical version of the workflow is presented in figure 6.

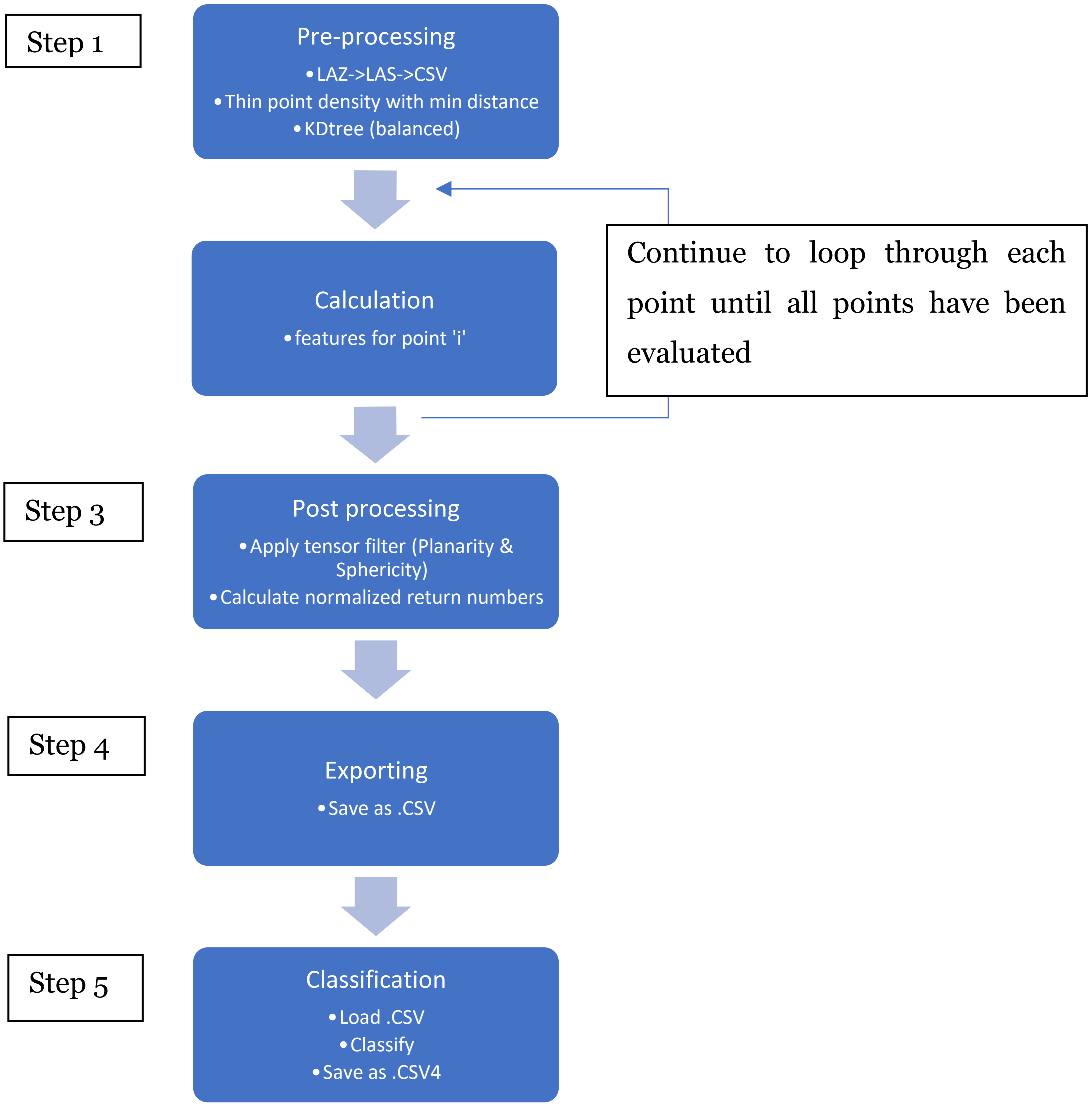


Figure 6: Technical workflow of the processing approach for full LiDAR point cloud feature calculation and classification.

3.1.1 Technical workflow

The flowchart lists five major steps: Pre-processing, calculation, post-processing, export and classification. These steps will be explained in detail in the paragraph below.

Step 1 – Pre-processing

Generally, LiDAR tiles are stored as a .LAZ file, which is a compressed variant of the .LAS file. The LAZ file needs to be uncompressed with the program LASzip, which is part of the LAStools suite (Isenburg, 2018), into a LAS file. Next, the LAS file is transformed into a Comma Separated Values (CSV) file as they are straightforward to load into most programs, as CSV is a very common file type. After conversion, the resulting CSV file is loaded in the program CloudCompare to subsample the point cloud. There are three methods within CloudCompare to subsample: Random, Octree-based and spatial (Girardeau-Montaut, 2015). Spatial subsampling is the methodology employed in the research of Lucas, C. 2017. which lets the user determine the minimum distance between each point. By specifying the minimum distance between each point, a uniform point density across the LiDAR tile is reached. Therefore, in the event of overlapping flight lines, the local point density variation will be removed. After subsampling, the resulting point cloud is saved and exported as a CSV file to be loaded in Python 3.6 (Python Software Foundation, 2016). Within Python, a K-Dimensional tree is built for further calculation. A K-D tree is a multidimensional binary search tree used for retrieving information through associative searches (Bentley, 1975). By constructing a K-D tree from the point cloud, each point can be assessed for its neighboring points.

Step 2 - Calculation

During the calculation phase – the KD Tree is used to retrieve the K closest neighbors. Based upon the relative location of each neighbor, all features can be calculated. The calculation phase has sequential approach – which means that each point is sequentially being evaluated.

Step 3 – Post-processing

After the calculation phase – a tensor filter is applied which removes points which do not meet the Sphericity and Planarity (filter) criteria. These points are deemed irrelevant as they are most likely not vegetation (Lucas, 2017). For the remainder of the points, the normalized returns are calculated.

Step 4 - Exporting

The point cloud with features is exported as a .CSV file for further processing.

Step 5 - Classification

During the classification phase, the .CSV file is first loaded in to CloudCompare. In CloudCompare, the user must manually segment a part of the point cloud for training data, separating the original point cloud in to two classes: Vegetation and non-vegetation. The training data is then exported as a .CSV and loaded back into Python to be used in the training of a balanced Random Forest (RF) classifier. Before the training occurs, a feature correlation check is done to remove features with a high correlation to reduce the number of features to use in the classification. The RF is then subsequently used in the classification of the remaining point cloud, which receives a 0 or 1 to indicate vegetation (0) / non-vegetation (1) (Lucas, 2017). Once the point cloud is classified, it is exported to a .CSV file again to be used in linear vegetation identification.

3.1.2 Computational performance analysis

Firstly, by producing a chart of the time required of each process of the workflow, an initial indication of bottlenecks can be found after which a decision can be made on which part of the workflow to focus and to optimize. Furthermore, by monitoring the CPU and RAM usage, further identification of possible component optimization can be achieved.

The tests are performed in duplicate per methodology. Each methodology is tested on the same area and on an equal amount of points. The data used is from the Dutch national dataset AHN3, see table 1. The complete LiDAR tile covers 37,5 km^2 in The Netherlands, region Beesd Geldermalsen, of which 10e6 points are used (0.625 km^2) in the tests. The time and resource consumption of a specific processing methodology have been visualized by producing a bar chart. The bar chart lists multiple processing steps alongside with the time it took.

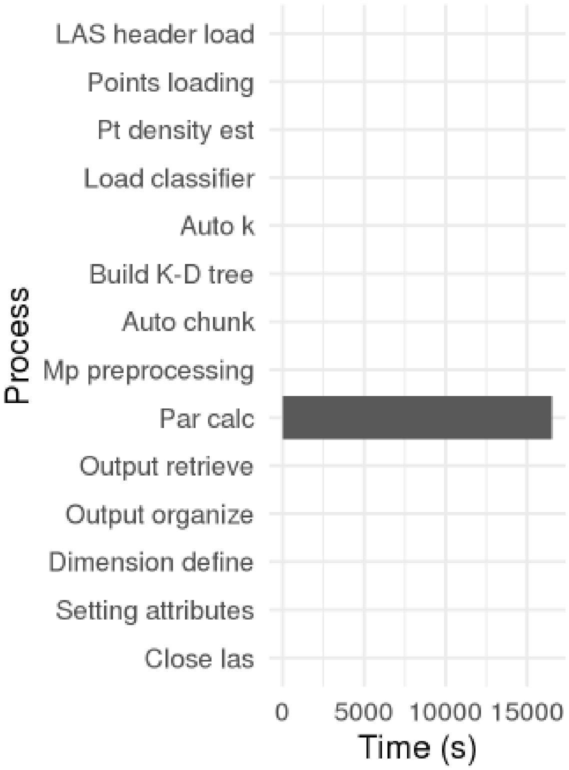


Figure 7: Measurement of the duration for the feature calculation of the old workflow. The workflow has several processing steps displayed in the Y-axis. The corresponding duration of each process is measured.

The calculation process is the biggest bottleneck of the workflow as there are 12 features being calculated, per point. Therefore, attention is first directed towards the optimization of the feature calculation. Once the feature calculation is optimized, new bottlenecks can be identified and adapted as necessary.

If the adaptations to the workflow are applied based upon the hardware and data properties, with the aim of achieving the maximum potential for a piece of hardware, the chance of successfully applying the same concept to a different machine will probably be large as there are no hard-coded rules. If the concept is transferable to different machines, it can be applied at better and or larger computers which will then also be pushed to their maximum capabilities, possibly yielding better results in terms of speed. A higher speed will probably also imply a higher feasibility towards processing at a continental scale.

The workflow, displayed in figure 6, will be evaluated by calculating and classifying all features for ten million points. The area coverage depends on the point density of the LiDAR tile. In case of 10 points / m^2 this equates to 0,625 km^2. The research area on which the original workflow was developed on covered an area of about 2 km^2 with an average point density (after thinning) of 9 points/$m^{2.}$ (Lucas, 2017)$^{.}$ As Europe has a surface area of 10,180,000 km^2 the test size is only a small fraction of a continent. However, as multiple tests need to be executed, the decision has been made to make the test size relatively small compared to processing on a continental scale. The CPU and RAM usage as well as time required per process will be recorded. When a major change to the workflow has been implemented, the same test will be re-run to evaluate the changes. This will provide a clear overview of the progression of optimization and will make it easier to decide which process to optimize next.

3.1.3 Hardware and software description

The workflow adaptations have been tested on a server supplied by the eEcoLiDAR research group. The software used is Python 3.6, CloudCompare and LAStools. A detailed package list is available in appendix D, alongside with server specifications.

3.2 Data variability

When the workflow is applied to a continental scale, it is inevitable that the workflow needs to handle different LiDAR datasets from various data suppliers, as there is no uniform continental scale dataset available for now. A large difference between datasets is the number of points per square meter.

3.2.1 Search radius

LiDAR tiles can differ locally and between tiles in point density. In general, point density differences are not an issue in processing data because it only dictates the area coverage, which is irrelevant in terms of processing speed. However, in terms of the classification of objects, point density does matter. Since point density determines how many points are within an area

– the amount of points to describe an object will also differ. For instance, an object such as a tree has a diameter of for example 5 meters. If we have a low point density of 1 point/m^2 we will only have 19-20 (19.6) points describing the tree, while at a higher point density (20 points/m^2), the number of points representing the same tree increases to 392 points. The RF classifier classifies points based upon their feature values. These feature values are derived from the neighboring points. This is done is by retrieving several (indicated by a K) neighbors around a point which is are then evaluated for their shape as a group. Currently, the number of neighbors (K) is a hard-coded number found by optimizing for a specific point density (Lucas, 2017). As the point density between countries and even between tiles within the country can vary – a rule replacing the fixed number of neighbors must be found. If the number of neighbors remains static, the search radius will increase with lower point densities and will incorporate multiple objects of different classes.

3.2.2 Classification

The point classification in to either vegetation or non-vegetation classes is done with a Random Forest (RF) classifier (Breiman, 2001). The methodology for classification is referred to as supervised classification, as the user needs to train the model beforehand with test data (Loog, 2018). By selecting points and annotating them to signify to which class they belong, we can train the classifier to classify points with similar features to the same class, see figure 6 step 5. The performance of the classifier is dependent on the quality of the training data.

The workflow presented in figure 6 has incorporated the training of the classifier in step 5. However, if the aim is to upscale, we want to circumvent the supervision for each tile of a continent as this is a time consuming and laborious task. This means that, once we have a trained classifier, it should be applied to all different tiles rather than re-training a new classifier for each different tile or dataset. Therefore, a method of saving and loading a trained classifier needs to be added.

3.2.3 Accuracy assessment

First, features for every point in each tile will be calculated for the 50 closest neighbors (K = 50) and in cases of low point densities, multiple K-values are used. Secondly, each tile will be divided into two classes: Vegetation and Non-Vegetation (Lucas, 2017). The manual segmentation will be done in CloudCompare with the "segment" tool (Girardeau-Montaut, 2018). Each LiDAR tile used for annotating will have its points visualized with a single-color (white). This is done to decrease manual segmentation bias, as this ensures the segmentation is based on shape and not on color. In LiDAR tiles with low point density (<5.0 points/m^2) the visualization will be slightly adjusted as the point thickness will be increased. This allows for zooming in without leaving gaps between the points in CloudCompare and will not affect the data as this is only a change in visualization, not processing. All points will be manually segmented with a top-down view because the segmenting tool creates an infinite cylinder

(Girardeau-Montaut, 2018). This means that, if it were to be done from a side-view, large amounts of points would be grouped together. Once manually segmented, these two classes are merged back together and receive a reference class column indicating the user segmentation (validation set). Next, each point will be classified using a Random Forest classifier (Lucas, 2017). After the classification, confusion matrix metrics are calculated by determining the difference between the Random Forest classified column and the reference (manually annotated) class column. Each point receives a value to indicate if the classification was a True Negative (TN), True Positive (TP), False Negative (FN) or False Positive (FP). A true positive case is where a class has been classified as '1' (positive) and the reference class also indicates '1' for that point. A True Negative is the same as a True Positive but then for the class '0' (negative) (Fawcett, 2006). The class '0' represents vegetation whereas the class '1' represents non-vegetation. The dataset can now be visualized and visually inspected (as well as numerically) to determine which cases cause a discrepancy between the reference class and the classification result.

3.2.4 Classifier transferability

To effectively classify point clouds for large number of tiles, the Random Forest classifier needs to be portable, so it can be applied on varying datasets instead of the dataset on which it was trained. After the initial classifier is trained on a valid dataset it can be saved and loaded along with its feature list. The feature list can be loaded to determine the different features to be calculated for a certain classifier.

Denmark, The Netherlands, Spain and Belgium (DHMVI & DHMVII) have had a tile evaluated. The full accuracy assessment tables are stored in the Appendix B. Each dataset evaluated has a tensor filter (Planarity and Sphericity), abbreviated as SPf, ON/OFF mode to show the effect of applying a filter to the processing workflow.

The Random Forest classifier is trained on the AHN3 dataset in the region between Beesd and Geldermalsen, see figure 3. The classification has been applied to Danmark: Vamdrup, Belgium: Assenede, The Netherlands: Beesd and Spain: O Santo, see figure 2-5.

4 Results

4.1 Computational performance

The new workflow, presented in figure 8, integrates the classification and tensor filtering inside the calculation phase. Originally, the classification and filtering were done post calculation. However, the classification and filtering has been added during the calculation phase to avoid the saving and loading of intermediate files, which might aid in total processing speed gain.

However, these two tasks can be turned off after which the workflow will relate closely to the original workflow.

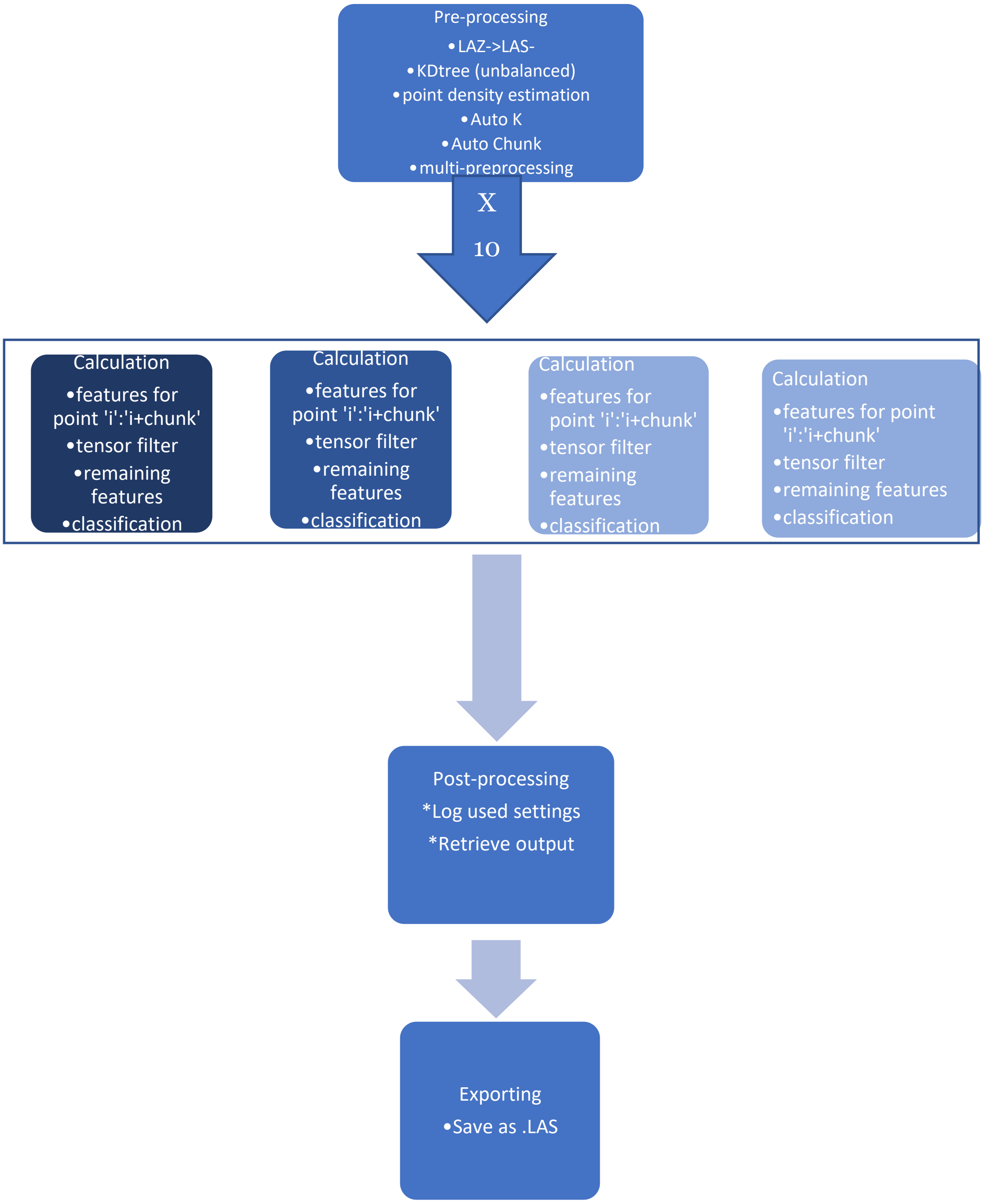


Figure 8: New multiprocessing workflow for continental scale feature calculation and classification.

4.1.1 Computational power

The similarity between LiDAR tiles is that they will all describe the same type of information. A 3-dimensional location and possibly extra attributes. LiDAR data from Spain will be roughly the same as that of The Netherlands. It will, however, differ in coverage and resolution. The coverage and resolution depict how many points are within a certain file for a certain area. Therefore, the difference between countries will not necessarily be a different data type but more the amount of points used to describe the surface flown over. This means that if a certain methodology that is capable calculation of points at a certain speed (points / time) it can do that for other countries as well. Therefore, if we can calculate 20 features for 1 million points within 10 seconds, the same can be applied to a different dataset.

Originally, the workflow of vegetation classification was developed with a sequential approach. The sequential approach meaning: each point is sequentially being evaluated. This causes the program to perform a search action for each individual point, retrieves the relevant information and calculates the varying metrics for the point. The program will repeat the same process for the next point until all points are evaluated and are supplied with the descriptive features.

The resulting time (s) and resource consumption for the sequential methodology is displayed in Figure 9. The resource consumption of the processor is a percentage based upon the total number of processors. As there are 40 logical processor cores, a processor use of 2.5% equates to one core being used.

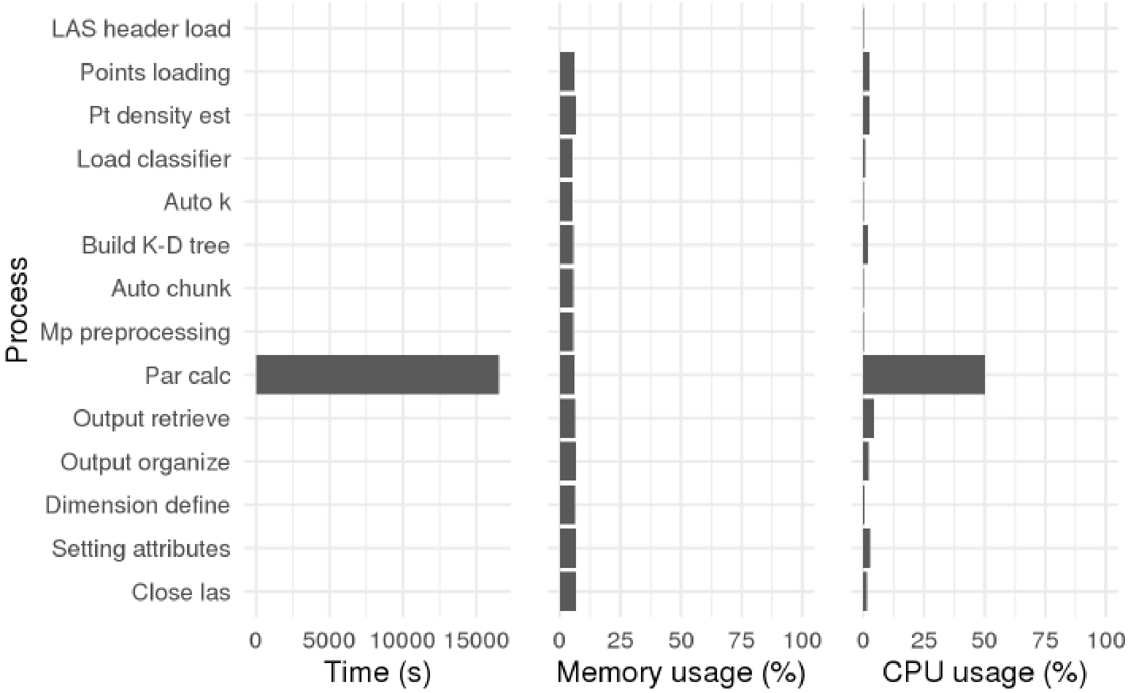


Figure 9: Resource consumption and duration for 1 core and no vectorizing calculations.

The resource consumption of memory is below 10% whereas the CPU usage is 50%. The resulting time for 'par_calc' which indicates the parameter calculation stage – is 4,5 hours (16.200 seconds). On average, the calculation phase did 600 iterations / s which equates to 600 points per second being evaluated. The largest bottleneck is the 'Parameter calculation' process as the other operations take only a fraction of the time required by the parameter calculation.

4.1.2 Sequential versus Vectorized

In general, calculating multiple values at once, using a vectorized approach, is faster in calculation time as compared to evaluating each individual point, called looping (Birkland, 2012; Bucher, 1983; Coffin, 2016). However, a loop is often preferred as using a vectorized calculation requires more internal memory to be allocated than a single point (Birkland, 2012). In theory, if we can evaluate one point for its neighbors, we should be able to do the same for any amount granted we have the memory to store the input required and perform the operation on. Of course, it matters greatly whether we evaluate the input required for a single point or evaluate the input required for 1000's of points. Each set of input requires to be loaded into the internal memory of the machine which has a finite capacity (Birkland, 2012). However, in order to calculate in a vectorized manner, the shape of your data matters. A lot of built in functions and modules have pre-defined ways of calculating metrics which can inhibit the vectorized calculation.

4.1.3 Vectorized covariance matrix

The main bottleneck of vectorized calculation for the parameter calculation is the computation of a covariance matrix. The covariance matrix is used in the calculation of the eigen values and eigen vectors (Lucas, 2017) which marks one of the very first calculation steps for the whole workflow, see figure 1. Normally, one would use a built-in function from the module SciPy (Scientific Python) in Python, which is a highly rated module developed for mathematics, science and engineering (Jones, et al., 2001). However, SciPy allows you to calculate a single covariance matrix based upon input, rather than vectorizing the manner. Therefore, a different function has been written to deal with this limitation.

The new function uses the function *einsum* from the Python module Numpy (Oliphant, 2006), which allows you to greatly customize how your function handles your input and output data. *"The einsum function evaluates the Einstein summation convention on the operands"* – Numpy documentation. In other words, it allows us to do multi-dimensional array operations. By specifying how the covariance matrix should be calculated, a 3-dimensional output matrix is generated which holds a covariance matrix per evaluation. The amount of covariance matrices to calculate is variable and will not change the further calculation steps. Furthermore, the Numpy library can compute eigen values and vectors in a vectorized manner by specifying the third dimension of the covariance matrix as the vector. This causes the Numpy library to treat each entry on the third dimension as a separate covariance matrix and outputs the resulting eigen values and vectors into two, two-dimensional matrices. Where each matrix holds the resulting eigen values or vectors for each covariance matrix.

The remaining parameter calculations are easily converted to a vectorized approach as they are based on simple addition, multiplication and division.

4.1.4 Semi-vectorization

Computing vectorized needs much more internal memory, also called Random Access Memory (RAM), than looping through the data (Birkland, 2012). Therefore, the workflow has been changed to a semi-vectorization approach in which the calculations are vectorized and the size of the input data is controlled. This means that instead of calculating all the metrics over all points to be evaluated simultaneously, input data is split in chunks. The size of these chunks is defined before calculation based upon a Central Processing Unit (CPU) / Random Access Memory (RAM) ratio and task list. For instance, if only the point cloud features need to be calculated and no classification needs to be done, the chunk size doubles compared to when classification is enabled. This is done to ensure enough free memory is available for both calculating and classifying.

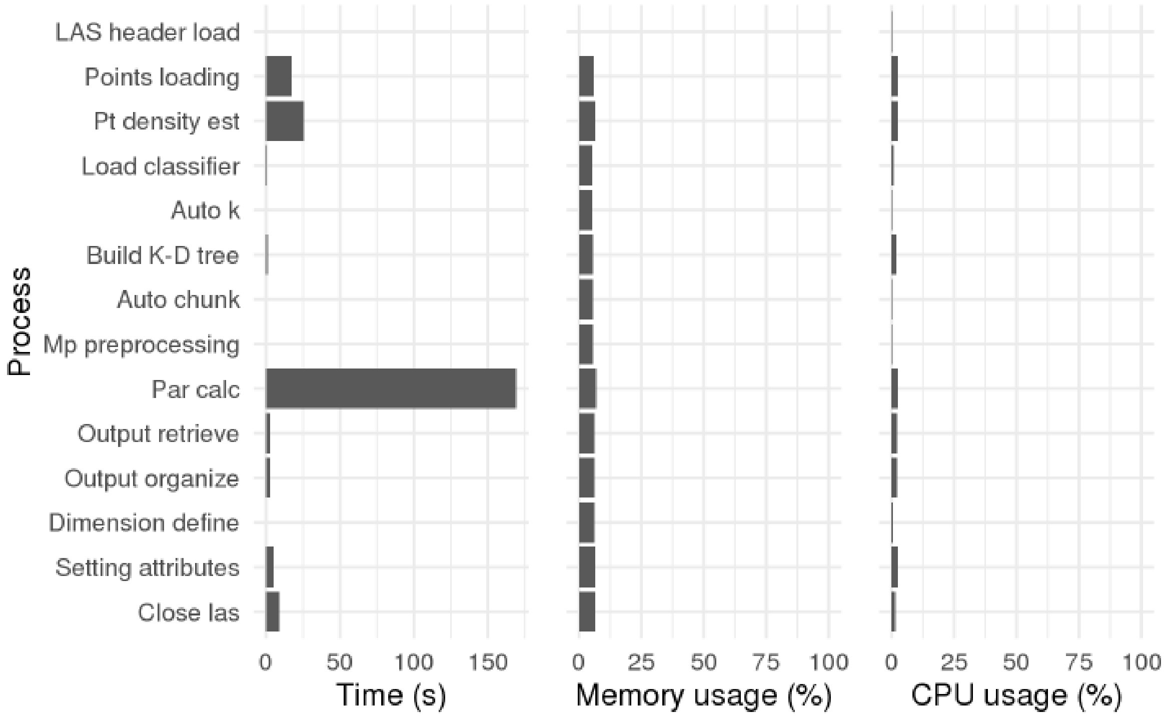


Figure 10: Resource consumption and duration for 1 core with vectorized calculations

As shown by the resource consumption in figure 10, this time with the new vectorized covariance calculation a single processor core is being utilized which corresponds to 2.5% total CPU usage. The memory consumption remains almost the same with a chunk size of 500.000 points. This means that 500.000 points are being evaluated each iteration. As the parameter calculation phase took 170 seconds (2 minutes and 49,8 seconds) and 10 million points are being evaluated, the new calculation method does 0,11 iterations per second. However, considering the chunk size of 500.000, this means that it is evaluating 58.823 points / s compared to the 600 points / s on the calculation without chunks. This is a speed increase of 90x of the amount of points / s being evaluated.

Table 2: Processing time for the Sequential and Single core vectorized methodology.

Method	Time (s)	Points/s
Sequential	16.666	600
Single core vectorized	170	58.823

4.1.5 Parallelization of the feature calculation

Once the calculation workflow is optimized another limitation must be overcome, which is the Global Interpreter Lock (GIL) of Python. The GIL ensures that when calculations are done within Python only a single processor-thread is being used to do all the calculations required (Wouters, 2017). The GIL can be bypassed by introducing a small addition of code which tells Python to lose the lock and allow for multiple threads to be created. Because now we are not limited to a single thread, the workflow can be adjusted for parallelization.

Parallelization of the workflow entails that it issues the exact same calculation instructions on different sets of data simultaneously (Nishihara, 2018). As the input data (LiDAR) is very homogenous (has x, y, z information for every point) we can estimate the required RAM for a set of calculations. In fact, by determining the maximum possible processor-threads to spawn and the available memory, we can allocate a memory size to each thread to ensure that we are not requesting more memory then we have available. First, the amount of threads needs to be determined. Often, a rule of thumb is not to spawn more threads than physical CPU-cores. As spawning more threads than physical cores causes the operating system to allocate calculation time to specific threads and therefore shift the threads from core to core. This causes the machine to have more overhead time (managing threads) and impacts the computation time negatively. Many of the current processing chips (containing multiple processor-cores) can split their physical core in to multiple logical cores. This is called hyper-threading and is widely available on recent computers and laptops (Stokes, 2002). In case of hyper-threading, you can allow twice as many threads as physical cores as the computer chip is built to handle two threads per physical core. By sending the exact same, but on different input data, calculation requests across all the processor resources, significant speed up can be achieved. Where previously a single core would be utilized to its maximum potential, a laptop with 4 cores can reach a significant speed-up when utilizing all cores as opposed to one. The same holds true for more powerful desktop personal computers and even more so for datacenter servers with up to, and possibly even exceeding, 20 physical cores. In fact, by aiming to utilize the full hardware capacity of a single machine, the software becomes extremely scalable.

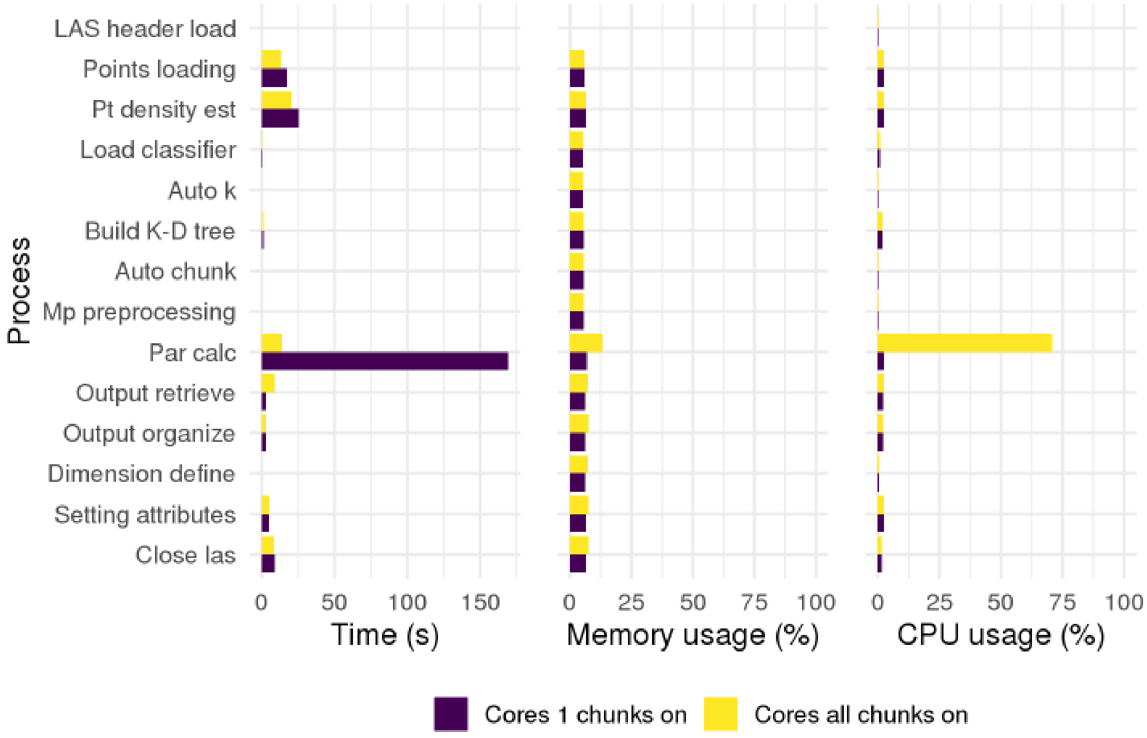


Figure 11: Resource consumption and duration for 1 core with vectorized calculations vs multiprocessing with vectorized calculations.

The memory consumption has increased considerably accordingly. When enabling all cores in combination with vectorized calculations, the calculation time lowers to 14 seconds. The multiprocessing vectorized approach evaluates 714.285 points / s on average, which is a speed increase of 13 x compared to the single core with chunks and 1.190 x with the multiple cores without the vectorized covariance calculation.

Table 3: Processing time for the Sequential, Single and Multi-core vectorized methodology.

Method	Time(s)	Points/s
Sequential	16,666	600
Single core vectorized	170	58.823
Multi core vectorized	14	714.285

4.1.6 Integration of Classification and Tensor filtering

The classification of points in a point cloud follows from the calculated descriptive features. Based upon these features, a filtering of unwanted points can be introduced. By filtering on a Planarity and Sphericity value, a large amount of unwanted points (for the classification of vegetation) can be removed (Lucas, 2017).Therefore, the tensor filter has been introduced as one of the first steps, during calculation. The calculation phase first calculates all the necessary values to determine the Planarity and Sphericity prior to calculating the remaining features. As soon as the Planarity and Sphericity values are calculated, a filter is applied to determine which points need to be removed. By removing these unnecessary points immediately, calculation time can be saved by avoiding unnecessary calculations on points which would be filtered out eventually. This has as a benefit that it reduces the output file size and of memory usage during calculations and post processing. In some cases, up to 96% of the points can be removed by applying a tensor filter, see Appendix B.

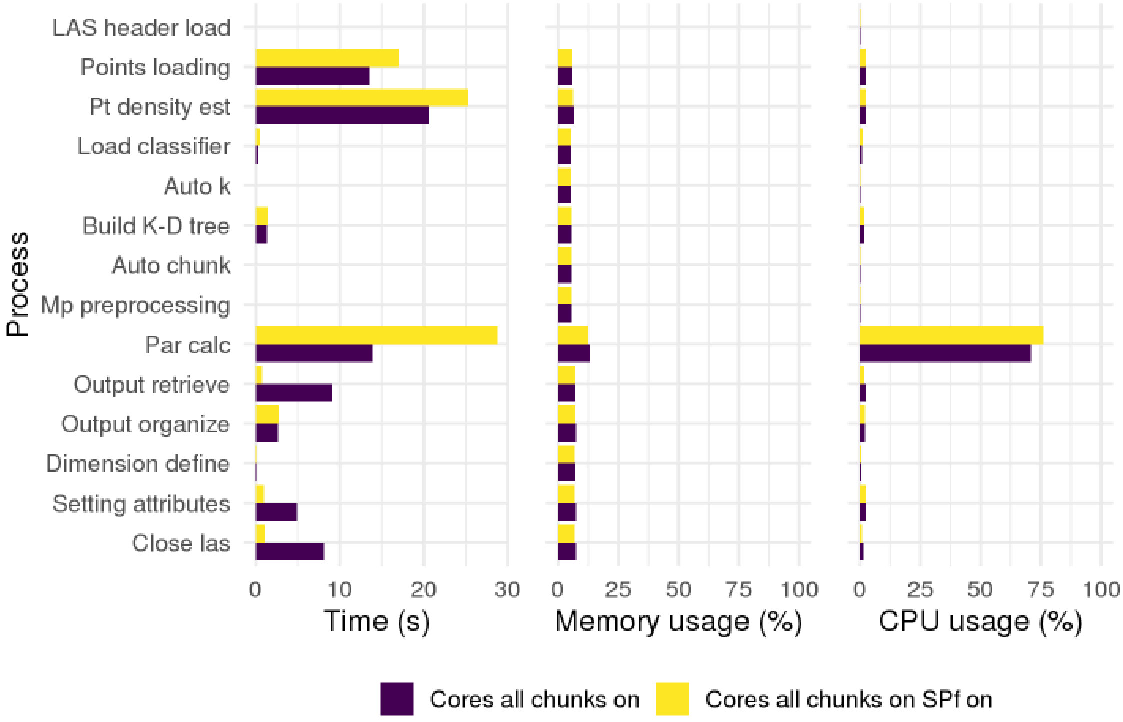


Figure 12: Resource consumption and duration for multiprocessing with a Planarity & Sphericity filter, and without.

The calculation time doubles when the filter is being applied. Nonetheless, even though it is slowing down the calculation phase, it does circumvent post-processing filtering, see figure 6. The classification step has also been introduced during the calculation phase ('parameter calculation'), see figure 6. This ensures that the classification of each chunk of points during calculation occurs within a certain thread. This has as a benefit that there is no intermediate saving and loading of points which saves time on the total processing workflow.

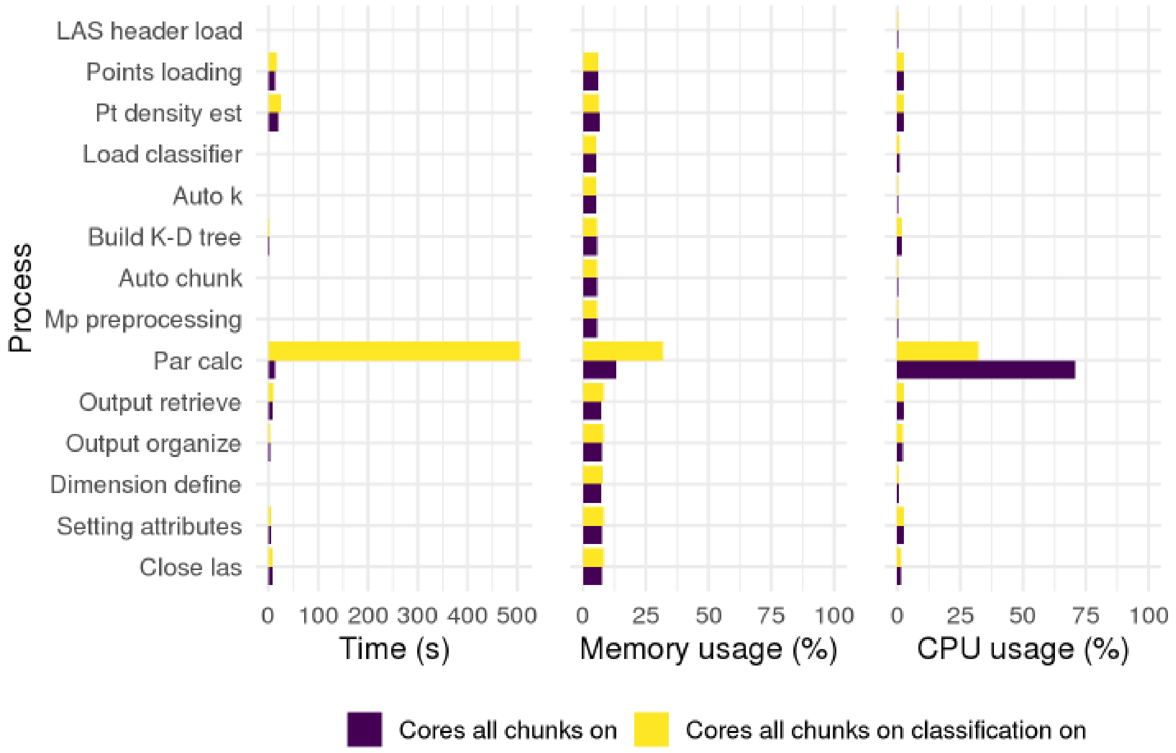


Figure 13: Resource consumption and duration comparison between classifying during calculation stage and not.

However, introducing the classification step during the calculation phase significantly slows down (35x) the calculation stage. Some increased calculation time is expected, as in general, more operations will require more time. However, this is a factor 35x slower, which indicates a new bottleneck.

4.1.7 K-Dimensional tree construction

The K-Dimensional tree is an integral step within the calculation of point cloud features. The building of the KD tree is currently recursive and therefore a single thread bound operation (Brown, 2015). This means that, due to the nature of creating a recursive KD tree, it cannot be speeded up by making multiple parallel KD Trees nor can multiple threads contribute to building one. Instead, to overcome this the method of building the KD Tree has been slightly changed. By default, a KD Tree is built by using compact nodes and rebalancing the KD tree (Pedregosa, et al., 2011). This allows for heterogeneous data to be accurately placed in an efficient data structure at the cost of building time. However, as point cloud data is very homogenous, an unbalanced KD tree can be built without losing the precision of the balanced KD tree, see Appendix B, figure 22 and 23. The unbalanced KD tree with large nodes builds many magnitudes faster and has almost no increase in query time. As the rest of the workflow is written for parallelization, these queries in turn also become parallel and suffer no calculation speed.

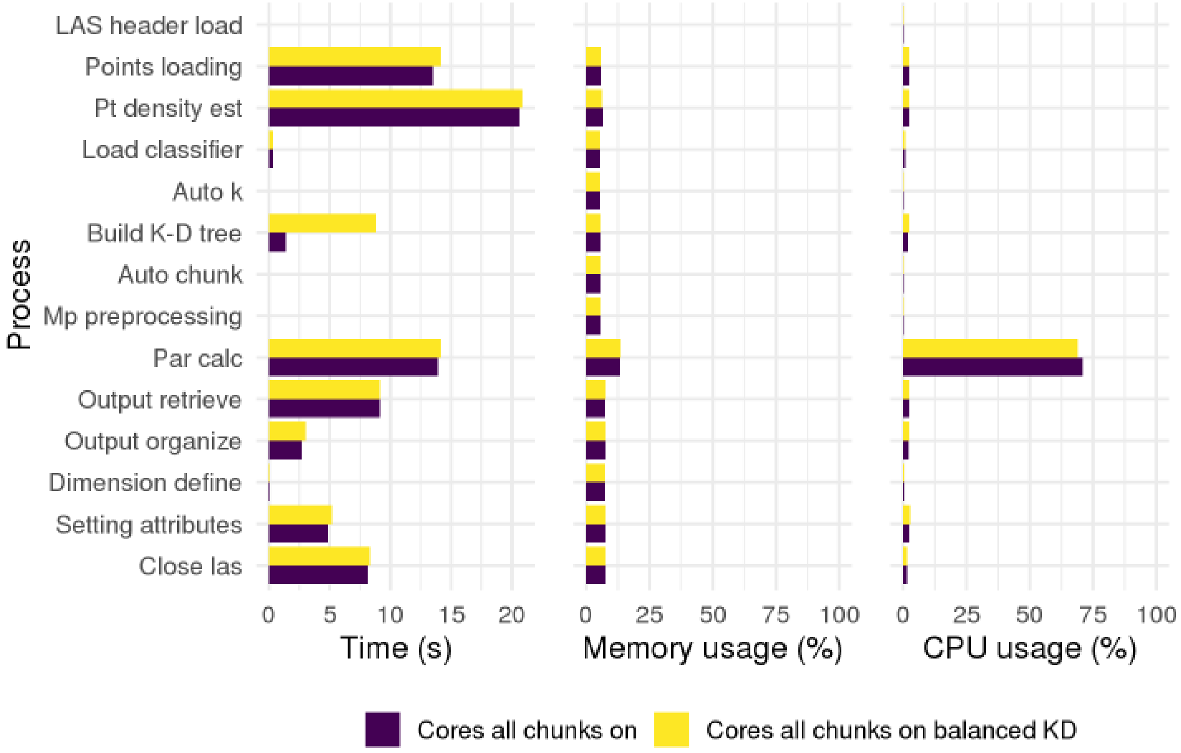


Figure 14: Resource consumption and duration comparison between using a balanced vs unbalanced KD tree.

As can be seen from the resource consumption in the '*Build K-D tree*' process, the unbalanced KD tree builds more than 5 x as fast compared to the balanced KD tree.

4.1.8 Parameter calculation versus Total processing time

The speed gain of the parameter calculation is as previously stated, is 1190 times. However, when we compare the processing speed gain of the total workflow, seen in figure 15, the speed gain is much lower. Furthermore, an important result can be seen from figure 15. When comparing the processing speed of the methodology which has a Planarity and Sphericity filter (SPf) (*Cores all chunks on SPf on*) to the methodology without (*Cores all chunks on*) the total workflow speed gain is almost the same (175x). The total run time for the methodology without a SPf is 93 seconds whereas the methodology with a SPf is 89 seconds. This is due to the lower time requirement for exporting the data, see figure 12. As the SPf consumes more time to calculate, it does significantly reduce the total amount of data to be exported.

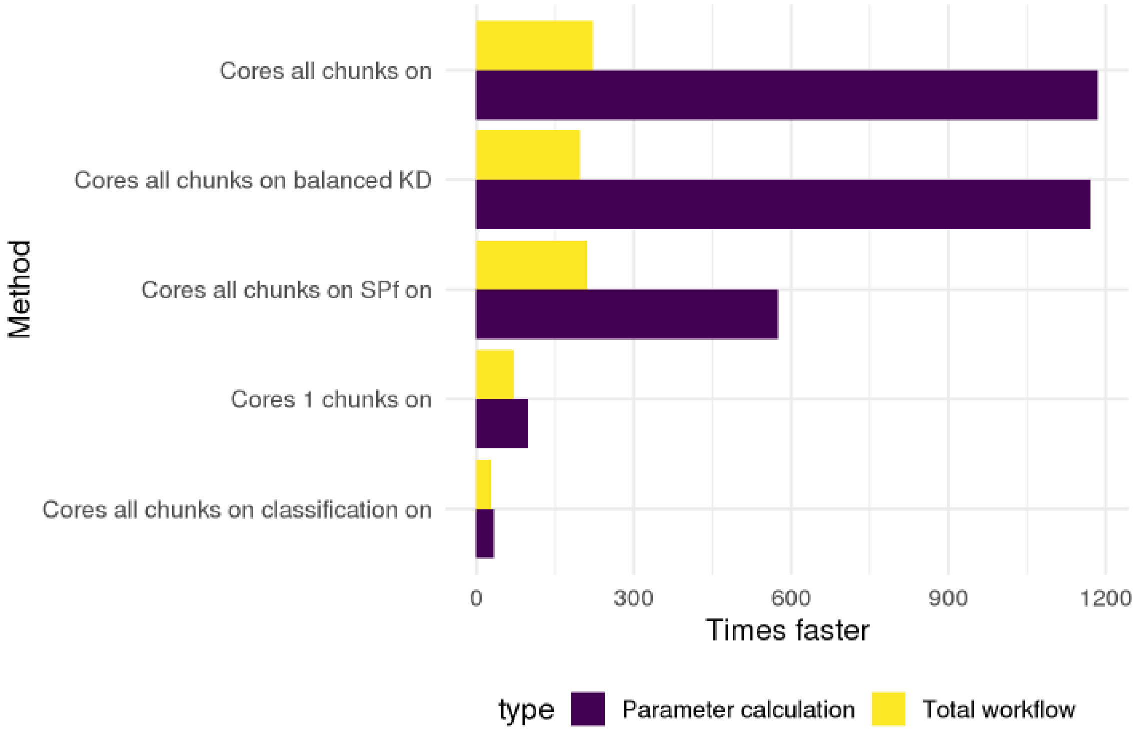


Figure 15: Parameter calculation speed gain versus total workflow time. Speed gain is compared to the sequential calculation methodology.

4.2 Model application – Accuracy assessment

The countries and tiles mentioned in the Data section have had an accuracy assessment. These accuracy assessments are in Appendix B. However, not all accuracy assessments have been visualized as a map as the visualizations are used to high-light specific functionalities and issues. As such, visualizations have been made for: The Netherlands, Denmark and Spain. Belgium's DHMVI & DHMVII datasets are comparable to Spain's and The Netherlands datasets and therefore have not been visualized.

4.2.1 Baseline scenario – Beesd Geldermalsen

Firstly, a classification of region Beesd Geldermalsen has been made. This is a region in The Netherlands where the original workflow has been developed on (Lucas, 2017). The difference between this classification and the original one is that for this research a new Balanced Random Forest classifier has been trained with a new training data set in order to, in the event of anomalies, be able to back track the full workflow. Afterwards, the classifier has been applied on the same dataset but this time without subsampling the point cloud, as the motivation for subsampling the point cloud was to reduce calculation time (Lucas, 2017).

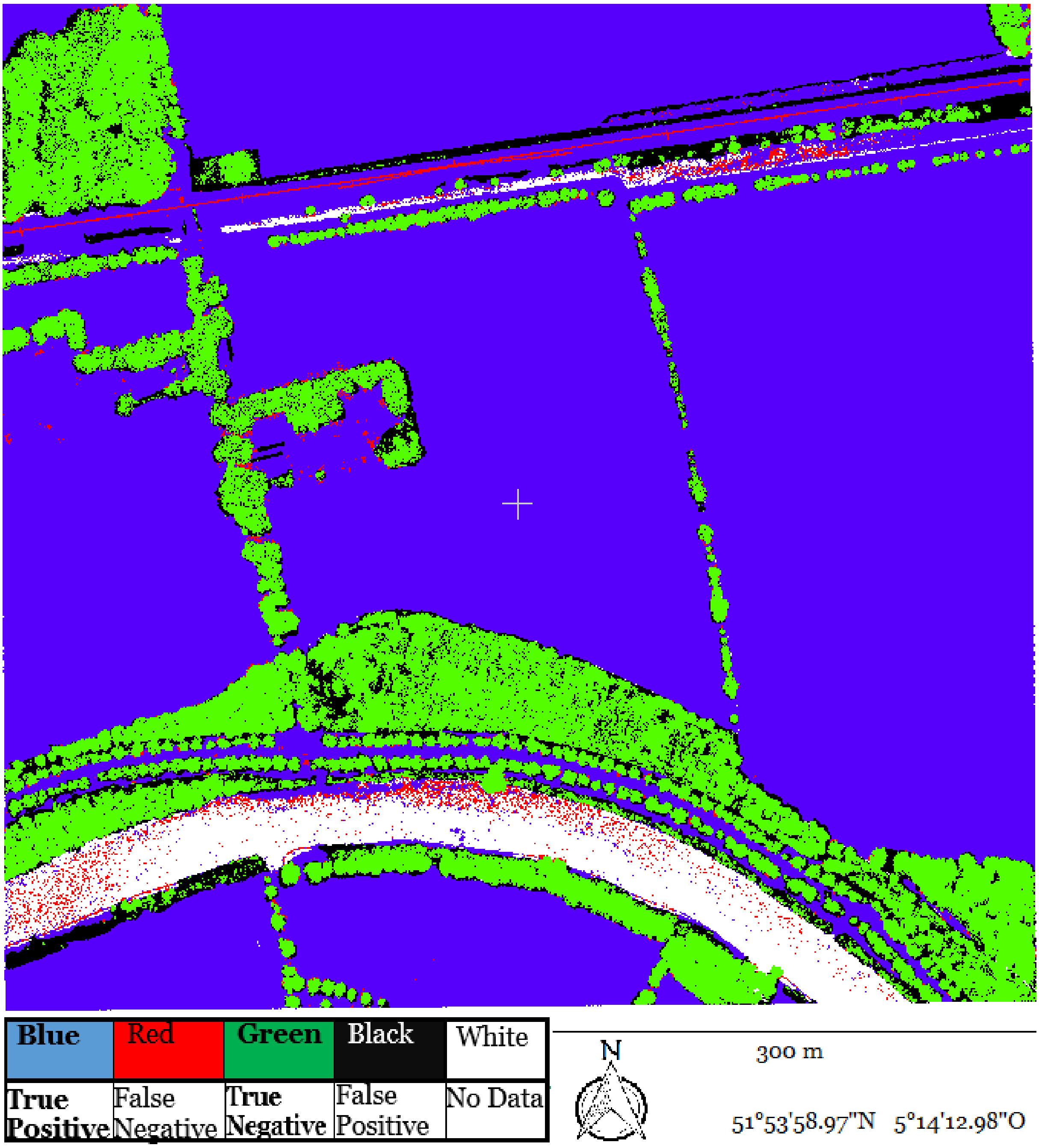


Figure 16: Classification accuracy for Beesd – tensor filter OFF – K50 – 16 points/ m²

The reported accuracy for this tile is 86.33%, see Appendix B, table 6. The largest false negative classification occurs on road-side vegetation (North-East) and riparian vegetation (South-West). This might be caused by the initial training of the dataset which is more biased towards larger vegetation as these groups of points are easier to recognize. Furthermore, the riparian and road-side vegetation are generally low in height and have a patchy distribution. This causes issues for manual segmentation as it is hard to distinguish where low vegetation continues or stops causing grouping of unwanted points. Other than the road-side and riparian vegetation the general classification performed generally well. We can see some misclassification in larger forested areas (North-West) where the ground points underneath the trees are labeled as a misclassification where it is in fact a wrong manual segmentation. However, even though it causes the accuracy to decline – it proves that the performance of the classifier is actually very good. Noteworthy is that the reported accuracy is a different accuracy value than the cross-fold-validation accuracy obtained during the training of the RF classifier. The classifier also has been evaluated on its training data by using a 10-fold-crossvalidation, for which it receives a score of 99%, see Appendix B, table 23. There is a difference in evaluation strategy between the two accuracies which is why there is a different score for accuracy between the two strategies. Conventionally, a 10-fold cross validation is used to determine the accuracy of the classifier. This is done by splitting the data 10 times into a training and validation set, after which a classifier is trained on the training data and evaluated on the validation set. The variance between the 10-fold cross validation results is a measure of model performance where a lower variance indicates that the model behaves relatively the same across multiple sets of training data. This can indicate that the training data is sufficient and that the model does not seem to overfit. The accuracy assessment as described in the 'Accuracy assessment' paragraph uses two (or more) different annotated data sets; one initial training dataset (Beesd) and a validation set per LiDAR tile. This means that the model is tested on an annotated dataset which spatially differs from the training dataset. From here on, only the accuracy of the accuracy assessment strategy described in *Accuracy assessment* paragraph will be used.

4.2.2 Foreign dataset - Denmark

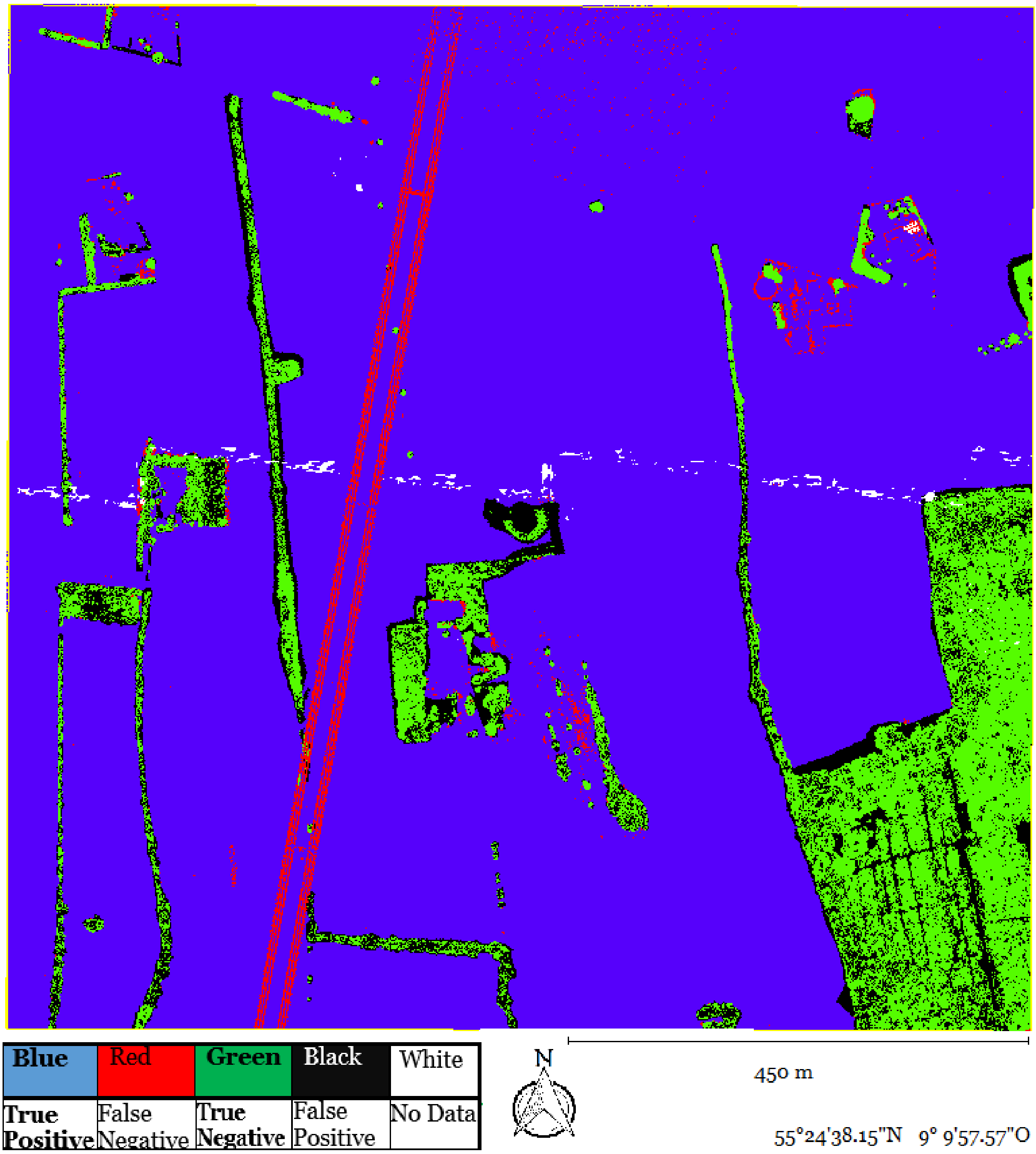


Figure 17: Classification accuracy for Vamdrup –tensor filter OFF - K50 – 20 points/m².

From South to North a misclassification of a powerline occurs. Furthermore, there is some misclassification with a few buildings in the North East, as they are seemingly identified as vegetation. In the North, we can see some red dots around the field which probably should have been classified as non-vegetation, see figure 2. Lastly, from West to East, there is some data lacking (white) which does not seem to correspond to any real-world objects and might have been an issue during measurements or post processing. The accuracy of the classifier for this specific tile is 90.55%. However, as accuracy is determined by the accuracy of both classes combined, it does not mean that both classes perform equally well. Prevalence of a class is the measure of representation of a class. The representation being the number of reference class (this is a user determined measure) points divided by the total number of reference class points. The prevalence of reference class 0 (vegetation) is 21.27 % whereas reference class 1 (non-vegetation) has a prevalence of 78.73%. Now, if we look at the accuracy of each class, i.e. the True Positive rate and the True Negative rate (specificity), then we can see that for class 1 a 99.50% TPR is reported whereas class 0 received a score of 57.45%. This means that in only half of the cases, there was an agreement on points being vegetation. However, when visualizing the points with their attributes (TN, TP, FN, FP) we can see some interesting behavior.

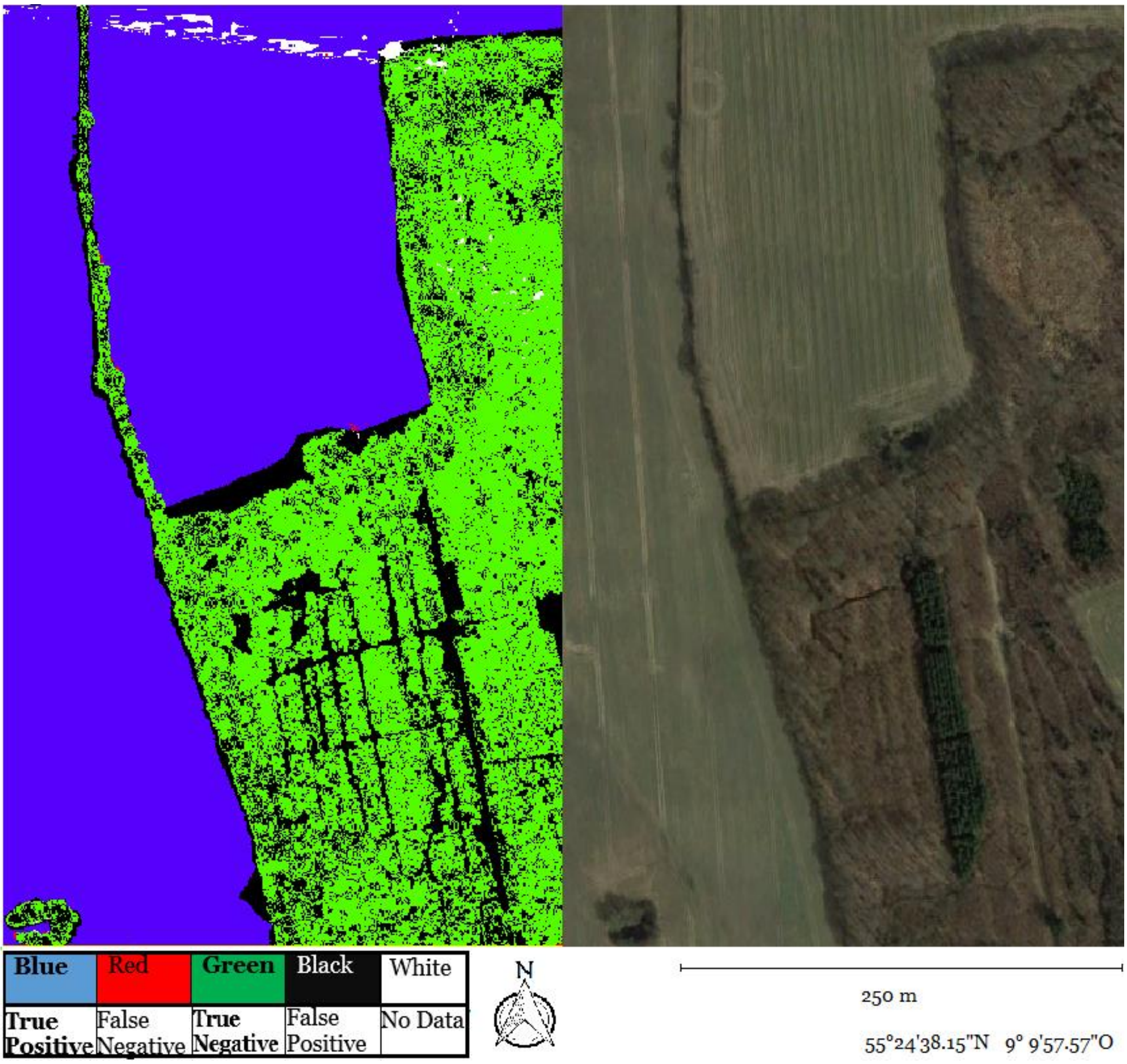


Figure 18: Classification accuracy for Vamdrup, an example of wrong annotation data.

What can be seen from the figure is a user mistake, causing the specificity to decline. The area shown has been marked as a group of points all belonging to the vegetation class. The black spots inside the group of points seem to be ground points.

And as confirmed by the RGB composite of the exact same area, there are lines (light green) between the vegetation where it should have been ground points. Therefore, the group of points should not have been grouped together as one reference class but instead have been divided into smaller sections to extract the vegetation from the non-vegetation.

The cause of the incorporation of ground points is due to the methodology of segmentation. Manual segmentation is done by drawing a cylinder with an infinite length around a group of points, extracting whatever is within the cylinder. This causes ground points to be frequently grouped to vegetation, whereas they should be separated from the vegetation class, see Appendix C for an example. In general, the classifier performed extremely well on a foreign dataset, much better than the manually annotation.

Applying a tensor filter on the same dataset filters most of the ground points and the powerlines. The resulting changes are displayed in table 4 and figure 19.

Table 4: Classification accuracy differences between the original classification and applying a Planarity and Sphericity filter.

Filter OFF			Removal	Filter ON		
Class	Accuracy	Prevalence		Class	Accuracy	Prevalence
0	57.45%	21.27%	17%	0	76.70%	95.60%
1	99.50%	78.73%	79%	1	81.12%	4.40%
Combined	90.55%	100%	96%	Combined	76.89%	100%

This also becomes evident when viewing the new prevalence of class 1, which lowered from 78.73% to 4.41%. The prevalence of the vegetation class has gone from 21.27% to 95.60%. The accuracy has now decreased from 90.55% to 76.89%. This is understandable considering the combined accuracy of the classes determines the total accuracy. With the tensor filter applied, 11.104.782 (96%) points have been removed and the accuracy of the non-vegetation class (1) has decreased from to 99.50% to 81.12%. The vegetation class (0) has had its accuracy increased from 57.45% to 76.70% as the tensor filter filtered 268.503 (17%) points from the vegetation (0) class. The accuracy is a weighted accuracy based upon both classes. Considering that the best performing class has its prevalence reduced from 78.73% to 4.41%, the high accuracy won’t contribute as much to the total accuracy as it did before filtering.

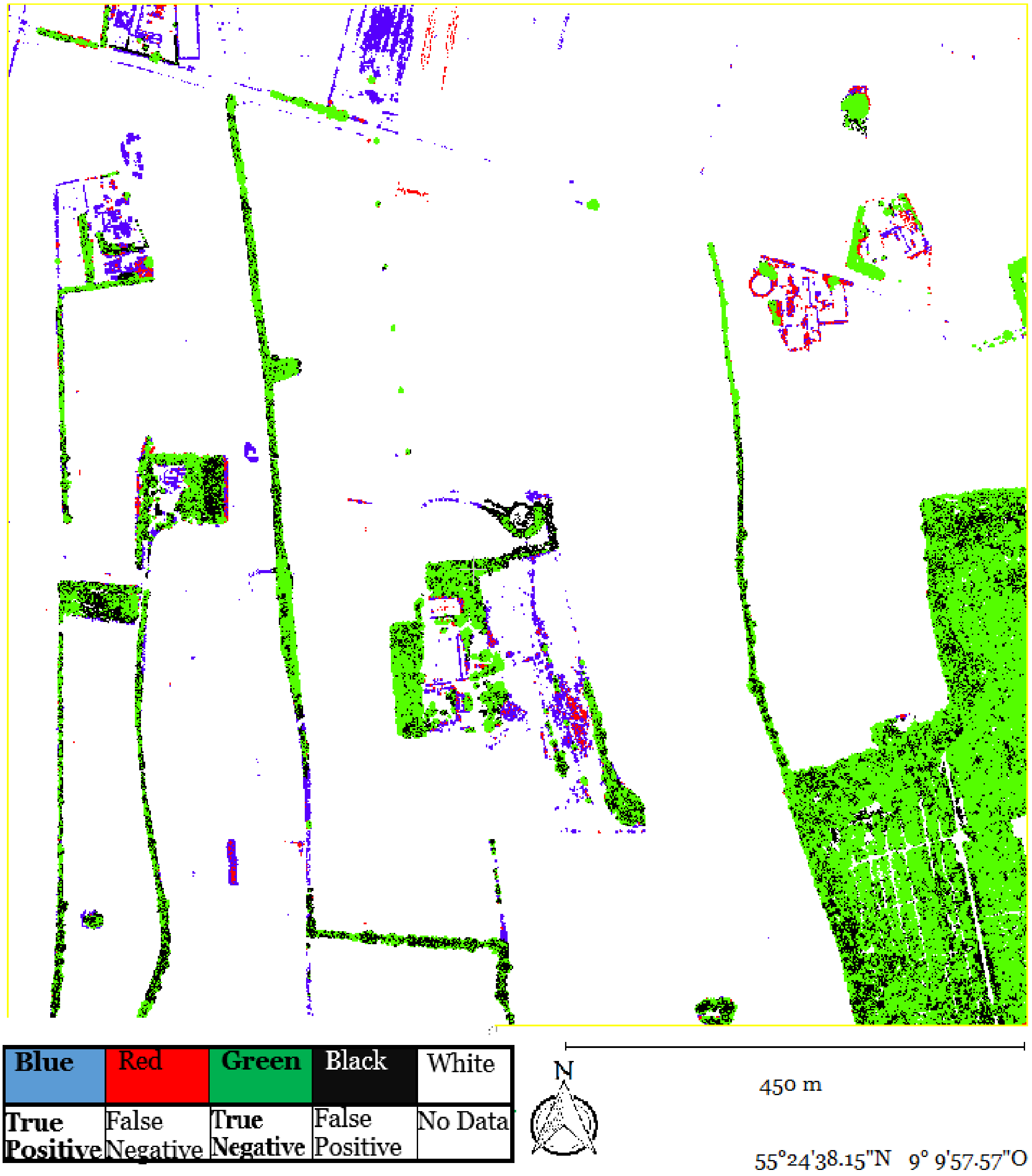


*Figure 19: Classification accuracy for Vamdrup –tensor filter ON - K50 – 20 points/m^2**

**=As multiple points are removed by thinning, it isn't the original point density anymore. However, many groups of objects will retain an average original point density while others will have 0 points remaining.*

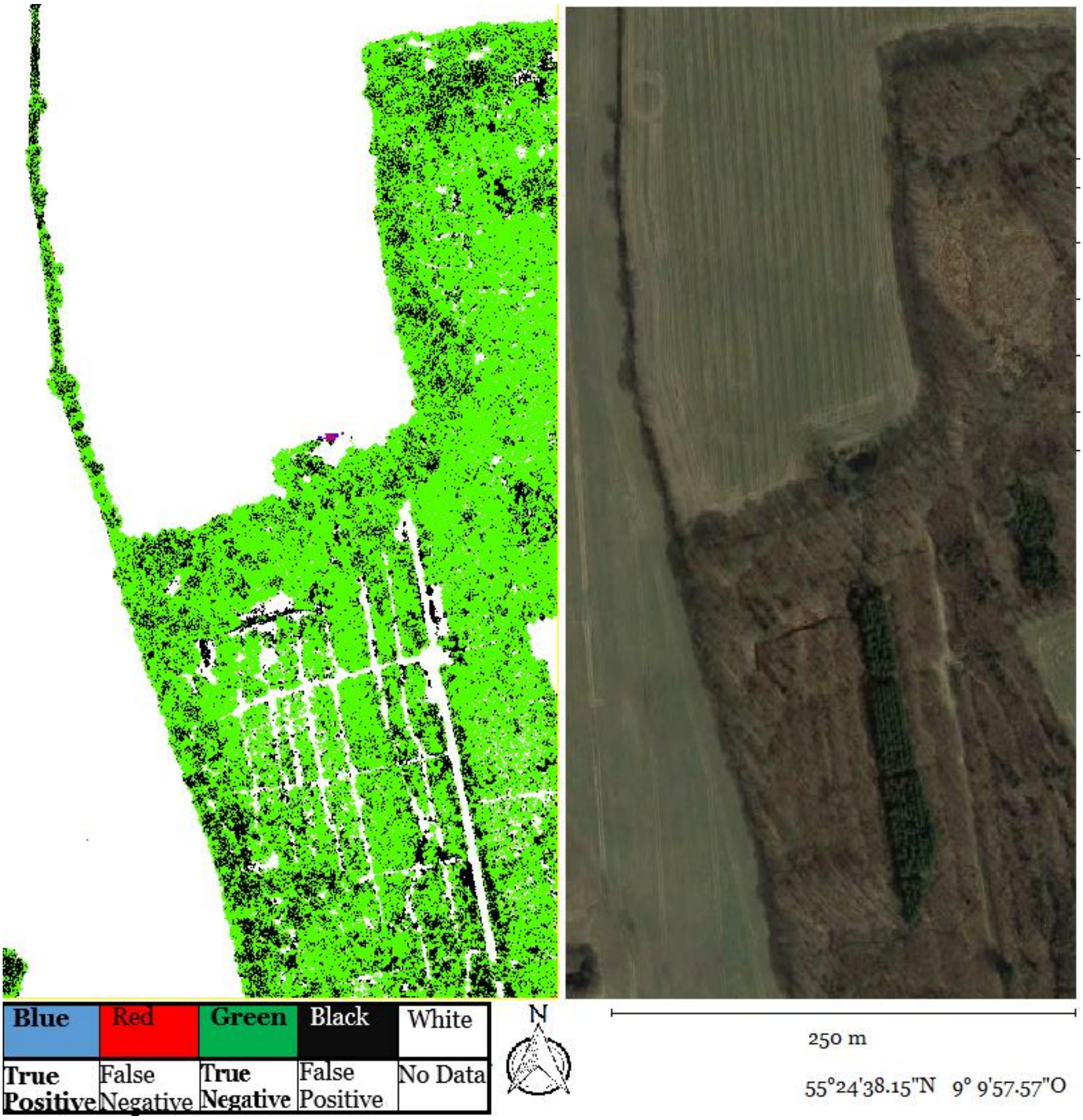


Figure 20: Location of Figure 18, now with a tensor filter applied.

Going back to the location of Figure 18, we can see that there are few black spots left and that these are not caused anymore by ground point discrepancy but seem to occur at the top and edges of some vegetation.

4.2.3 Fixed K between varying point densities

As seen in the previous sections, similar point density tiles seem to work reasonably well. However, when the methodology is applied on a different LiDAR tile, with a lower point density, the classification accuracy severely reduces. To show the effect, a classification has been performed with a K of 50 on a LiDAR tile from Spain, see table 2. The LiDAR tile captures a small area of about 2 km^2 in the region of O Santo, which is located in the North West of Spain, see figure 5.

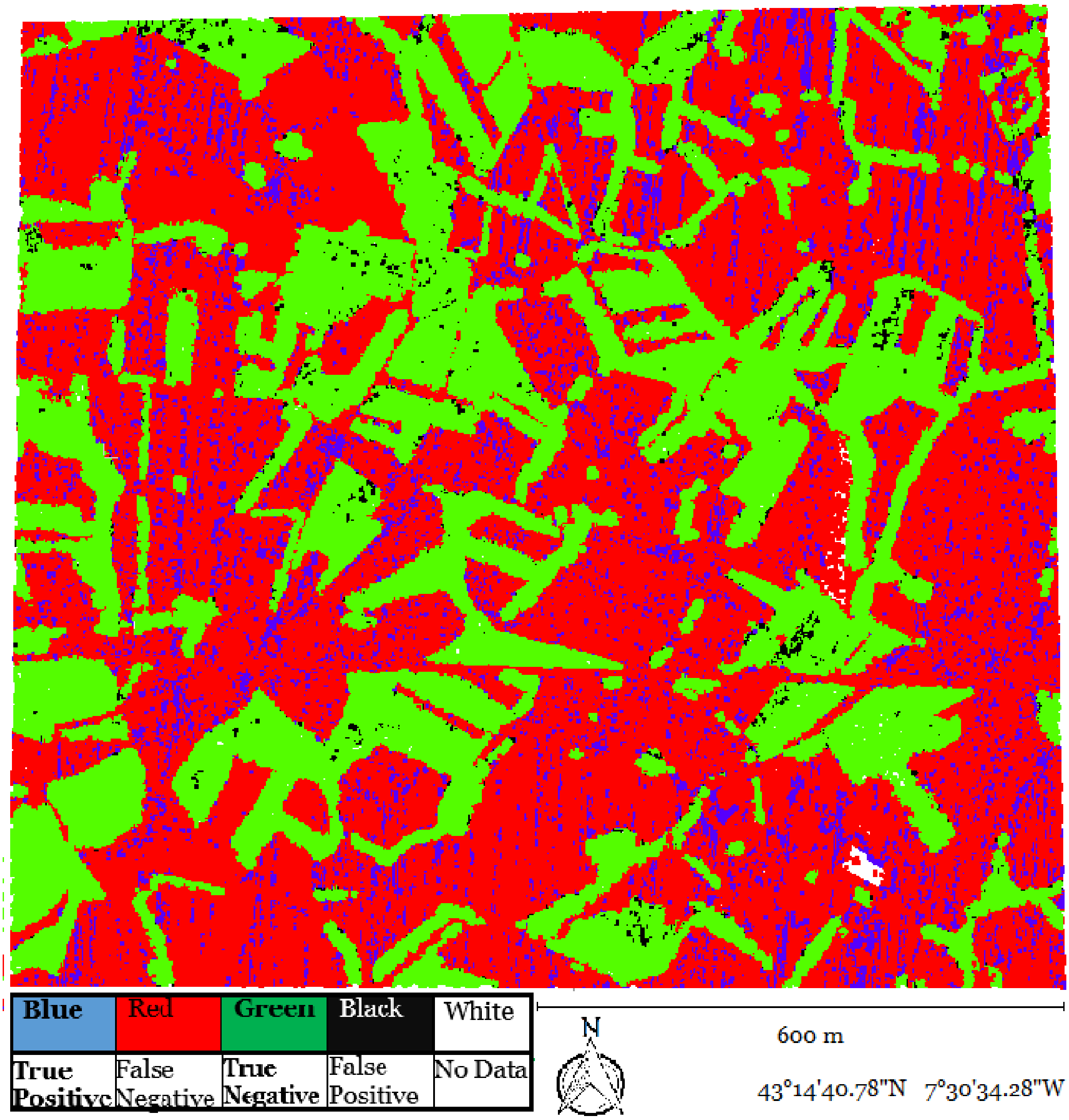


Figure 21: Classification accuracy for Spain, O Santo – K = 50, tensor filter OFF 0.5 points/m².

The accuracy for this classification is 35.01%. Initially, it looks like the classifier managed to do a relatively good job, as there is a lot of agreement over the vegetation classification between the reference and RF classified class. However, red indicates a discrepancy between non-vegetation (reference) and vegetation (RF classified). The large amount of falsely classified points is largely due to all the points being classified as vegetation. As the number of RF vegetation classified points is 673.234 and the non-vegetation having 99.012 points. The accuracy of the non-vegetation class is only 0.66% whereas the vegetation class receives an accuracy score of 93.16%. When the point density of a tile is low, and a high K is being used, every point will be classified as a vegetation class (0). This is due to the increase in search distance for the neighbors as a result of having less points per square meter, the number of square meters to cover increases to meet the criteria of 50 closest neighbors. The result is that points of multiple objects will be grouped together, and as the vast majority of non-vegetation points have a high planarity and low sphericity, the number of points fitting those criteria will lower. When applying a tensor filter based on Planarity and Sphericity, it will remove a lot of the wrongly classified (reference class 1) points and reduce the error. This is to be expected as the tensor filter is a very effective method of dealing with points which are most likely not going to be part of a vegetation object. However, the tensor filter will still not solve the underlying effect of falsely classifying every point as vegetation.

4.2.4 Adaptive K

Within the workflow a K-Dimensional Tree is built to efficiently retrieve the K closest neighbors for a certain query. There are two methods of retrieving the closest neighbors: radius and fixed K. The fixed K determines the number of neighbors to be found whereas with a radius a maximum distance can be chosen which will give K neighbors for a maximum distance. However, this causes variables to have an uneven length as two points might have different number of neighbors fulfilling the same distance criteria. The uneven number of neighbors prevents vectorized calculations as voids are filled with Not a Number (NaN) which is unusable. As distance is directly related to an object, the distance (and therefore number of neighbors) needs to be roughly the same if we are to describe equal objects at different point densities. For instance, doubling the distance causes points to be assessed for neighbors being part of a different object, such as a house or other man-made object. In the event of a cluster of vegetation, i.e. a forest, this effect will be less pronounced as the point will still be classified as a vegetation point as the neighbors share the same object class (vegetation). Therefore, a new function has been written which 1) estimates the point density of the LiDAR dataset and 2) determines a K which is in correspondence with the distance searched during the training of the classifier. This causes the number of neighbors to be equal for each point and the distance searched nears that of the original search radius. This effect is illustrated in figure 22 where three point clouds are displayed and each (point) color corresponds to a separate point cloud. The yellow bounding box gives the maximum extent of the area covered by these points.

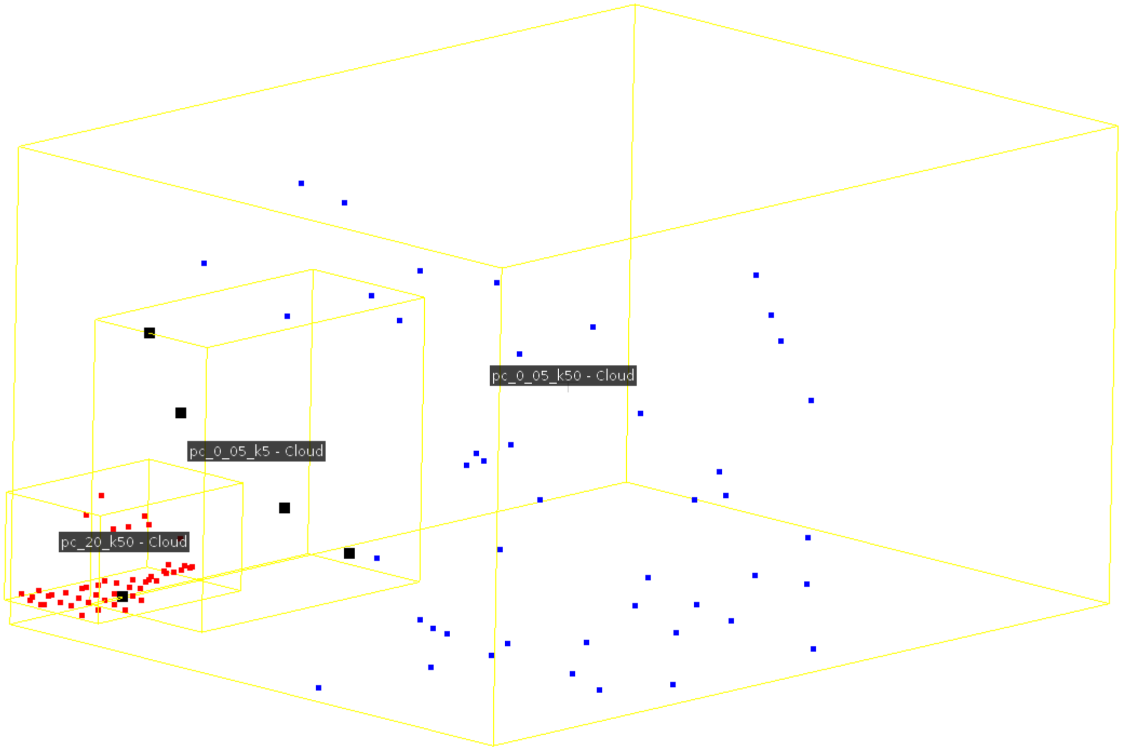
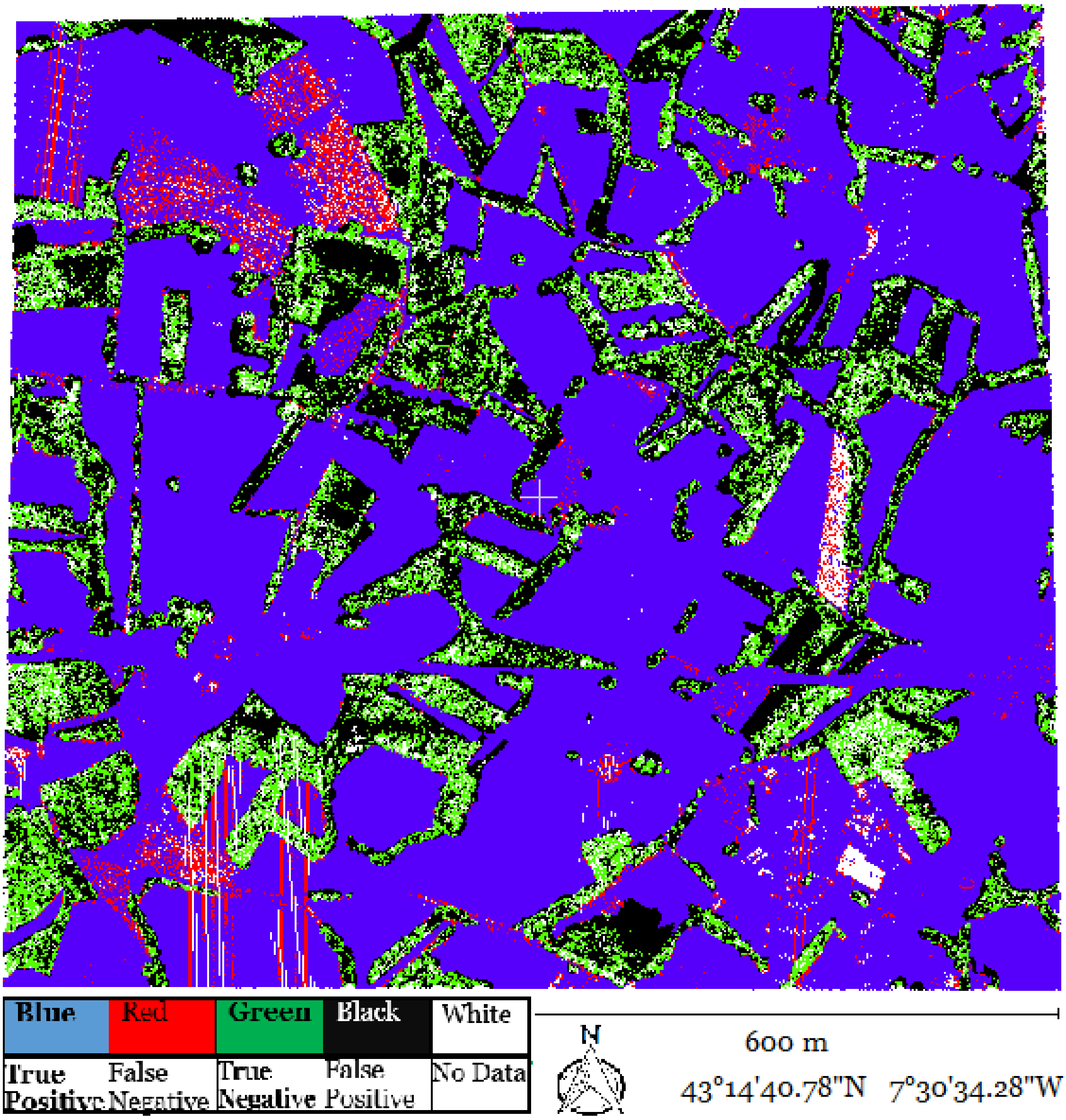


Figure 22: Illustration of the effect of an adaptive K. Three point clouds are displayed with each having a different point color. Blue is 0.5 points/m² - K50, Black is 0.5 points/m² - K5, Red is 20 points/m² - K50.

As the search distance increases, the eigenvalues vectors and properties change for that specific evaluation point as well. However, linearly scaling down the value for K = 50 for 16 points per square meter, to 0.5 points per square meter causes the adaptive K to reach 1. A minimum number of neighbors of 3 is required to have sortable eigenvectors and values, of which almost all features are dependent on. Therefore, a minimum K = 3 is introduced to produce any significant results. However, by trial and error, a minimum value of 5 for K seems to be an appropriate number of neighbors to maintain an equal search distance as compared to K = 50 for 16 points / m². Consequently, the minimum K used has been set to 5.



Blue	Red	Green	Black	White
True Positive	False Negative	True Negative	False Positive	No Data

Figure 23: Spain, O Santo – K = 5 tensor filter OFF -0.5 points/m²

When using an adaptive K, the accuracy reaches 78.73%. Still, there is quite some misclassification which is also why an accuracy of 78% is reached and not much higher. However, generally spoken the adaptive K seems to be an effective first measure, as shown in the assessment for Spain's dataset. It is a low point density dataset which, when supplied with a large K (50), will tend to perform the worst (see Appendix B: Spain and Belgium (DHMVI)). This shows that using an adaptive K based on the point density is an effective measure of dealing with varying (between datasets) point densities.

Denmark, The Netherlands and Belgium's DHMVII datasets perform extremely well. They have comparable point densities and proves the feasibility of applying this methodology on a continental scale.

5 Discussion

5.1 Computational performance

The performance gain in the parameter calculation process became 1190 times as fast as the sequential approach. However, the total workflow speed gain is 175 times. This is due to introducing extra pre and post-processing steps to facilitate the new calculation method.

Even though the initial performance gain is significant, fine tuning the allocation of memory might yield better results as a relative conservative approach has been applied to allow point clouds of up to 500 million points to be safely calculated without running out of memory. The chunk size has automatically been set to 500.000 which means that the multiprocessing with chunks methodology will be able to facilitate 20 million points per loop with 40 cores. As there are only 10 million points being evaluated in this research, the performance (in processed points / second) is only half of its capacity and might therefore increase in performance until 20 million points.

Compared to the calculation of features, the classification now takes too long and is now considered a bottleneck. When classification is added as a task, the feature calculation is nearly 35 times slower. This is in close correspondence with the amount of logical processing cores the server has. This means that during the parameter calculation, the classifier is probably being locked by a single thread whereas 39 other threads are waiting in a queue to use the same classifier. This issue might be solved by rebuilding the classifier with the number of threads to use during classification. As building a RF classifier with multiple threads allows the classifier to be used by multiple threads at the same time (Pedregosa, et al., 2011).

The tensor filter was intended as an effective measure of reducing calculation times (Lucas, 2017). However, the way it is implemented in this research requires more time to determine which points should be removed and what the original index (per thread) was than it saves in calculation time. Instead being an effective measure to reduce calculation times, it is still an effective measure to reduce file size. As file size gets reduced by leaving un-wanted points out. This can be helpful when the machine on which to visualize is not capable of loading the entire point cloud into the RAM. Furthermore, even though the parameter calculation process takes more time, the total processing time is lower with the tensor filter on. This is largely due to less time spent exporting the results, as there is less data left when a tensor filter is applied. We can expect an improved speed gain on increasingly larger point clouds.

Another possible way of overcoming the increased data export times is by using the same workflow with a different In/Out (I/O) library. Point Data Abstraction Library (PDAL) released Python compatibility in which it lets the user use the PDAL engine to perform LAS manipulations and communicate that with a Python environment (Bell, et al., 2018). As PDAL is written in C++, it is most likely more efficient with data. Both C++ and Python are high-level programming languages (Chawla, 2016). However, the difference between the two is that Python uses an interpreter and C++ uses a compiler. The interpreter translates the written code line per line to instructions useable for the hardware whereas a compiler translates the complete code (regardless of line occurrence) to machine code (Programiz, 2011). PDAL would then manage the import and export of LAS files where Python would handle the value manipulation. This would probably ensure the best performance of both environments and might prove a significant speed-up with storing results. Furthermore, the server uses a hard-disk drive (HDD) which has generally lower writing speeds (50 – 120 MB/s) compared to the solid-state drive (SSD) (220 – 500 MB/s) (Baxter, n.d.). Exchanging the HDD or adding an SSD to the server might increase the output operation.

5.2 Model application

The overall classification accuracy across different national datasets was in general, better than the annotated dataset. The largest issues seem to occur when points are grouped together as a single class, while in reality they should have been separated, see Appendix C. This frequently happens when annotating trees in which the ground points underneath a tree are grouped together with the tree. During pre-processing of the data, a filter could possibly be applied which removes all points labeled as 'ground' which would clearly not be part of vegetation (Chang, et al., 2008). The higher point density datasets performed better than the low point density datasets. Therefore, effort should be made to further optimize the classification accuracy of low point density datasets. This might be achieved by a combination of retraining a classifier on a low point density in combination with a variable K. Once the point density of a tile is determined, the corresponding classifier could be automatically loaded.

Since the user can make mistakes while manually segmenting, the metrics provided might be caused by either the classifier and or the user. Denmark had some misclassification around the edges of trees, which might indicate that the value for K was too low. Originally, on 16 points/m^2 a K of 50 has been found for optimizing for the specific density (Lucas, 2017). As Denmark has had a LiDAR tile which has 20 points / m^2 the used K was probably too low. This can also explain why the misclassification occurs around the edges of trees for example, as not enough neighbors are considered and therefore the radius of the search area is too low (compared to a K=50 for 16 points/m^2).

For the accuracy of classification, the standardly used confusion matrix metrics have been used, see Appendix B. However, arguably, the accuracy could be redefined depending on the aim. If the aim is to classify points as vegetation, then:

$$Accuracy = \frac{True\ Negative}{(True\ Negative + False\ Negative + False\ Positive)}$$

This would weigh the classification of vegetation more, as these points are required for linear vegetation element detection.

Each point is evaluated to be 'part of' vegetation or not. However, in the case of the thesis, points were only classified and not filtered if they were points belonging to an object of interest, such as: a hedge, tree or other larger vegetation objects. Generally, small shrubbery and fields are also considered as vegetation elements. Therefore, it depends on the aim and definition of what is of interest. It is up to the user to determine what the aim and interest is and to use those criteria to determine which points should be included to vegetation or not. If the user creates a training dataset which predominantly includes forests, other vegetation points might be less represented and therefore be classified as non-vegetation.

Lastly, as the classification has been performed on different national datasets with similar environmental settings, the classification performance in urban or mountainous areas are still untested and might prove to behave differently in terms of classification accuracy.

5.3 Future outlook

Due to the increase in computational speed, applying the methodology of classification starts to become more feasible to do at a continental scale. However, applying the methodology on a continental scale will not be an easy task – as the machines required to do so within an acceptable timeframe will not be cheap. A cost comparison for the classification of Europe has been made between the two methodologies: Sequential and multi core vectorized. First, if we assume that Europe will averagely have 10 points/m^2 and Europe has a surface area 1.01 x 10^^{13} m^2, the resulting number of points would be 1.01 x 10^^{14}. Furthermore, in 2014, it was estimated that Amazon has around 1,5 million servers in at their disposal (Clark, 2014),which would be an option for a task such as this. With the current pricing, choosing servers which have specifications closely related to the specifications this framework was developed on (Appendix D), a single server would cost $1.75 / hour (Amazon, 2018).

Table 5: Cost estimation for the classification of Europe.

	Sequential	Multi core vectorized
Points / s / server	600	111.000
Time required for 1 machine (years)	5040	28,8
Number of servers < 40 hours Calculation + classification	1.200.000 + n.d.	6300 + 220.500 = 226.800
Costs Calculation + classification	$2.756.250.000 + n.d.	$441.000 + $15.309.000 = $15.750.000

Computing time can be reduced by using multiple servers which, just like the current frame work splits the data, splits data across machines. With around 6300 servers calculating the features for Europe could be done in less than 2 days. The classification aspect, however, takes considerably more time. In fact, if we determine a ratio between the calculation speed and classification speed – we can determine how many more machines would have to full time classify to keep up to the pace of the feature calculations. The classification took 35 times as long, thus the ratio becomes 1:35 (1 feature calculation server: 35 classification servers). For the sequential approach however, the time required to classify remained untested. Therefore, the cost presented for the sequential approach does not include the number of servers required to classify Europe, the presented number of servers would only be calculating features. Regardless of the number of servers used, the total costs of classifying Europe will remain relatively the same as using Amazon's services depend on hourly processing costs. Less machines mean longer processing time at a low hourly rate, whereas more machines mean less processing time at a larger hourly rate. Pursuing a task like that on a personal computer or laptop will take too long to be worthwhile. Personal computers might be used for smaller research areas with up until 50 million points per tile.

The features chosen are calculated because they are needed for the classification of vegetation. However, potentially these same features can be used for different research purposes which involve object recognition. Furthermore, the framework can easily introduce new features and will quite likely be able to process new features at equal speeds.

Therefore, the added benefit of being able to process on a high resolution and being able to view the feature values per point, might aid in further research and development. The presented computational framework might be the stepping stone for further development of high-resolution processing of LiDAR tiles.

6 Conclusion

The presented computational framework proved to be able to process LiDAR data in an unprecedented speed, achieving a calculation speed increase of 1190 times and a total workflow speed increase of 175 times. Meanwhile, classification accuracy for extreme low point densities (< 5 points/m^2) has been increased from 35% to 78% by introducing an adaptive K, which is a significant step forward. However, as classification of low point density (< 5 points/m^2) LiDAR tiles is still not at a desired accuracy, effort should be made to further develop a methodology of handling low point density point clouds. Furthermore, the effect of different landscapes is still untested and might have a significant impact on classification accuracy. The ability of classifier transportability saves the laborious task of annotating training data per case, which is essential to processing LiDAR data on a continental scale. Furthermore, building an unbalanced K-Dimensional tree saves up to 5 x building time compared to the default, balanced K-Dimensional tree and does not negatively impact the precision of retrieving neighbors. The classification process has become a new bottleneck, which should be solved by building a multiprocessing capable classifier. Furthermore, the Sphericity and Planarity filter slowed down the calculation speed but marginally improved the total processing time. Even though we can’t effectively apply the framework on a continental scale yet, it can already have implications for further LiDAR based research due to its flexibility and rapid processing.

7 References

Amazon, 2018. *Amazon EC2 Dedicated Hosts Pricing*. [Online]
Available at: https://aws.amazon.com/ec2/dedicated-hosts/pricing/
[Accessed 17 August 2018].

Baxter, A., n.d. *SSD vs HDD*. [Online]
Available at: https://www.storagereview.com/ssd_vs_hdd
[Accessed 5 September 2018].

Bell, A. et al., 2018. *PDAL / Python*. [Online]
Available at: https://pdal.io/python.html
[Accessed 17 August 2018].

Bentley, J. L., 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM,* 9, Volume 18, pp. 509-517.

Birkland, A., 2012. *Vectorization,* Ithica: Cornell University Centre for Advanced Computing.

Breiman, L., 2001. Random Forests. *Machine Learning,* 45(1), pp. 5-32.

Brown, R. A., 2015. Building a Balanced k-d Tree in O(kn log n) Time. *Journal of Computer Graphics Techniques,* 4(1), pp. 50-68.

Bucher, I. Y., 1983. *The Computational Speed of Supercomputers..* Minneapolis, ACM, pp. 151-165.

Buler, J. J., Moore, F. R. & Woltmann, S., 2007. A MULTI-SCALE EXAMINATION OF STOPOVER HABITAT USE BY BIRDS. *Ecology,* Volume 88, pp. 1789-1802.

Cardinale, B. J. et al., 2012. Biodiversity loss and its impact on humanity. *Nature,* Volume 486, p. 59.

Ceballos, G. et al., 2015. Accelerated modern human--induced species losses: Entering the sixth mass extinction. *Science advances,* Volume 1 no. 5, e1400253.

Chang, Y.-C., Habib, A. F., Lee, D. C. & Yom, J.-H., 2008. *AUTOMATIC CLASSIFICATION OF LIDAR DATA INTO GROUND AND NONGROUND.* Beijing, s.n.

Chawla, M., 2016. *Levels of Programming Languages*. [Online]
Available at: https://thebittheories.com/levels-of-programming-languages-b6a38a68c0f2
[Accessed 5 September 2018].

Clark, J., 2014. *5 Numbers That Illustrate the Mind-Bending Size of Amazon's Cloud: https://www.bloomberg.com.* [Online]
Available at: https://www.bloomberg.com/news/2014-11-14/5-numbers-that-illustrate-the-mind-bending-size-of-amazon-s-cloud.html
[Accessed 17 August 2018].

Cody, M. L., 1985. *Habitat selection in birds.* s.l.:Academic Press.

Coffin, J., 2016. *Why is vectorization, faster in general, than loops?.* [Online]
Available at: https://stackoverflow.com/questions/35091979/why-is-vectorization-faster-in-general-than-loops
[Accessed 1 August 2018].

Corine, 2016. *Corine Land Cover - Copernicus land monitoring service 2016.* s.l.:s.n.

Davies, A. B. & Asner, G. P., 2014. Advances in animal ecology from 3D-LiDAR ecosystem mapping. *Trends in ecology & evolution,* Volume 29, pp. 681-691.

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

Ferrier, S., Watson, G., Pearce, J. & Drielsma, M., 2002. Extended statistical approaches to modelling spatial pattern in biodiversity in northeast New South Wales. I. Species-level modelling. *Biodiversity & Conservation,* Volume 11, pp. 2275-2307.

Fuller, R. J., 2012. *Birds and habitat: relationships in changing landscapes.* s.l.:Cambridge University Press.

Girardeau-Montaut, D., 2015. *Edit\Subsample.* [Online]
Available at: https://www.cloudcompare.org/doc/wiki/index.php?title=Edit%5CSubsample
[Accessed 19 August 2018].

Girardeau-Montaut, D., 2018. *Interactive Segmentation Tool.* [Online]
Available at:
https://www.cloudcompare.org/doc/wiki/index.php?title=Interactive_Segmentation_Tool
[Accessed 1 August 2018].

Hoekstra, J. M., Boucher, T. M., Ricketts, T. H. & Roberts, C., 2005. Confronting a biome crisis: global disparities of habitat loss and protection. *Ecology letters,* pp. 23-29.

Isenburg, M., 2018. *LASmoons: Maria Kampouri.* [Online]
Available at: https://rapidlasso.com/category/lastools/
[Accessed 17 August 2018].

Isenburg, M., 2018. *rapidlasso GmbH.* [Online]
Available at: https://rapidlasso.com/lastools/
[Accessed 5 January 2018].

Jones, E., Oliphant, T., Peterson, P. & others, 2001. *SciPy: Open source scientific tools for Python.* [Online]
Available at: http://www.scipy.org/
[Accessed 1 October 2017].

Kissling, W. D., Field, R. & Böhning-Gaese, K., 2008. Spatial patterns of woody plant and bird diversity: functional relationships or environmental effects?. *Global Ecology and Biogeography,* Volume 17, pp. 327-339.

Kissling, W. D. et al., 2015. Towards global interoperability for supporting biodiversity research on essential biodiversity variables (EBVs). *Biodiversity,* Volume 16, pp. 99-107.

Kissling, W. D., Seijmonsbergen, A. C., Foppen, R. P. B. & Bouten, W., 2017. eEcoLiDAR, eScience infrastructure for ecological applications of LiDAR point clouds: reconstructing the 3D ecosystem structure for animals at regional to continental scales. *Research Ideas and Outcomes,* Volume 3, p. e14939.

Lausch, A. et al., 2016. Linking Earth Observation and taxonomic, structural and functional biodiversity: Local to ecosystem perspectives. *Ecological Indicators,* Volume 70, pp. 317-339.

Loog, M., 2018. Supervised Classification: Quite a Brief Overview.. *Machine Learning Techniques for Space Weather.,* pp. 113-145.

Lucas, C., 2017. *Delineating linear vegetation elements in agricultural landscapes using LiDAR point clouds,* Amsterdam: DARE.

Newbold, T. et al., 2015. Global effects of land use on local terrestrial biodiversity. *Nature,* Volume 520, p. 45.

Nishihara, R., 2018. *Parallel Processing and Multiprocessing in Python.* [Online]
Available at: https://wiki.python.org/moin/ParallelProcessing
[Accessed 4 September 2018].

Oliphant, T. E., 2006. *A guide to NumPy,* s.l.: Trelgol Publishing.

Patrick McCormick, M., 2005. Airborne and Spaceborne Lidar. In: C. Weitkamp, ed. *Lidar: Range-Resolved Optical Remote Sensing of the Atmosphere.* New(York): Springer New York, pp. 355-397.

Paula, M. D., Groeneveld, J. & Huth, A., 2016. The extent of edge effects in fragmented landscapes: Insights from satellite measurements of tree cover. *Ecological Indicators,* Volume 69, pp. 196-204.

Pedregosa, F. et al., 2011. Scikit-learn: Machine Learning in Python. *Machine Learning Research,* Volume 12, pp. 2825-2830.

Pereira, H. M. et al., 2013. Essential biodiversity variables. *Science,* Volume 339, pp. 277-278.

Pfeifer, N., Mandlburger, G., Otepka, J. & Karel, W., 2014. OPALS – A framework for Airborne Laser Scanning data analysis. *Computers, Environment and Urban Systems,* Volume 45, pp. 125-136.

Popescu, S. C., Zhao, K., Neuenschwander, A. & Lin, C., 2011. Satellite lidar vs. small footprint airborne lidar: Comparing the accuracy of aboveground biomass estimates and forest structure metrics at footprint level. *Remote Sensing of Environment,* Volume 115, pp. 2786-2797.

Programiz, 2011. *Interpreter Vs Compiler : Difference Between Interpreter and Compiler.* [Online]
Available at: https://www.programiz.com/article/difference-compiler-interpreter
[Accessed 5 September 2018].

Python Software Foundation, 2016. *Python 3.6.0.* [Online]
Available at: https://www.python.org/downloads/release/python-360/
[Accessed 5 January 2018].

Rodríguez, J. P., Brotons, L., Bustamante, J. & Seoane, J., 2007. The application of predictive modelling of species distribution to biodiversity conservation. *Diversity and Distributions,* Volume 13, pp. 243-251.

Skidmore, A. K. & Pettorelli, N., 2015. Agree on biodiversity metrics to track from space: ecologists and space agencies must forge a global monitoring strategy. *Nature,* Volume 523, pp. 403-406.

Stockwell, D. R. B. & Peterson, A. T., 2002. Effects of sample size on accuracy of species distribution models. *Ecological Modelling,* Volume 148, pp. 1-13.

Stokes, J., 2002. *Introduction to Multithreading, Superthreading and Hyperthreading*. [Online]
Available at: https://arstechnica.com/features/2002/10/hyperthreading/
[Accessed 4 September 2018].

Turner, W. et al., 2003. Remote sensing for biodiversity science and conservation. *Trends in ecology & evolution,* Volume 18, pp. 306-314.

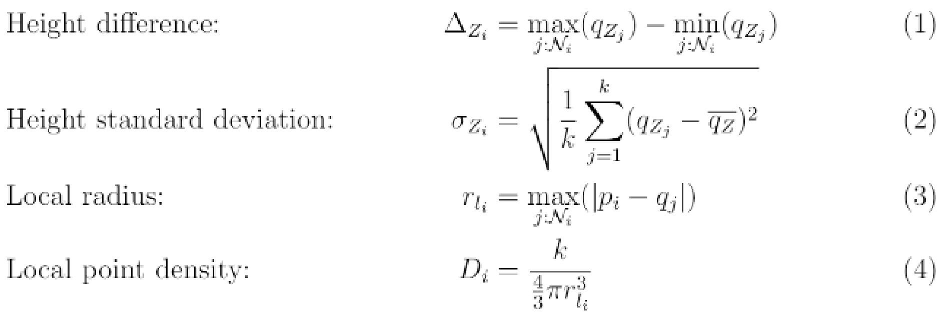
Wescott, B., 2013. *Every Computer Performance Book*. s.l.:CreateSpace Independent Publishing Platform , USA ©2013.

Wiens, J. A., 1989. Spatial scaling in ecology. *Functional ecology,* Volume 3, pp. 385-397.

Wouters, T., 2017. *Global Interpreter Lock*. [Online]
Available at: https://wiki.python.org/moin/GlobalInterpreterLock
[Accessed 4 September 2018].

Appendix A

Point cloud features



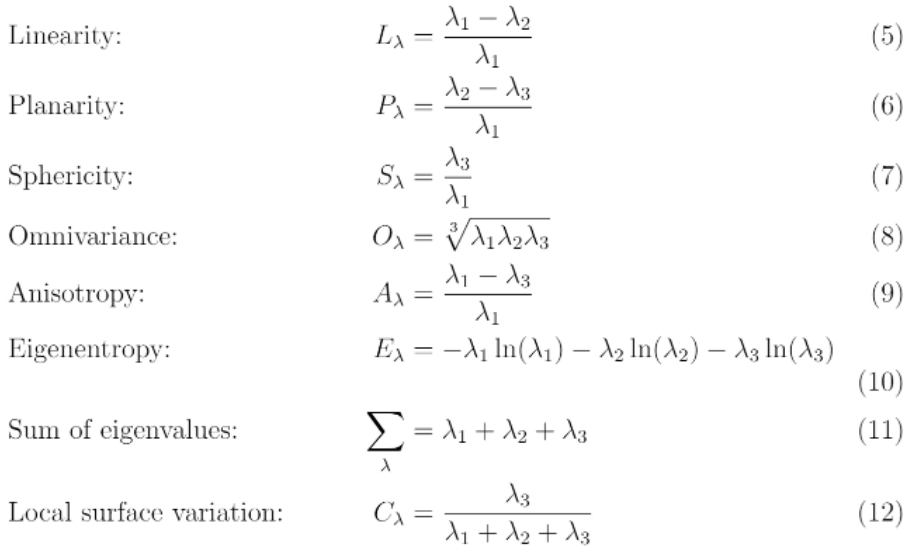
Height difference: $$\Delta_{Z_i} = \max_{j:\mathcal{N}_i}(q_{Z_j}) - \min_{j:\mathcal{N}_i}(q_{Z_j}) \tag{1}$$

Height standard deviation: $$\sigma_{Z_i} = \sqrt{\frac{1}{k}\sum_{j=1}^{k}(q_{Z_j} - \overline{q_Z})^2} \tag{2}$$

Local radius: $$r_{l_i} = \max_{j:\mathcal{N}_i}(|p_i - q_j|) \tag{3}$$

Local point density: $$D_i = \frac{k}{\frac{4}{3}\pi r_{l_i}^3} \tag{4}$$

Figure 24: Basic geometry calculations to determine point features. (Lucas, 2017)



Linearity: $$L_\lambda = \frac{\lambda_1 - \lambda_2}{\lambda_1} \tag{5}$$

Planarity: $$P_\lambda = \frac{\lambda_2 - \lambda_3}{\lambda_1} \tag{6}$$

Sphericity: $$S_\lambda = \frac{\lambda_3}{\lambda_1} \tag{7}$$

Omnivariance: $$O_\lambda = \sqrt[3]{\lambda_1\lambda_2\lambda_3} \tag{8}$$

Anisotropy: $$A_\lambda = \frac{\lambda_1 - \lambda_3}{\lambda_1} \tag{9}$$

Eigenentropy: $$E_\lambda = -\lambda_1 \ln(\lambda_1) - \lambda_2 \ln(\lambda_2) - \lambda_3 \ln(\lambda_3) \tag{10}$$

Sum of eigenvalues: $$\sum_\lambda = \lambda_1 + \lambda_2 + \lambda_3 \tag{11}$$

Local surface variation: $$C_\lambda = \frac{\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3} \tag{12}$$

Figure 25: Structure tensors which are used as point features. (Lucas, 2017)

Appendix B

Confusion matrix results

Table 6: Beesd – tensor filter - OFF K50 – 16 points/m² – balanced KD tree

Metric	Formula	Result
True Negative (veg)	Count	1651519
True Positive (nonveg)	“”	4268642
False Positive	“”	20614
False Negative	“”	916508
Accuracy	(TP + TN) / Total	86.33%
Error Rate	1 – Accuracy	13.67%
True Positive Rate	TP / Reference 1	99.52%
False Positive Rate	FP / Reference 0	35.69%
Specificity	TN / reference 0	64.31%
Precision	TP / reference 1	99.52%
Prevalence class ‘1’	Reference 1 / total	62.55%
Prevalence class ‘0’	Reference 0 / total	37.45%
Total reference class ‘0’	Count	2568027
Total reference class ‘1’	“”	4289256
Total classified class ‘0’	“”	1672133
Total classified class ‘1’	“”	5185150
Total number of points	“”	6857283

Table 7: The Netherlands, Beesd – tensor filter ON – K50 – 16 points/m² – balanced KD tree.

Metric	Formula	Result
True Negative (veg)	Count	1420852
True Positive (nonveg)	“”	38175
False Positive	“”	5835
False Negative	“”	331044
Accuracy	(TP + TN) / Total	81.24%
Error Rate	1 – Accuracy	18.76%
True Positive Rate	TP / Reference 1	86.74%
False Positive Rate	FP / Reference 0	18.90%
Specificity	TN / reference 0	81.10%
Precision	TP / reference 1	86.74%
Prevalence class ‘1’	Reference 1 / total	02.45%
Prevalence class ‘0’	Reference 0 / total	97.55%
Total reference class ‘0’	Count	1751896
Total reference class ‘1’	“”	44010
Total classified class ‘0’	“”	1426687
Total classified class ‘1’	“”	369219
Total number of points	“”	1795906

Table 8: The Netherlands, Beesd – tensor filter OFF – K50 – 16 points/m² – unbalanced KD tree

Metric	Formula	Result
True Negative (veg)	Count	1651519
True Positive (nonveg)	“”	4268642
False Positive	“”	20614
False Negative	“”	916508
Accuracy	(TP + TN) / Total	86.33%
Error Rate	1 – Accuracy	13.67%
True Positive Rate	TP / Reference 1	99.52%
False Positive Rate	FP / Reference 0	35.69%
Specificity	TN / reference 0	64.31%
Precision	TP / reference 1	99.52%
Prevalence class ‘1’	Reference 1 / total	62.55%
Prevalence class ‘0’	Reference 0 / total	37.45%
Total reference class ‘0’	Count	2568027
Total reference class ‘1’	“”	4289256
Total classified class ‘0’	“”	1672133
Total classified class ‘1’	“”	5185150

Table 9: The Netherlands, Beesd – tensor filter ON – K50 – 16 points/m² – unbalanced KD tree.

Metric	Formula	Result
True Negative (veg)	Count	1420852
True Positive (nonveg)	“”	38175
False Positive	“”	5835
False Negative	“”	331044
Accuracy	(TP + TN) / Total	81.24%
Error Rate	1 – Accuracy	18.76%
True Positive Rate	TP / Reference 1	86.74%
False Positive Rate	FP / Reference 0	18.90%
Specificity	TN / reference 0	81.10%
Precision	TP / reference 1	86.74%
Prevalence class ‘1’	Reference 1 / total	02.45%
Prevalence class ‘0’	Reference 0 / total	97.55%
Total reference class ‘0’	Count	1751896
Total reference class ‘1’	“”	44010
Total classified class ‘0’	“”	1426687
Total classified class ‘1’	“”	369219

Table 10: Denmark, Vamdrup accuracy assesment without Sphericity & Planarity filter (no SPfilter) – K50

Metric	Formula	Result
True Negative (veg)	Count	1.621.075
True Positive (nonveg)	“”	10.394.027
False Positive	“”	52.632
False Negative	“”	1.200.686
Accuracy	(TP + TN) / Total	90.55%
Error Rate	1 – Accuracy	9.45%
True Positive Rate	TP / Reference 1	99.50%
False Positive Rate	FP / Reference 0	42.55%*
Specificity	TN / reference 0	57.45%*
Precision	TP / reference 1	99.50%
Prevalence class ‘1’	Reference 1 / total	78.73%
Prevalence class ‘0’	Reference 0 / total	21.27%
Total reference class ‘0’	Count	2.821.761
Total reference class ‘1’	“”	10.446.659
Total classified class ‘0’	“”	1.673.707
Total classified class ‘1’	“”	11.594.713

Table 11: Denmark, Vamdrup accuracy assesment with tensor filter (Sphericity & Planarity) – K50

Metric	Formula	Result
True Negative (veg)	Count	1389431
True Positive (nonveg)	“”	67788
False Positive	“”	15773
False Negative	“”	422143
Accuracy	(TP + TN) / Total	76.89%
Error Rate	1 – Accuracy	23.11%
True Positive Rate	TP / Reference 1	81.12%
False Positive Rate	FP / Reference 0	23.30%*
Specificity	TN / reference 0	76.70%*
Precision	TP / reference 1	81.12%
Prevalence class ‘1’	Reference 1 / total	04.41%
Prevalence class ‘0’	Reference 0 / total	95.60%
Total reference class ‘0’	Count	1811574
Total reference class ‘1’	“”	83561
Total classified class ‘0’	“”	1405204
Total classified class ‘1’	“”	489931

*Are heavily influenced by user segmentation. Segmentation is done by drawing an infinite cylinder around a group of points, extracting whatever is within the cylinder. This causes ground points to be frequently grouped to vegetation, whereas they should be separated from the vegetation class.

Table 12: Belgium, Boekhoute - Assenede DHMVII – tensor filter OFF – K50

Metric	Formula	Result
True Negative (veg)	Count	700920
True Positive (nonveg)	“”	1021027
False Positive	“”	19185
False Negative	“”	89587
Accuracy	(TP + TN) / Total	94.06%
Error Rate	1 – Accuracy	05.94%
True Positive Rate	TP / Reference 1	98.16%
False Positive Rate	FP / Reference 0	11.33%*
Specificity	TN / reference 0	88.67%*
Precision	TP / reference 1	98.16%
Prevalence class ‘1’	Reference 1 / total	56.82%
Prevalence class ‘0’	Reference 0 / total	43.18%
Total reference class ‘0’	Count	790507
Total reference class ‘1’	“”	1040212
Total classified class ‘0’	“”	720105
Total classified class ‘1’	“”	1110614

Table 13: Belgium, Boekhoute - Assenede DHMVII – tensor filter ON – K50

Metric	Formula	Result
True Negative (veg)	Count	573427
True Positive (nonveg)	“”	4908
False Positive	“”	8934
False Negative	“”	13403
Accuracy	(TP + TN) / Total	96.28%
Error Rate	1 – Accuracy	03.72%
True Positive Rate	TP / Reference 1	35.46%
False Positive Rate	FP / Reference 0	02.28%
Specificity	TN / reference 0	97.72%
Precision	TP / reference 1	35.46%
Prevalence class ‘1’	Reference 1 / total	02.30%
Prevalence class ‘0’	Reference 0 / total	97.70%
Total reference class ‘0’	Count	586830
Total reference class ‘1’	“”	13842
Total classified class ‘0’	“”	582361
Total classified class ‘1’	“”	18311

Table 14: Belgium, Boekhoute – Assenede DHMVI – tensor filter OFF – k5

Metric	Formula	Result
True Negative (veg)	Count	1999
True Positive (nonveg)	“”	84730
False Positive	“”	666
False Negative	“”	2107
Accuracy	(TP + TN) / Total	96.90%
Error Rate	1 – Accuracy	03.10%
True Positive Rate	TP / Reference 1	99.22%
False Positive Rate	FP / Reference 0	51.32%
Specificity	TN / reference 0	48.68%
Precision	TP / reference 1	99.22%
Prevalence class ‘1’	Reference 1 / total	95.41%
Prevalence class ‘0’	Reference 0 / total	04.59%
Total reference class ‘0’	Count	4106
Total reference class ‘1’	“”	85396
Total classified class ‘0’	“”	2665
Total classified class ‘1’	“”	86837

Table 15: Belgium, Boekhoute – Assenede DHMVI – tensor filter ON – k5

Metric	Formula	Result
True Negative (veg)	Count	1168
True Positive (nonveg)	“”	310
False Positive	“”	349
False Negative	“”	189
Accuracy	(TP + TN) / Total	73.31%
Error Rate	1 – Accuracy	26.69%
True Positive Rate	TP / Reference 1	47.04%
False Positive Rate	FP / Reference 0	13.93%
Specificity	TN / reference 0	86.07%
Precision	TP / reference 1	47.04%
Prevalence class ‘1’	Reference 1 / total	32.69%
Prevalence class ‘0’	Reference 0 / total	67.31%
Total reference class ‘0’	Count	1357
Total reference class ‘1’	“”	659
Total classified class ‘0’	“”	1517
Total classified class ‘1’	“”	499

Table 16: Belgium, Boekhoute – Assenede DHMVI – tensor filter OFF – k50

Metric	Formula	Result
True Negative (veg)	Count	4049
True Positive (nonveg)	“”	8543
False Positive	“”	76853
False Negative	“”	57
Accuracy	(TP + TN) / Total	14.07%
Error Rate	1 – Accuracy	85.93%
True Positive Rate	TP / Reference 1	10.00%
False Positive Rate	FP / Reference 0	01.39%
Specificity	TN / reference 0	98.61%
Precision	TP / reference 1	10.00%
Prevalence class ‘1’	Reference 1 / total	95.41%
Prevalence class ‘0’	Reference 0 / total	04.59%
Total reference class ‘0’	Count	4106
Total reference class ‘1’	“”	85396
Total classified class ‘0’	“”	80902
Total classified class ‘1’	“”	8600

Table 17: Belgium, Boekhoute – Assenede DHMVI – tensor filter ON – k50

Metric	Formula	Result
True Negative (veg)	Count	2995
True Positive (nonveg)	“”	0
False Positive	“”	1467
False Negative	“”	0
Accuracy	(TP + TN) / Total	67.12%
Error Rate	1 – Accuracy	32.88%
True Positive Rate	TP / Reference 1	0
False Positive Rate	FP / Reference 0	0
Specificity	TN / reference 0	1
Precision	TP / reference 1	0
Prevalence class ‘1’	Reference 1 / total	32.88%
Prevalence class ‘0’	Reference 0 / total	67.12%
Total reference class ‘0’	Count	2995
Total reference class ‘1’	“”	1467
Total classified class ‘0’	“”	4462
Total classified class ‘1’	“”	0

Table 18: Spain, O Santo – tensor filter OFF – k5

Metric	Formula	Result
True Negative (veg)	Count	47505
True Positive (nonveg)	“”	560509
False Positive	“”	13211
False Negative	“”	151021
Accuracy	(TP + TN) / Total	78.73%
Error Rate	1 – Accuracy	21.26%
True Positive Rate	TP / Reference 1	97.70%
False Positive Rate	FP / Reference 0	76.07%
Specificity	TN / reference 0	23.93%
Precision	TP / reference 1	97.70%
Prevalence class ‘1’	Reference 1 / total	74.30%
Prevalence class ‘0’	Reference 0 / total	25.71%
Total reference class ‘0’	Count	198526
Total reference class ‘1’	“”	573720
Total classified class ‘0’	“”	60716
Total classified class ‘1’	“”	711530

Table 19: Spain, O Santo – tensor filter ON – k5

Metric	Formula	Result
True Negative (veg)	Count	22149
True Positive (nonveg)	“”	7500
False Positive	“”	6465
False Negative	“”	19318
Accuracy	(TP + TN) / Total	53.49%
Error Rate	1 – Accuracy	46.51%
True Positive Rate	TP / Reference 1	53.71%
False Positive Rate	FP / Reference 0	46.59%
Specificity	TN / reference 0	53.41%
Precision	TP / reference 1	53.71%
Prevalence class ‘1’	Reference 1 / total	25.19%
Prevalence class ‘0’	Reference 0 / total	74.80%
Total reference class ‘0’	Count	41467
Total reference class ‘1’	“”	13965
Total classified class ‘0’	“”	28614
Total classified class ‘1’	“”	26818

Table 20: Spain, O Santo – tensor filter OFF – k50

Metric	Formula	Result
True Negative (veg)	Count	184956
True Positive (nonveg)	“”	85442
False Positive	“”	488278
False Negative	“”	13570
Accuracy	(TP + TN) / Total	35.01%
Error Rate	1 – Accuracy	64.99%
True Positive Rate	TP / Reference 1	14.89%
False Positive Rate	FP / Reference 0	06.84%
Specificity	TN / reference 0	93.16%
Precision	TP / reference 1	14.89%
Prevalence class ‘1’	Reference 1 / total	74.29%
Prevalence class ‘0’	Reference 0 / total	25.71%
Total reference class ‘0’	Count	198526
Total reference class ‘1’	“”	573720
Total classified class ‘0’	“”	673234
Total classified class ‘1’	“”	99012

Table 21: Spain, O Santo – tensor filter ON – k50

Metric	Formula	Result
True Negative (veg)	Count	104351
True Positive (nonveg)	“”	174
False Positive	“”	26290
False Negative	“”	113
Accuracy	(TP + TN) / Total	79.83%
Error Rate	1 – Accuracy	20.17%
True Positive Rate	TP / Reference 1	00.66%
False Positive Rate	FP / Reference 0	00.11%
Specificity	TN / reference 0	99.89%
Precision	TP / reference 1	00.66%
Prevalence class ‘1’	Reference 1 / total	20.21%
Prevalence class ‘0’	Reference 0 / total	79.79%
Total reference class ‘0’	Count	104464
Total reference class ‘1’	“”	26464
Total classified class ‘0’	“”	130641
Total classified class ‘1’	“”	287

Table 22: Balanced KD tree vs non-balanced KD tree accuracy assessment.

Metric	Formula	BALANCED	UNBALANCED
True Negative (veg)	Count	1651519	1651519
True Positive (nonveg)	“”	4268642	4268642
False Positive	“”	20614	20614
False Negative	“”	916508	916508
Accuracy	(TP + TN) / Total	86.33%	86.33%
Error Rate	1 – Accuracy	13.67%	13.67%
True Positive Rate	TP / Reference 1	99.52%	99.52%
False Positive Rate	FP / Reference 0	35.69%	35.69%
Specificity	TN / reference 0	64.31%	64.31%
Precision	TP / reference 1	99.52%	99.52%
Prevalence class ‘1’	Reference 1 / total	62.55%	62.55%
Prevalence class ‘0’	Reference 0 / total	37.45%	37.45%
Total reference class ‘0’	Count	2568027	2568027
Total reference class ‘1’	“”	4289256	4289256
Total classified class ‘0’	“”	1672133	1672133
Total classified class ‘1’	“”	5185150	5185150

Table 23: First 20 index numbers for two different K-D trees. Balanced versus Unbalanced K-D Trees have both received an identical query for the closest 50 neighbors. Both K-D trees retrieve the exact same neighbors.

Index number	Balanced	Unbalanced
0	0	0
1	188476	188476
2	1	1
3	188477	188477
4	11	11
5	12	12
6	9	9
7	13	13
8	188474	188474
9	34	34
10	35	35
11	32	32
12	8	8
13	188467	188467
14	36	36
15	188473	188473
16	31	31
17	71	71
18	68	68
19	72	72
20	7	7

Table 23: 10-fold crossvalidation results for the classifier used in this research.

Metric	Score
Accuracy	99%
Error rate	0.7%
True Positive Rate	90%
True Negative Rate	99%
False Positive Rate	0.6%
Specificity	99%
False Negative Rate	9.2%
Precision '1'	35%
Precision '0'	99%
Prevalence '1'	0.4%
Prevalence '0'	99%
F1 '1'	51.1%
F1 '0'	99%

Appendix C

Supplementary documentation

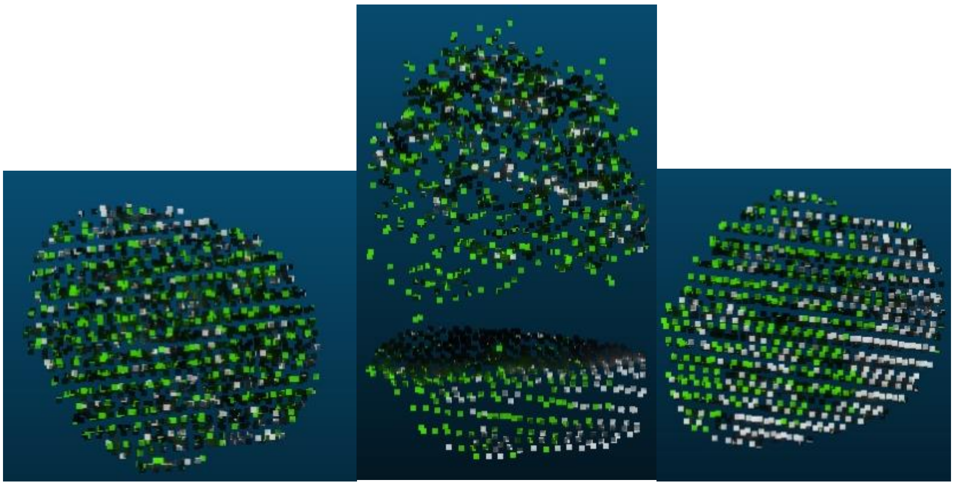


Figure 26: From left to right: Top, side and bottom view of a single tree. White are wrongly classified vegetation points. Green are successfully classified vegetation points.

Figure 26 shows a single tree segmented from a top-view. Green points indicate agreement for the vegetation class whereas white points indicate points which are classified as ground points but had a non-vegetation reference class. It is shown that the segmentation has been done on visual inspection of the top of the vegetation object. However, a vegetation object does not have to be one single filled cylinder and can incorporate multiple classes (vegetation and non-vegetation) within the cylinder as shown in the figure. Unfortunately, these points are treated as a single reference class.

One method to deal with these limitations is to apply a tensor filter to the dataset. By removing unwanted points, ground points also in general get removed. This methodology skews the output results and should be treated with care as we are creating artificially unbalanced classes.

Writing and reading .LAS files.

Conventionally, a comma separated values (CSV) file is used to store additional information other than the supplied data from a .LAS file. However, a user is also able to specify a custom field within the LAS file where the user can store additional information. A python library named LasPy was developed in order to handle this type of information and data manipulation. By defining extra dimensions within the the .LAS file, a user can store close to any type of information. The module handles the data like a large matrix in which you can store multiple new columns with data, corresponding to the points recorded in the .LAS file. The resulting .LAS file is read by most of the current LAS processing tools such as: CloudCompare and LAStools. Therefore, a function has been written which uses the functions from LasPy to create dynamic column names corresponding to the settings used to calculate the varying features.

Appendix D

Hardware, software and data source description.

Code

https://github.com/BerendWijers/LiDAR-feature-extraction

Hardware

CPU: 2 x Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz (2x10 physical cores / 2x20 logical cores)

2x 640KiB L1 cache

2x 2560KiB L2 cache

2x 25MiB L3 cache

Memory: 8 x 32 GB (256 GB) DDR4 @ 2400 MHz

Hard drive: Seagate Enterprise Capacity 2.5 HDD ST1000NX0423 - hard drive - 1 TB - SATA 6Gb/s

Software

Operating system: CentOS Linux 7 (Core)

Programming: Python 3.6

Python packages: Numpy, pandas, sklearn, imblearn, glob, laspy, os, multiprocessing, Queue, TQDM

Custom Python packages: Lidar_funcs (Wijers), classification packages (Lucas)

Data sources

The Netherlands: Publieke Dienstverlening op de Kaart (AHN3)

https://www.pdok.nl/nl/ahn3-downloads

Denmark: Styrelsen for Dataforsyning og Effektivisering

https://download.kortforsyningen.dk/content/dhmpunktsky

Belgium: Informatie Vlaanderen, BVK OpenData portaal (DHMVI & DHMVII)

https://overheid.vlaanderen.be/informatie-vlaanderen

Spain: Plan Nacional de Ortofotografía Aérea

http://pnoa.ign.es/presentacion