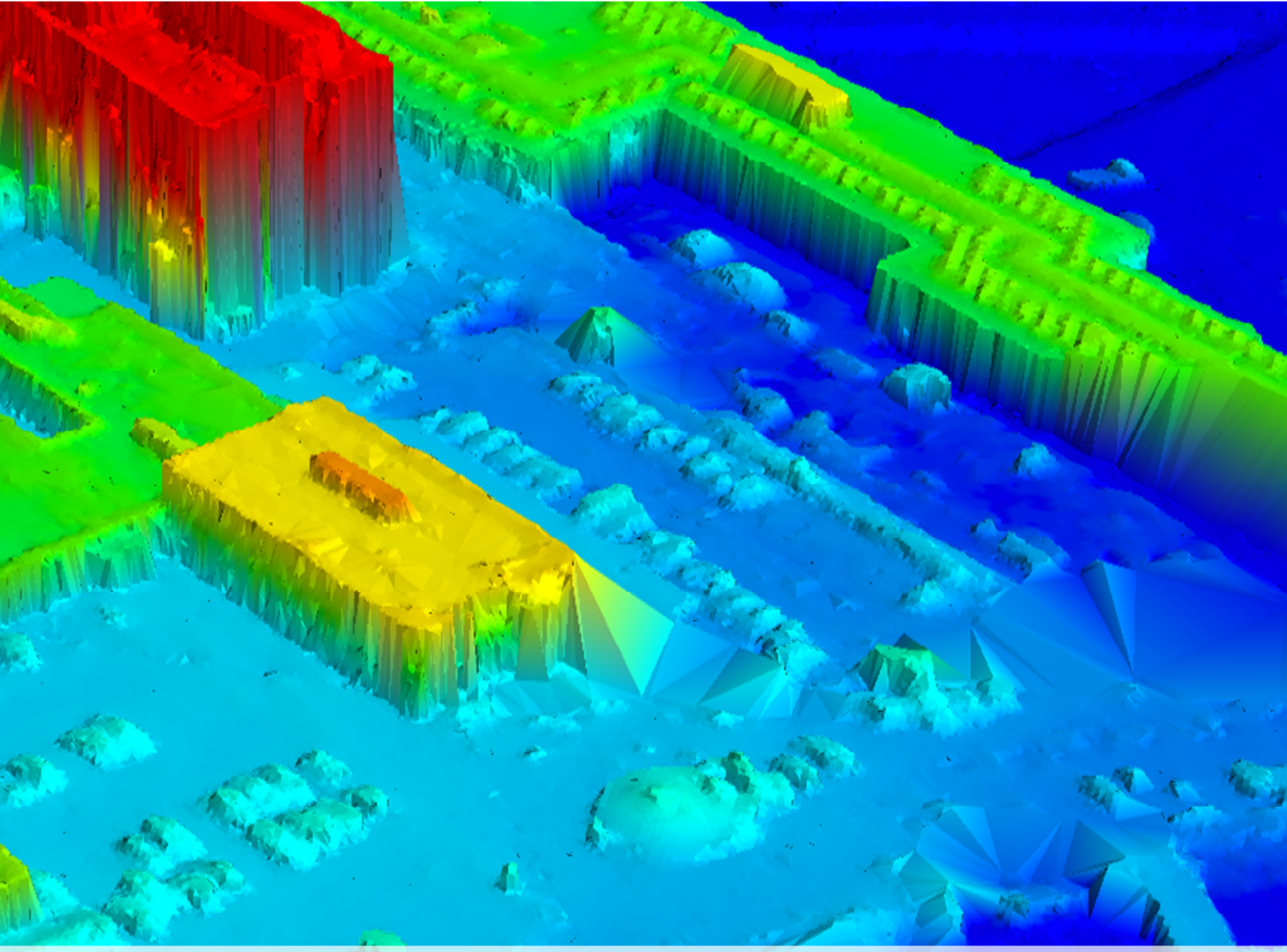
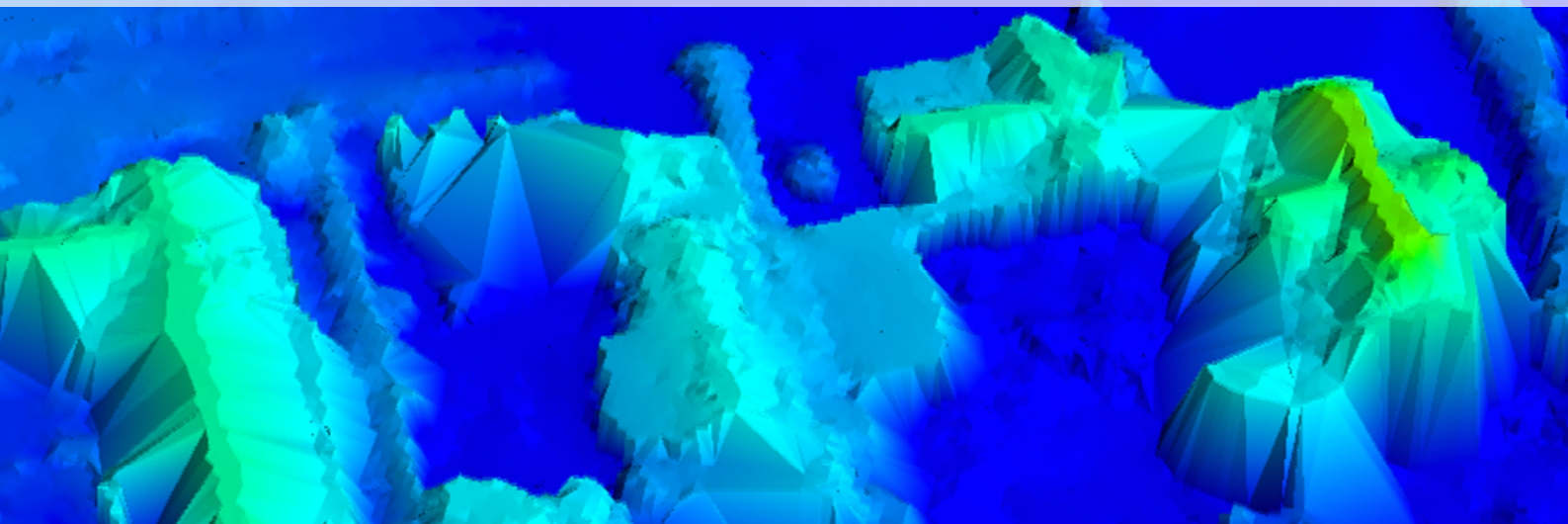


DTM Generation

UAV Point Cloud Classification

Project Report

Aalborg University
Department of Development and Planning
Surveying and Mapping, 4th semester, 2014



Front page photo:

Raw point cloud before filtering

Title page

Title:
DTM Generation - UAV Point Cloud Classification

Theme:
Master thesis

Period:
February 2014 to June 2014

Project group:

_____
Anders Westh Matthesen

_____
Kathrine Schmidt

Supervisors:
Karsten Jensen
Jens Juhl

No. of copies:
5

No. of pages:
158 pages

No. of appendices:
8 (Appendix A-H incl. DVD)

Abstract:

This project is about DTM generation based on a photogrammetrically produced point cloud. The project originates from a desire to improve the DTM algorithm proposed in the previous project "*Terrain Modelling – DTM Generation using UAVs*" by Matthesen and Schmidt (2014).

In the previous report, a number of problems were experienced. Based on investigations of these problems, a new and optimized algorithm has been proposed. In the proposed algorithm, two steps are used to remove the non-terrain points.

The first step uses a surface-based filter to remove the big clusters of non-terrain points like e.g. houses and trees. The used order of the fitted surface polynomial is automatically adjusted according to the terrain.

The second step of the algorithm removes the remaining non-terrain points by using a slope-based filter which automatically adapts itself according to the slope.

The optimized algorithm is implemented in a GUI program written in Python and Cython. Using this program, the algorithm was applied to different test-areas, and the quality of the algorithm could be assessed.

Preface

This project report is the master thesis written on the 4th semester of The Master of Science program in Surveying and Mapping at Aalborg University. The project period was from February 2014 to June 2014.

The master thesis continues the work from the 3rd semester about DTM generation using UAVs. The 3rd semester report can be found at the DVD in the folder: *'DVD:\previous_report\'*

The report is of interest for people with interest in DTM generation and especially DTM generation based on point clouds generated from photogrammetry. It is expected that the reader have a basic knowledge of terrain modeling.

An appendix folder is created which supplements this report. The appendices are numbered using the letters A-H. The folder with appendices also contains a DVD (Appendix H). The following is an example of a reference to a document on the DVD: '*DVD:\folder_name\filename*'.

Source references are indicated by author's last name, year of publication and page number. The following is an example of a source reference: (Liu 2008, 13-14). For sources relating to a single sentence, the source is placed before the dot, while sources covering an entire paragraph are placed after the dot. Quotes are marked this way: *"This is a quote"* (source reference). If words are removed from a quote, it is marked with three dots in parenthesis: *"Some words (...) are missing"* (source reference). Figures produced by others have a source reference in the figure text. The sources can be seen more detailed in the literature list.

Throughout the project period, a number of scripts and programs are written in Python 2.7. In order to run the scripts, it is recommended to install the Python distribution for scientific computing called *Anaconda 1.7.0 (64 bit).* By installing Anaconda, the extension libraries used in this project are already pre-installed. Anaconda can be downloaded for free at the web address http://continuum.io/downloads. Alternatively, a windows installer can be found at '*DVD:\python_install\Anaconda-1.7.0-Windows-x86_64.exe*'.

The project group would like to thank the consulting engineering company COWI for providing data for this project. Especially, the project group would like to thank the following people:

- *Jesper Falk*, Head of Surveying at COWI.
- *Mevludin Mesha Besic*, Specialist at COWI.

Anders Westh Matthesen
Kathrine Schmidt

Table of Contents

Chapter 1: Intro

1.1 Introduction

Danish surveying companies make increasingly use of *unmanned aerial vehicles* (UAV) for surveying. As a result of price and legislation, the typically used UAV is a small lightweight *drone* equipped with a cheap consumer camera. By doing correlation between images and self-calibrating bundle block adjustments in photogrammetry software, it is possible to create height models and orthophotos based on the images taken by the camera drone.

Figure 1 – The senseFly eBee camera drone used in the previous project (Matthesen og Schmidt 2014, 9)

The height model (point cloud) produced by most photogrammetry software is a *digital surface model* (DSM). However, a *digital terrain model* (DTM) is often preferred, for instance when creating an orthophoto, or when the architect or engineer needs a design basis for e.g. earthmoving or flooding.

In the previous project "*Terrain Modelling – DTM Generation using UAVs*" by Matthesen and Schmidt (2014), it was found that the consulting engineering company *COWI* currently uses a quite manual procedure when creating a DTM based on the high density point cloud from the photogrammetry software. They simply draw polygons around all non-terrain objects like houses and trees. Afterwards, all points inside the manually drawn polygons are deleted.

COWI have software which can automatically create a DTM based on a point cloud produced with aerial laser scanning. However, the algorithms used in the software cannot handle the photogrammetrically produced point cloud from the drones, possibly because of the irregular point distribution or the high density. (Matthesen og Schmidt 2014, 5)

At the annual professional meeting for Danish chartered surveyors (Fagligt Møde 2014) held from the 31st of January to the 1st of February 2014, it was clear that not only COWI struggles when creating DTMs based on data from camera drones. It seems like a general problem encountered by multiple surveying companies. Furthermore several articles states that filtering of LiDAR data is still a challenge, especially in steep and complex landscapes with forest cover

and in urban areas with large buildings (e.g., (Maguya, Junttila og Kauranne 2013), (Chen, et al. 2012), (Li, et al. 2013)).

In the mentioned AAU project about DTM generation by Matthesen and Schmidt (2014), an algorithm for automatic DTM creation was proposed. This present project originates from a desire to improve the algorithm proposed in the previous report. In other words, this project can be seen as an extension of the previous project.

1.1.1 Previous project about DTM generation

In the previous project by Matthesen and Schmidt (2014), the proposed filtering algorithm for DTM generation works by using a *surface-based filtering* approach. In general, the algorithm works by fitting a surface polynomial to the point cloud. The surface polynomial is fitted using least squares adjustments. Initially, a surface is fitted with equal weights for all points. Since there are more points than the number of unknowns describing the surface, a residual exists for each point. A positive residual means that the point is placed above the fitted surface. Similarly, if the residual is negative, the given point is placed below the surface. (Matthesen og Schmidt 2014, 154)

Next step is to use a weight function to assign new weights for each point based on the size of the residuals. The exact weight for a given residual is defined in a weight function, see Figure 2 below. From the weight function, it can be seen that the points above the fitted surface (positive residuals) are given a low weight. (Matthesen og Schmidt 2014, 154)

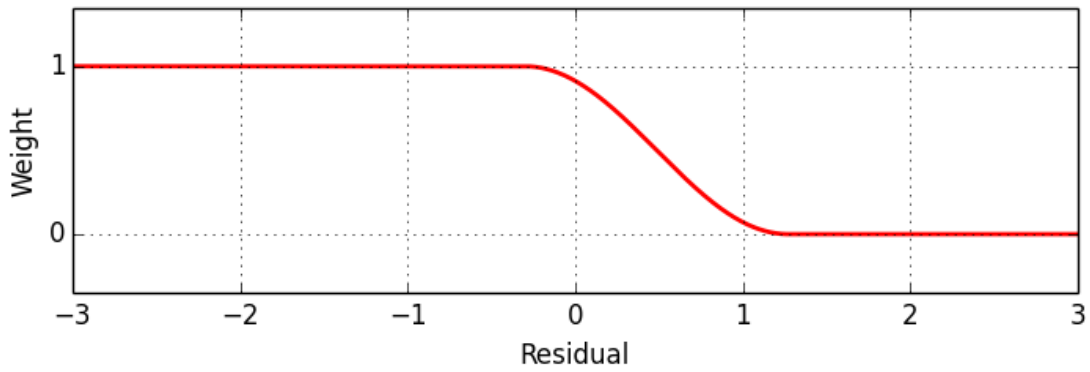


Figure 2 - One of the weight functions used in the iterative surface fitting (Matthesen og Schmidt 2014, 95)

Based on the assigned weights, a surface can again be fitted to the point cloud. Since the off-terrain points are given a low weight, the new surface will be lower than the surface fitted with equal weights. Again, the weights can be updated according to the weight function and a new least squares surface fitting can be performed. This process can be iterated until a steady state has reached, where the surface has stopped moving downwards and represents the terrain. (Matthesen og Schmidt 2014, 154)

The algorithm is working hierarchically using a two-level data pyramid consisting of a *coarse DSM* and a *fine DSM*. The fine DSM is simply all the measured points, and the coarse DSM is a low resolution point cloud created using a block-minimum filter with a grid size of 5 m. The

hierarchical approach using a data pyramid is needed since surface-based filtering cannot handle big clusters of non-terrain points. (Matthesen og Schmidt 2014, 154)

Initially, a surface is fitted to the coarse DSM using the described iterative process where off-terrain points are outweighed. Afterwards points inside a buffer zone around the fitted surface are selected as possible terrain points. Large clusters of non-terrain points are now removed. Based on the selected possible terrain points, a new surface is fitted iteratively where non-terrain points are outweighed. A narrow buffer around the surface is used to do the final selection of terrain points. (Matthesen og Schmidt 2014, 154)

It is not possible to apply the algorithm to the whole inputted point cloud, since the terrain will be too complex to describe with a simple polynomial. The solution has to work inside smaller squares. Each square should be so big that there is almost no risk that it contains only non-terrain points. However, at the same time, the square should not be too big, since it will then be impossible to describe the terrain with a simple polynomial. Furthermore, the computer runs out of memory if the square is too big/contains too many points. Overlap between the squares is needed, since the fitted polynomials might give a bad description of the terrain near the edges of the squares. This is especially problematic, if a non-terrain object, like a building, is placed right at the edge of a square. (Matthesen og Schmidt 2014, 154)

Based on a series of tests, the optimal parameters to use for the algorithm were found, see Table 1 below.

Step	Description	Parameters
1	Block minimum filter, where the coarse DSM is created	Grid size: 5 m
2	Surface fitting on the coarse DSM	Size of area: 100x100 m Overlap between squares: 30 m Polynomial order: 7th order Weight function: medium aggressiveness, a = -0.3 and b = 2 Iterations: 6
3	Selection of points in the fine DSM, which are inside a buffer around the fitted surface from step 2.	Lower buffer: 2 m Upper buffer: 1.25 m
4	Surface fitting on the selected points from the fine DSM in step 3	Size of area: 40x40 m Overlap between squares: 10 m Polynomial order: 6th order Weight function: medium aggressiveness, a = -0.2 and b = 2.5 Iterations: 4
5	Selection of points in the fine DSM, which are inside a buffer around the fitted surface from step 4.	Lower buffer: 2 m Upper buffer: 0.2 m

Table 1- Parameters used in the developed algorithm (Matthesen og Schmidt 2014, 155)

The whole algorithm was implemented in a Python GUI program, which in a user friendly way made it possible to apply the algorithm to a big dataset. Figure 3 shows an example of a big dataset before and after applying the algorithm.

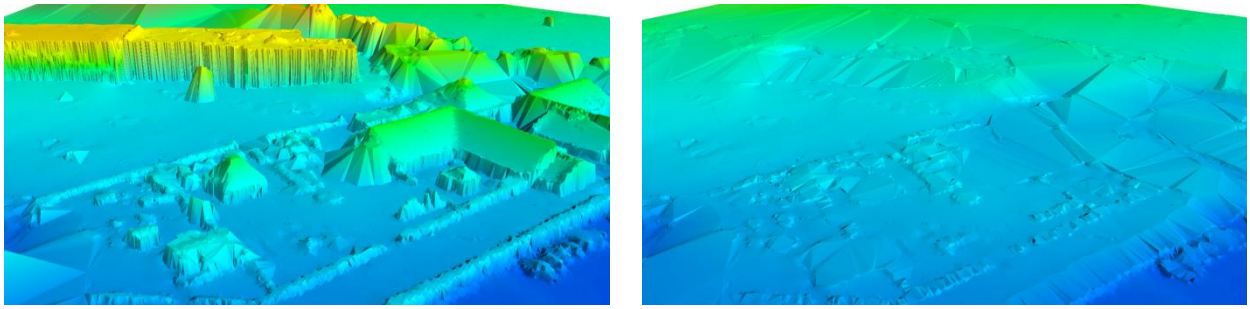


Figure 3 – Point cloud before and after filtering. (Matthesen og Schmidt 2014, 155)

In general, the algorithm in the earlier project by Matthesen and Schmidt (2014) does a good job removing non-terrain points, as seen in Figure 3. There is, however, room for improvement of the algorithm. Throughout the project by Matthesen and Schmidt (2014) a number of problems were experienced, and the discussion suggests that a more intelligent algorithm should be developed.

1.1.2 Problems with the algorithm from the earlier project

As mentioned, a number of problems were encountered during the development of the previous algorithm. Below is a list of problems which should be solved in order to improve the performance of the algorithm:

Problem 1 - Remains of non-terrain points because of buffer: In the point cloud from the camera drone a lot of points are measured on the sides of non-terrain objects, mostly on houses, vegetation and cars. This is caused by many oblique pictures of the same object from different angles. When the upper buffer of 20 cm is applied to the dataset in the final selection of terrain points, these side measurements are included in up to 20 cm over the fitted surface. The result is that the DTM is not smooth and has a lot of bumps, especially in areas with small family housing.

Figure 4 - Remains of non-terrain points because of buffer

Problem 2 - Remains of non-terrain points on big buildings: Some places the algorithm struggles removing the non-terrain points on big buildings covering large areas. The problem only exists if the big building is a low-rise building. A tall building is not that hard to remove in comparison, since it cannot be mistaken as a hill.

Before ***After***

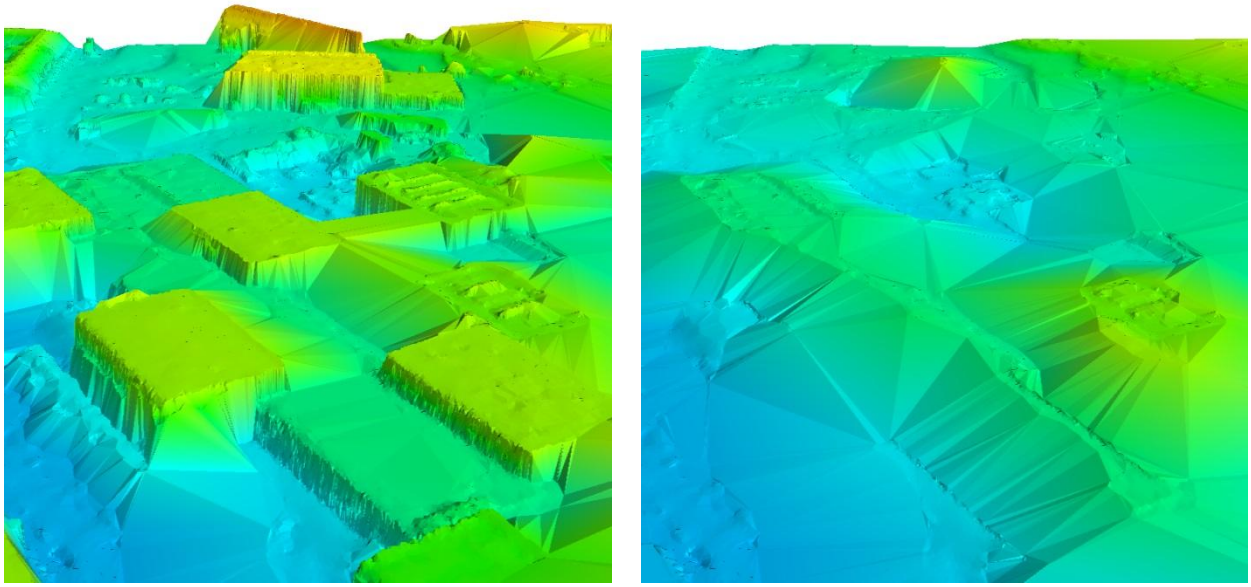


Figure 5 - Remains of non-terrain points on big buildings

Problem 3 - Fixed to one terrain type: The choice of one specific polynomial order means that the algorithm is only designed to fit to landscapes with specific characteristics. This can be problematic in complex landscapes which cannot be described with the chosen order of polynomial. Furthermore, it could be an advantage to use a low order polynomial for simple landscapes, because this makes it easier to remove non-terrain objects. In other words, the algorithm should be able to adapt its parameters according to the specific types of terrain.

Problem 4 - Non-correct surface fitting at the edges of a square: The surface fitting algorithm works inside smaller squares, covering the area of the point cloud. A surface is iteratively fitted to the terrain inside each square. If the points are equally distributed in the square, the fitted surface gives a good description of the terrain inside the middle of the square. However, the surface might give a bad description of the terrain near the edges of the square. The reason is that a non-terrain object placed right at the edge will be mistaken as a hill. To overcome this problem, the squares are slightly overlapping. The problem is, however, only solved if the points are evenly distributed in the square. If this is not the case, non-terrain points might be selected by the algorithm, see Figure 6.

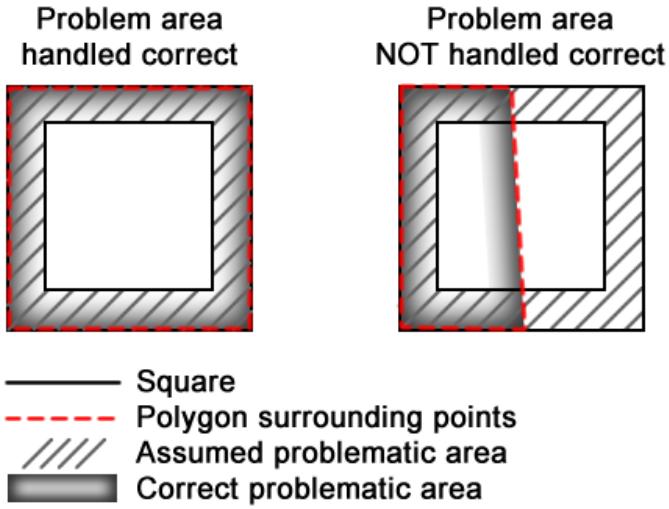


Figure 6 – Problems with surface fitting outside the surrounding polygon.

Problem 5 - Discontinuities and sharp edges in the terrain: The algorithm assumes that the terrain can be described with a smooth surface polynomial. As a result, the algorithm cannot handle discontinuities and sharp edges in the terrain.

Problem 6 - Memory errors: In some areas the algorithm did not work because of memory errors. The number of points in a square could be so big that the computer had problems calculating the least squares solution. This is mainly a question about the computers amount of RAM, but even though the calculations were made on a computer with 16 GB RAM there were memory problems.

Problem 7 - Speed issues: Connected to the memory errors, the speed of the algorithm can be discussed. The algorithm is relatively slow, which made testing a slow process. This means that the algorithm is not fully tested, because you do not run it once more just to try e.g. another polynomial order. If the algorithm could be optimized to run faster, testing could be done more easily.

1.1.3 Legislation for drone operations

It is not legal for everyone to buy a drone and use it everywhere in Denmark. If a company wants to use a drone, they will, in most cases, need a permit from the Danish Transport Authority. The 20th Marts 2014 the guidelines for commercial use of drones were updated. The biggest difference is an added distance requirement towards the royal family's properties, police stations, prisons and military areas. Furthermore there is also added a distance requirement to accident scenes where police or rescue service works. The requirements for education of drone pilots are toughened and for every drone in the air there must be a drone pilot on the earth. (Danish Transport Authority 2014)

The flying height is still maximum 100 meters above terrain. Before the new guidelines was published there was a general talk between surveyors that the limit would be raised to 120

meter, but as seen in the guidelines this is not the case. The limit of 100 meter gives some limitations in which tasks a drone is useful. (AIM 2014)

Different versions of the dispensation from the Danish Transport Authority exist, but the most common for surveying companies is the A1 category. Here the most important rules are:

- Lives or property must not be endangered during flight with the UAV.
- The UAV should be within the pilot's line of sight, and if two drones are used 2 pilots is needed.
- The maximal flight height is 100 m above terrain.
- Flight within 5 km from public runways or 8 km from military runways requires special permission from The Danish Transport Authority.
- Operations must not be performed between sunset and sunrise without a special permission.
- Flights closer than 150 m towards the royal family's properties, police stations, prisons and military areas, require permission.
- Flight closer than 200 m towards accident scenes where police or rescue service works is not allowed.
  (AIM 2014)

1.2 Initial problem statement

As mentioned, this present project originates from a desire to improve the algorithm proposed in the previous project "*Terrain Modelling – DTM Generation using UAVs*" by Matthesen and Schmidt (2014). The previous project was about creation of an algorithm which can create a DTM based on a point cloud from a camera drone. It is therefore decided that the focus in the present project still is to create an algorithm which can filter a point cloud from a camera drone. This choice does not mean that the filter algorithm cannot be applied to LiDAR data. It does however mean that the algorithm cannot rely on LiDAR-information such as intensity and multiple echoes.

Section 1.1.2 lists a number of problems which should be solved in order to improve the performance of the algorithm by Matthesen and Schmidt (2014). The focus of this present project is to solve as many of the problems as possible, and thereby improve the previous algorithm. In order to do this, the problem in the previous algorithm has to be investigated further, and filter theories has to be studied. Based on these investigations, various solutions to the problems can be found. These initial studies needed before the improved algorithm can be created, can be described using the following initial problem statement:

What filtering methods exist, which can handle a photogrammetrically created point cloud?

How can the problems experienced in the algorithm by Matthesen and Schmidt (2014) be solved, and how can the presented filter theory help solving some of the problems?

At a general level, how can the solutions to the problems be combined into an optimized algorithm for DTM generation?

Chapter 2: Initial Studies

2.1 Introduction

The purpose of this chapter with initial studies is to answer the initial problem statement and form the basis of a final problem statement.

According to the initial problem statement, different filtering methods for photogrammetrically produced point clouds should be studied. The study is conducted through literature studies of some of the existing filtering algorithms. There are a lot of different methods, and it is impossible to explain all of them in this report, and therefore the focus is on the different principles with examples of concrete algorithms under each principle. The studies of filtering principles will be presented in a theory section. The theory section will also describe the advantages and disadvantages of each of the filtering principles.

When the general knowledge about filtering principles is gathered, the problems experienced in the algorithm by Matthesen and Schmidt (2014) will be discussed. Both experience from the previous project by Matthesen and Schmidt (2014) and the knowledge from the theory section will be used to find solutions to each problem.

The discussion of how the problems can be solved is used to construct some suggested solutions to a new and optimized algorithm. The suggested solutions can either focus on the solution of one problem or try to solve several problems at once.

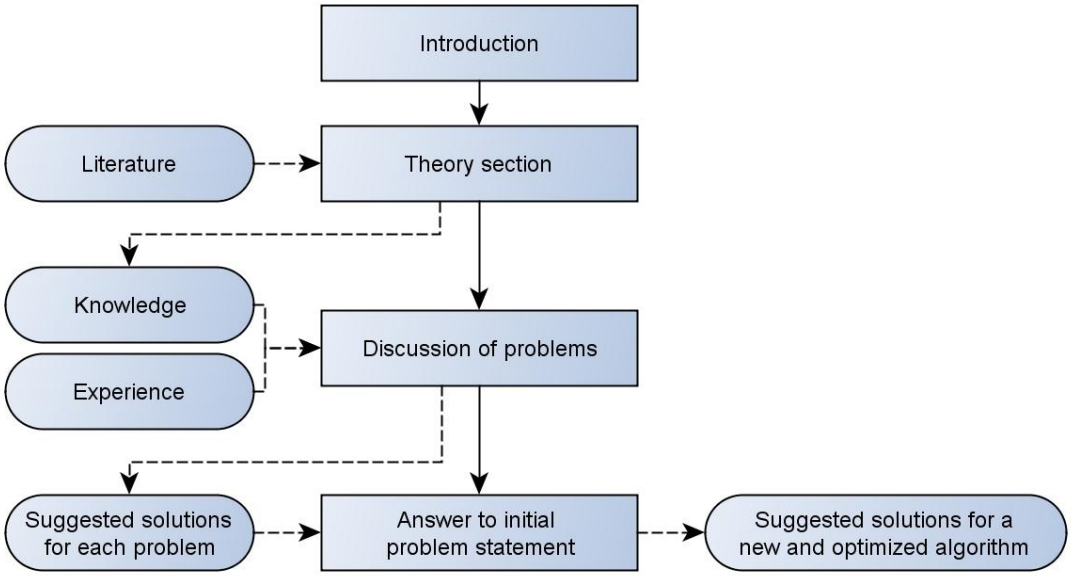


Figure 7 – Illustration of the initial studies conducted in order to answer the initial problem statement

2.2 Theory section

This section will through literature studies investigate various filtering principles. The focus is to find the advantages and disadvantages for each principle, which can be used when solutions to the problems are discussed in section 2.3 under.

Before the investigation of the filtering principles, the two different types of errors are defined.

2.2.1 Errors in classification

In filtering and classification of a point cloud (or raster dataset), errors can be divided into two groups; *errors of omission* and *errors of commission*. To understand the nature of the two types of errors, the definition from the dictionary is used.

Error of omission is explained as: *"A mistake that consists of not doing something you should have done, or not including something such as an amount or fact that should be included"* (Cambridge University Press 2014).

Error of commission is explained as: *"A mistake that consists of doing something wrong, such as including a wrong amount, or including an amount in the wrong place"* (Cambridge University Press 2014)

When filtering a point cloud and classifying the points as terrain or non-terrain, the errors can be explained as (Zhang og Whitman, Comparison of Three Algorithms for Filtering Airborne Lidar Data 2005, 3):

Error of omission: A terrain point is classified as non-terrain
Error of commission: A non-terrain point is classified as terrain

In filtering algorithms, it is in general considered that it is better to classify all non-terrain points as non-terrain, even at the cost of classifying a few terrain points as non-terrain. Therefore, the focus in most of the filtering algorithms is to minimize the number of commission errors. (Sithole and Vosselman 2003)

2.2.2 Filtering principles

Today, several different filtering algorithms exist which can be used to classify a point cloud into terrain and non-terrain points. The methods can be grouped based on of the principle used in the algorithm. On a general level, the filtering algorithms can be divided into algorithms working with either raster datasets or raw point clouds as input. In the previous project by Matthesen and Schmidt (2014) the algorithm used the point cloud as input. However, there might be some advantages of using raster datasets instead. In order to investigate this further, it is decided to look into the pros and cons for raster datasets versus a raw point cloud:

Pros and cons for algorithms working on raster image	
+	The filtering runs faster than filtering based on the point cloud.
+	Data is, since it is gridded, equally distributed throughout the area.
÷	When interpolating the point cloud to a grid raster format there is a significant loss of information, which leads to errors in the resulting classification. The problems occur when elevation data are interpolated between terrain and non-terrain points. This means that the elevation difference between terrain and non-terrain will be reduced and as a result it is harder to correctly identify and remove the non-terrain points.
÷	Areas with no measured points (holes in the dataset) will be disguised among the other data.
Pros and cons for algorithms working on the point cloud	
+	There is no loss of information.
+	Areas with no measured points are still visible.
÷	The method is computationally more expensive than the raster algorithms. This can e.g. result in a longer computing time.

Table 2 – Pros and cons for raster and point cloud algorithms (Vosselman 2000, 3) (Liu 2008, 9)

There are pros and cons for both raster and point cloud algorithms, but the loss of information in the raster algorithms is a serious disadvantage. When working with data from a camera drone (or photogrammetry in general), there is a big chance that no points exist in forested areas, as experienced in Matthesen and Schmidt (2014). Since the DTM cannot be trusted in such areas, it is wished that the holes in the dataset are still visible after filtering. Based on these considerations, it is decided that the further project work should focus on filtering of a raw point cloud, rather than a raster dataset. This also means that the following discussion about filtering methods will focus on algorithms working on a point cloud.

The filtering algorithms suitable for irregular point clouds can be grouped into several categories based on the used principles. The categories are:

- Morphological and slope-based filtering
- Progressive densification
- Surface-based filtering
- Segment-based filtering

In the following, the principles will be described with focus on advantages and disadvantages.

2.2.2.1 Morphological and slope-based filtering

Morphological filtering is based on the theory of mathematical morphology, which is used to analyze grayscale images. Nevertheless the method can also be used on a raw point cloud. *Slope-based filtering* derives from the morphological operations and to some extent looks similar to a morphological filter. Nevertheless there are differences which will be explained in

this section. Both the morphological filtering and the slope-based filtering build on the same basic idea:

"The basic idea (…) is based on the observation that a large height difference between two nearby points is unlikely to be caused by a steep slope in the terrain. More likely, the higher point is not a ground point." (Vosselman 2000, 2)

The two basic operations in mathematical morphology are *erosion* and *dilation*. Both operations are performed on a neighborhood defined by a structure element. The structure element is defined as a figure (such as a circle or a square) or the nearest X points[1]. It is often circular since it *"…is more suitable for removing trees and preserving terrain."* (Chen, et al. 2007, 2). The idea in the morphological operations is to update the elevation of a point to match other points within the structure element. Erosion is used to make high regions smaller by lowering the elevation of the point to match the lowest point inside the structure element, see Figure 8b. Dilation is on the other hand used to expand high regions by lifting the point to match the elevation of the highest point inside the structure element, see Figure 8c. (Mongus og Zalik 2011, 2-3)

Based on the two operations the processes *opening* and *closing* can be defined. Opening is erosion followed by dilation, and the closing operation is the opposite; dilation followed by erosion. The opening and closing processes are often used to remove outliers in a dataset. Opening removes high outliers and closing removes low outliers. The processes and the operations can be applied to a dataset in different sequences to remove specific details without changing the surface of the terrain. (Chen, et al. 2007) (Mongus og Zalik 2011)

The slope-based filter by Vosselman (2000) is closely related to the morphological erosion operation. Here a point is classified as either terrain or non-terrain depending on the height difference between neighboring points. The acceptable maximum height difference between two points is defined as a function of the horizontal distance, from now on called the *evaluation function*. (Vosselman 2000) Figure 9 illustrates how the slope-based filter works. The structure element with the evaluation function is placed at every point in the point cloud. If there are points under the function inside the structure element, the point is classified as non-terrain (marked with red at Figure 9). On the other hand, a point is classified as terrain, if there are no points below the function (marked with green at Figure 9).

[1] X refer to a number

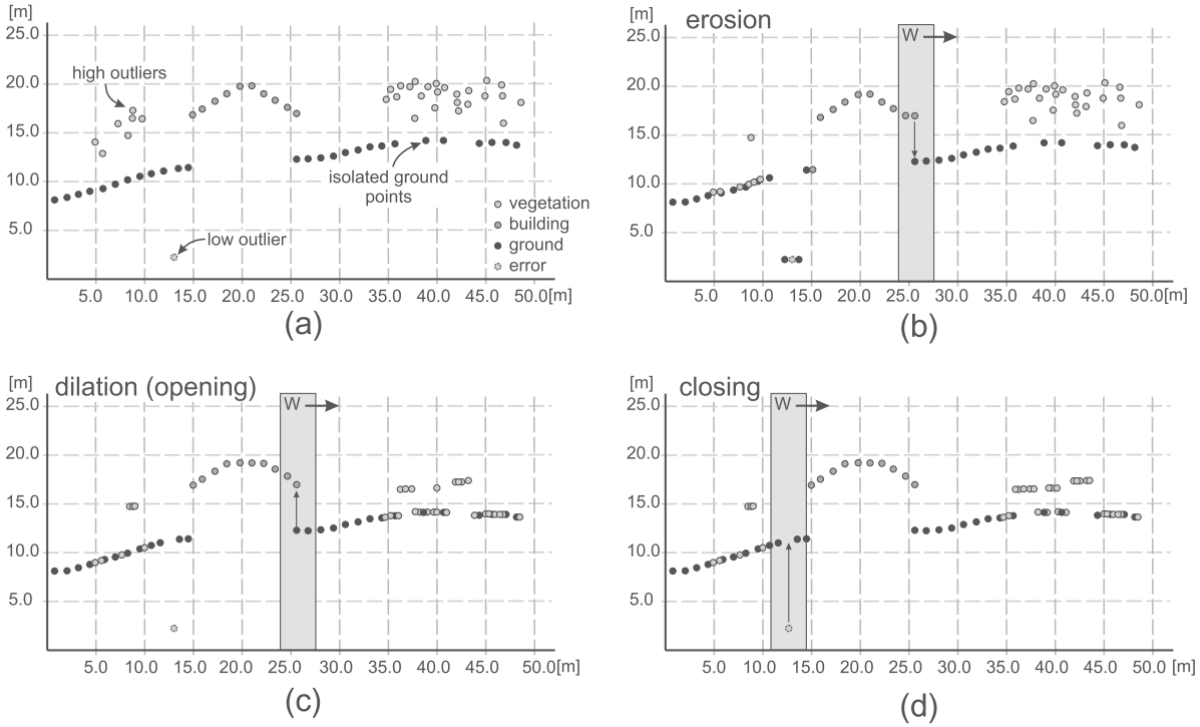


Figure 8 – "Cross section of LiDAR pointset (a), erosion of pointset (b), dilation of eroded pointset i.e. morphological opening (c), and morphological closing of opened pointset (d)" (Mongus og Zalik 2011, 3)

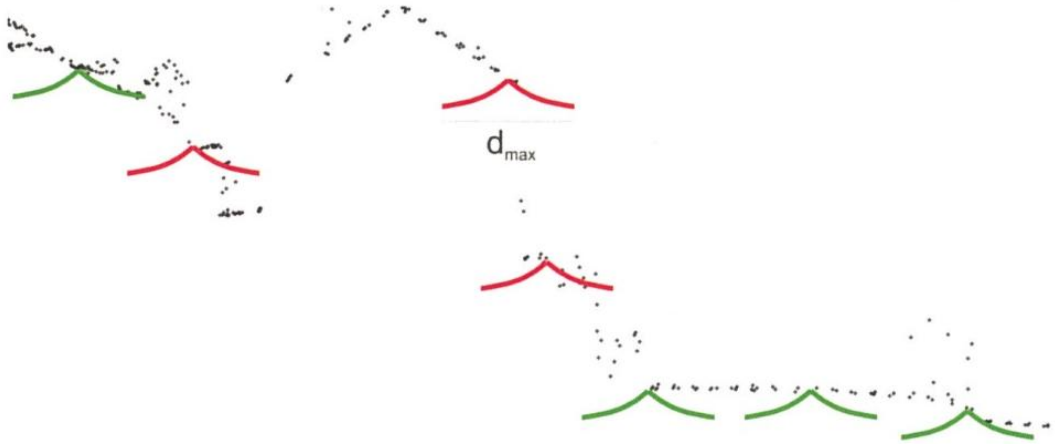


Figure 9 - A structure element with the size d_{max} is positioned at each point. For selected points, the evaluation function is shown. A green evaluation function means that the point is a terrain point, since no points are below the evaluation function. A red evaluation function means that the point is classified as non-terrain, since there are points below the function. (Vosselman [1] and Mass 2010, 141)

The biggest difference between the morphological filters and the slope-based filters is whether the original measurement of the elevation is preserved or if it is updated to match the terrain in the best possible way. The pros and cons are almost identical for these two filter types, so they are listed together below:

Pros and cons for morphological filtering and slope-based filtering	
+	The methods are simple and easy to implement.
+	Works well in flat terrain.
+	Remove small objects effectively with a small window size, without flattening the terrain.
+	Are good at removing outliers.
÷	It is problematic to select an optimal window size for the structure element. A small window size can be used to remove small object points such as trees and cars, but cannot be used to remove objects bigger than the window size, such as buildings. A big window size will on the other hand be able to effectively remove buildings, but it will have a tendency to remove valid terrain points too, and as a result flatten the terrain.
÷	For the slope-based filtering, the definition of the evaluation function is a challenge, but it can usually be determined through training areas.
÷	Do not work well when the slope of the terrain increases and especially in steep and forested areas.
÷	Tends to remove isolated ground points.

Table 3 – Pros and cons for morphological filtering and slope-based filtering (Liu 2008, 7) (Chen, et al. 2007, 1) (Mongus og Zalik 2011, 3)

To overcome the problems with window size and threshold function, Sithole(2001) has made a modified slope-based filter where the evaluation function changes according to the slope of the terrain. The method is called *Slope Adaptive Filter*. In the group of morphological filtering, several algorithms have been developed to overcome the problem with the window size. The filters generally deal with the use of progressively increased windows sizes. (Killian, Haala og Englich 1996), (Zhang, Chen, et al. 2003) and (Chen, et al. 2007) are examples of such algorithms.

2.2.2.2 Progressive densification

Some filtering algorithms start with a small subset of points which are known terrain points. The subset of points is iteratively densified with new points, based on certain threshold values. Such types of algorithms are called *progressive densification* algorithms. (Matthesen og Schmidt 2014, 67)

One of the most popular algorithms for progressive densification is the *Progressive TIN Densification* algorithm created by Axelsson (2000). Here, the initial subset of points is found using a block minimum filter with a grid size of 50-100 meters. The initial subset of points is

used to create a coarse TIN representing the terrain. Afterwards, multiple iterations are done, where a point is added to each TIN facet if a given criterion is met. The criterion mainly depends of the given point's distance to the facet plane d and the angles α, β, γ, see Figure 10. (Axelsson 2000)

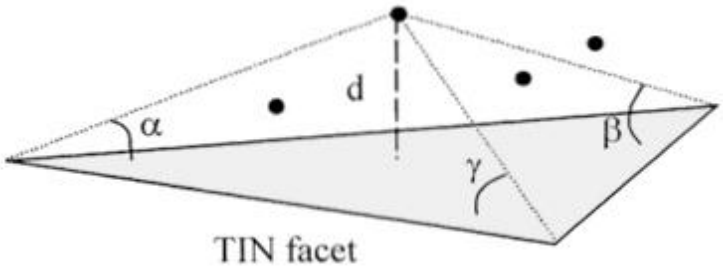


Figure 10 – Illustration of a point's angles to the nodes and the distance to the TIN facet plane (Axelsson 2000, 3)

The method for progressive TIN densification created by Axelsson (2000) can handle discontinuities in the landscapes. This is a major advantage over other types of filtering algorithms. (Sithole and Vosselman 2003, 11)

It is possible to handle discontinuous landscapes because the algorithm does not only look at the TIN facet surrounding the point, but also at surrounding TIN facets. This is done by mirroring the point to the closest node point (3D distance). The mirrored point is inside a different TIN facet, and the trick is to look at the deviation from this TIN facet as well as the TIN facet with the original point. The principle is illustrated at Figure 11 and Figure 12. (Axelsson 2000, 3)

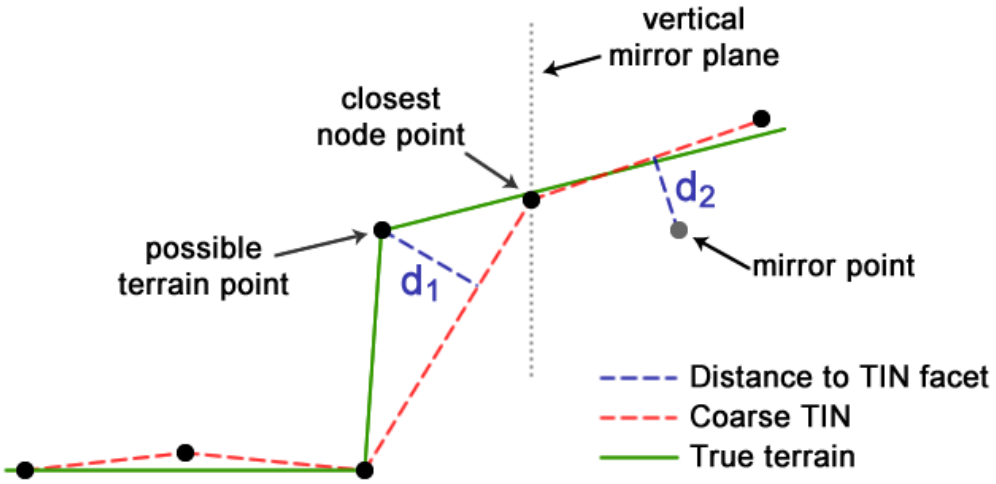


Figure 11 – Mirror points are used to handle discontinuities

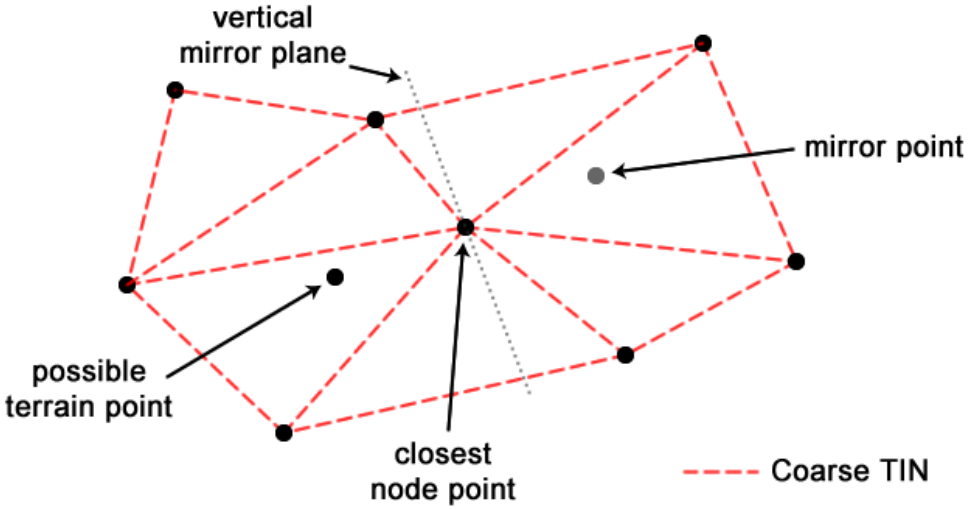


Figure 12 - Creation of mirror point seen from a top view

Sithole and Vosselman (2003) have compared different filtering algorithms. The test showed that progressive TIN densification using the algorithm proposed by Axelsson was best at handling steep slopes. However, the test also showed that the algorithm has a bias towards classifying non-terrain objects as terrain. Most other filtering algorithms are designed to aggressively remove as many non-terrain objects as possible, "*… even if it is at the expense of removing valid terrain*" (Sithole [2] og Vosselman 2003, 5).

Mostly, it is desired that almost all non-terrain points are stripped from the terrain model, even though this means that some terrain points has also been removed. It is therefore assessed that the bias towards classifying non-terrain objects as terrain is a weakness in the TIN densification algorithm.

Below is a table of pros and cons for using Progressive TIN Densification to produce a digital terrain model:

Pros and cons for Progressive TIN Densification algorithms	
+	Some implementations of the algorithm can handle discontinuities in the landscape.
+	Good at handling steep slopes.
÷	Bias towards classifying objects as terrain (commission error).

Table 4- Pros and cons for progressive TIN densification
(Sithole and Vosselman 2003, 11) (Sithole [2] og Vosselman 2003, 5)

2.2.2.3 Surface-based filtering

The algorithm developed in Matthesen and Schmidt (2014) is using surface based filtering. A description of the method can be found in section 1.1 .

A fact about the surface-based filters, which is not explicitly described before, is that the purpose or idea is to define a region in the 3D space where terrain points are expected to be

(Chen, et al. 2007, 1). Therefore, the output of the filter contains not only terrain points, but will also include some non-terrain points, which have to be removed in another way.

Pros and cons for surface-based filtering	
+	Tend to give better results in tests, compared to other filtering methods, even though the difference is not that big.
+	When used on a coarse DSM the method is good at removing big clusters of non-terrain points.
÷	Do not work in terrain with steep slopes and large variability.
÷	Tends to misclassify small non-terrain objects and do not preserve terrain details because only a rough approximation of the surface can be obtained.
÷	Relies on some predefined parameters, which only make it work on only one type of terrain at a time.

Table 5 – Pros and cons for surface-based filtering
(Liu 2008, 6) (Sithole and Vosselman 2003, 24) (Mongus og Zalik 2011, 1)

2.2.2.4 Segment-based filtering

The segment-based filters normally consist of two steps. The first is segmentation where points in the point cloud are clustered. The second is a filtering based on the generated segments. When filtering the input is, opposite all the other filtering principle, a segment of points and not only a single point. (Lin og Zhang 2014, 4) The simplest way to filter a segmented point cloud is use the assumption that *"any points that cluster must belong to an object if their cluster is above its neighborhood"* (Sithole [3] 2005, 30).

The segmentation part of the filter is more complex and there are a lot of different methods which can be used for this purpose. Segmentation originates from image analyses, but can also be applied to irregular point clouds. Some segmentation algorithms for point clouds use additional information, such as orthophotos, or convert the point cloud into a raster format. In general the segmentation algorithms can be grouped in four groups: (Sithole [3] 2005, 67-71)

- **Pattern based techniques:** A number of geometric features are determined for each point. In addition to the location of a point the features can be the coefficients of the best fitting surface, the surface normal estimated by the best fitting plane or the reflectance of the ALS measured point. Based on these features the point cloud is divided into segments.
- **Edge detection techniques:** The algorithm search for edges in the point cloud. If the edges form a closed boundary, all points within the closed edge are segmented.
- **Graph based techniques:** These algorithms are *"based on the simple notion that points within segments are closer to each other than they are to points in other segments"* (Sithole [3] 2005, 69)
- **Region growing techniques:** Typically planer or non-planer surfaces are selected in the point cloud. Afterwards the areas are connected or grouped based on different measures (e.g. differences in normals, slopes or curvature).

These different segmentation techniques all have different pros and cons, but for segment-based filtering the general pros and cons are listed in Table 6.

Pros and cons for segment-based filtering	
+	Preserve discontinuity.
+	Works great in complex urban areas where there are steep edges between terrain and objects.
÷	In forested areas there are often too many and too small clusters after segmentation, which makes the filtering difficult.
÷	Have problems with noisy data, e.g. low vegetation.

Table 6 - Pros and cons for segment-based filtering
(Perera 2007, 24) (Lin og Zhang 2014, 4) (Sithole [3] 2005, 71)

2.2.2.5 Combined methods

Several algorithms have been developed as a combination of the mentioned methods above. This is mainly done to improve the applicability in steep and forested areas. Examples of combined principles are:

- Morphological and surface-based filtering by Zakšek og Pfeifer (2004)
- Segment-based and surface-based filtering by Tóvári og Pfeifer(2005)
- Segment-based and progressive densification by Lin og Zhang (2014)

It should also be mentioned that it is possible to use existing map data in the filtering algorithms. The existing map data can be used to remove points on known buildings and other non-terrain objects. This is done by simply deleting the points inside the polygons in e.g. a building theme. Furthermore, map data can be used to adjust parameters according to landscape characteristics.

In Denmark most of the core map datasets are released for free commercial use. This means that for instance a building theme with polygons surrounding all large buildings are freely available and map-based filtering easily could be incorporated in a filtering algorithm.

It is not possible to create a good DTM using map-based filtering alone. There will always be non-terrain objects which are not mapped because of size, because they are movable (e.g. cars), or simply because they are out of scope with the original purpose of the map. Parts of the map can also be obsolete if for instance a house is demolished or a new house is built after the map was created.

Even though map-based filtering cannot be used alone, it is a great supplement to existing filtering algorithms, since it can be used to remove the problematic big clusters of non-terrain points before applying more advanced filtering techniques.

2.3 Discussion of problems

This section discusses possible solutions to the following problems experienced in the previous project by Matthesen and Schmidt (2014):

- Problem 1: Remains of non-terrain points because of buffer
- Problem 2: Large low-rise buildings
- Problem 3: Fixed to one terrain type
- Problem 4: Boundaries
- Problem 5: Discontinuities and sharp edges in the terrain
- Problem 6+7: Memory and speed issues

Possible solutions to the problems listed above are found using the knowledge gained in the theory section (see section 0) and experience from the previous project.

2.3.1 Problem 1: Remains of non-terrain points because of buffer

The output from the algorithm by Matthesen and Schmidt (2014) contains not only terrain points, but also some non-terrain points. This is because of the basic principle of the surface-based filters. They can only be used to define a region in the 3D space where the terrain points reside. In the algorithm, non-terrain points up to 20 cm above the fitted surface are included in the model because of the used buffer. A principle sketch is shown in Figure 13.

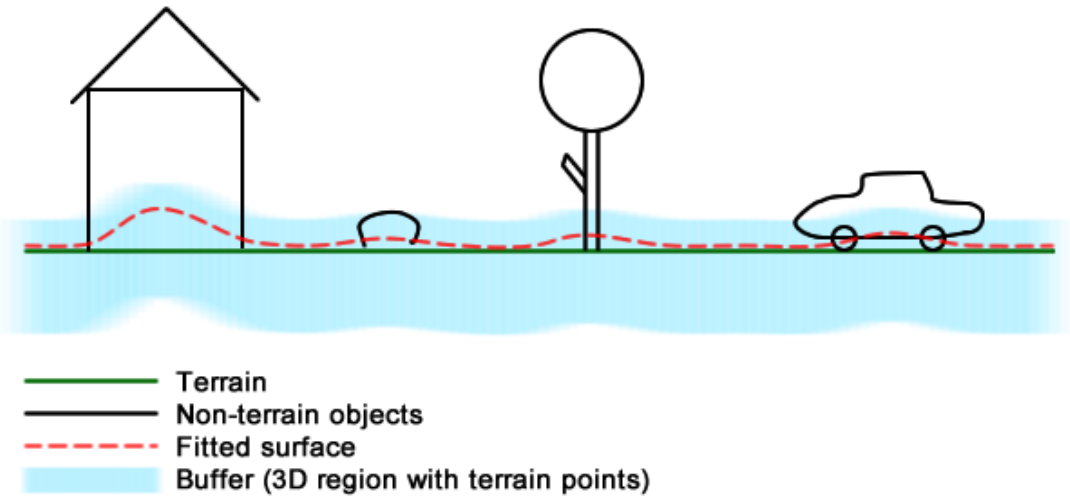


Figure 13 – Principle sketch of surface based filtering

The principle in surface-based filtering means that the filtering method cannot be used to remove the last non-terrain points in the point cloud. Because of this principle it can be discussed whether or not the surface-based filtering should be used in the filtering process. When the last non-terrain points have to be removed in another way, maybe all non-terrain points can be removed with another filtering method.

One method is the morphological and slope-based filters. According to the theory section (0) these filters are good at removing small objects when the structure element is small. A principle sketch of a slope based filter is showed in Figure 14.

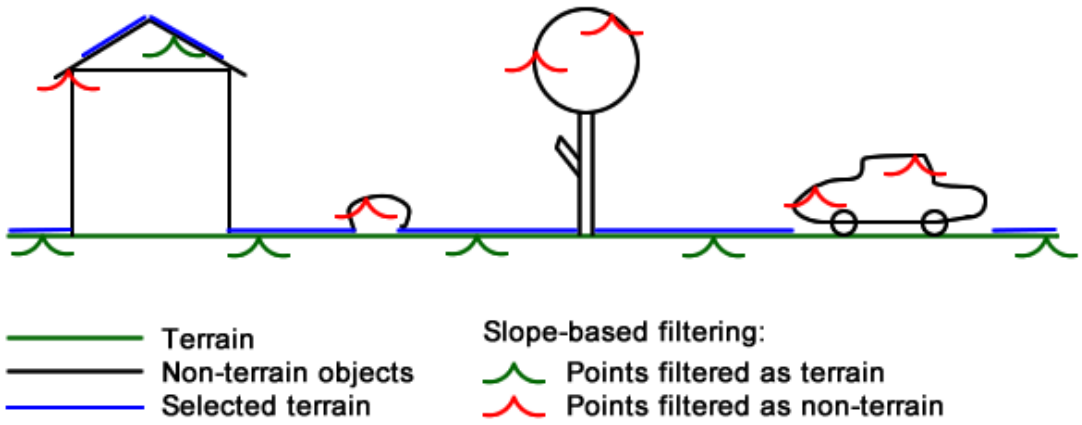


Figure 14 – Principle sketch of slope-based filtering

The principle sketch of the slope-based filtering shows, that it will work well and remove non-terrain points on small objects, but have troubles with the points on e.g. a roof top. The points on the roof top can be removed if the structure element is bigger than the house. This will however also mean that the terrain will be flattened in hilly areas. The filters therefore have problems removing big clusters of non-terrain points and at the same time preserve the details in the terrain.

If the big clusters of non-terrain points can be removed in an initial filtering step, a morphological or slope-based filter can be used to remove all the small objects, such as trees, vegetation and cars.

2.3.2 Problem 2: Large low-rise buildings

In the surface-based algorithm by Matthesen and Schmidt (2014), it was hard to remove non-terrain points on low-rise buildings covering a large area. This is somewhat expected, since the polynomial is so flexible that it will be able to crawl on top of buildings covering a large area. In other words, the large low-rise buildings covering large areas are mistaken as hills.

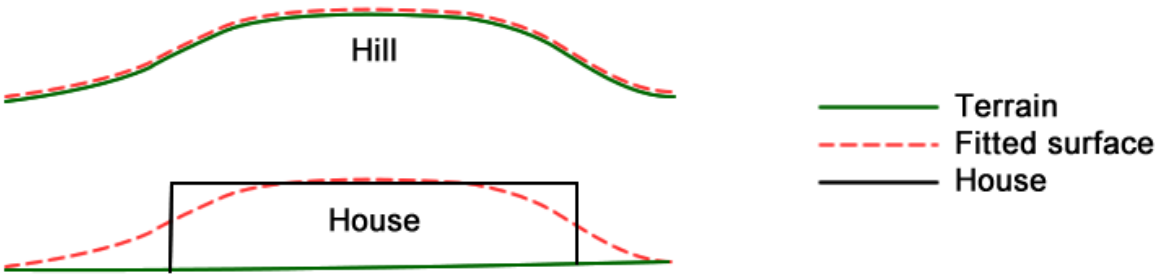


Figure 15 – The algorithm cannot differentiate between a hill and a large low-rise building.

In the previous algorithm by Matthesen and Schmidt (2014), the big clusters of non-terrain points are removed in the initial selection, where a 7th order surface polynomial is fitted to block minimum points inside 100 x 100 m squares. A 7th order surface polynomial was chosen in order to make the algorithm work in the most complex areas which can be expected in Danish terrain. However, most of Denmark is very flat, and could easily be described with a lower order of polynomial. A lower order surface polynomial will be less flexible compared to a high order polynomial. By using a surface polynomial with an order lower than 7, the surface will be less likely to crawl on top of the buildings. A solution could therefore be to make the algorithm more intelligent, so it automatically detects the order of polynomial needed. Studies in Matthesen and Schmidt (2014) shows that the best order of polynomial can be found by iteratively increasing the order of the fitted polynomial, and stop when the change in standard deviation is insignificant.

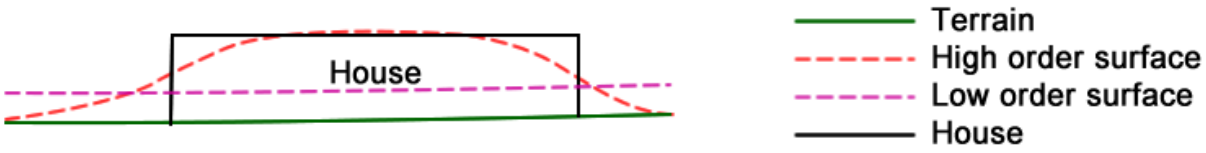


Figure 16 – A low order surface polynomial makes it easier to remove points on large low-rise buildings, because it is less flexible.

A block-minimum filter with a grid size of 5 meters is applied to the point cloud before the initial surface fitting in Matthesen and Schmidt (2014) is performed. The block-minimum filter simply picks the lowest point inside each cell in a grid. If the cell size is big enough, the result from the block-minimum filter will be a coarse DTM. By increasing the cell size, big clusters of non-terrain points are reduced, which will make it easier to filter away large low-rise buildings. The downside of this method is that it will cause problems in complex terrain, as illustrated at Figure 17 below. Since Danish terrain is in general very simple, it is assessed that it is a good idea to increase the cell size to maybe 10 meters. It is not recommended to use a very big cell size of e.g. 50 meters, because the risk of cutting away valid terrain is too big.

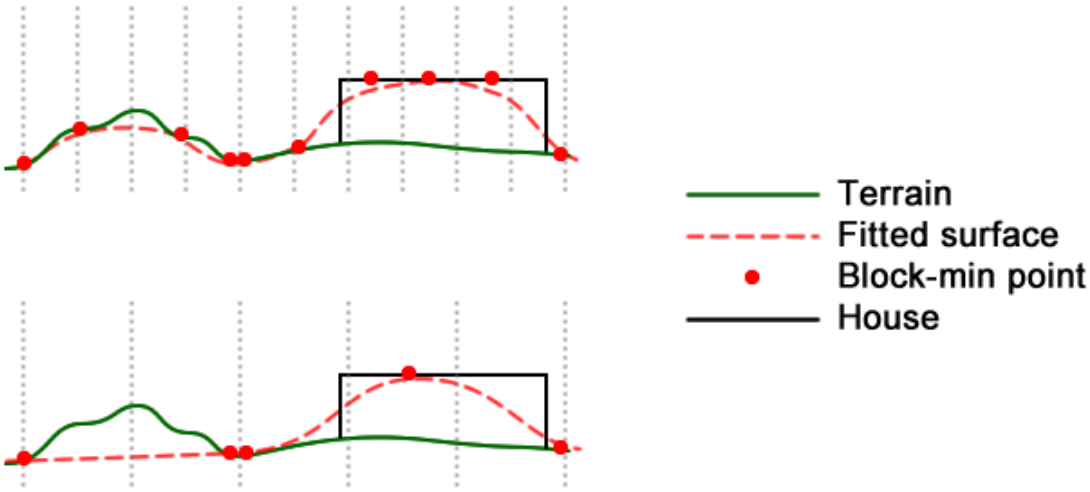


Figure 17 – A block-minimum filter with a large grid size results in fewer points on the problematic buildings, but might happen at the cost of removing valid terrain.

It is assessed that the problem with low-rise buildings covering large areas is not isolated to surface-based algorithms. Large buildings with flat roofs look similar to a discontinuous terrain. It is therefore expected that filtering algorithms like TIN densification and segment based filtering algorithms, which can handle discontinuities, won't be able to remove low-rise buildings covering large areas. Morphological filtering is not considered a good solution either, since it cannot remove big buildings without flattening the surrounding terrain. It is therefore assessed that the best solution to problem 2 is to optimize the existing surface-based algorithm.

As a final note, it should be mentioned that the large low-rise buildings could be removed using existing map data. For instance, the building theme from the Danish vector map *Kort10* could be used for the filtering. Even though most big clusters on roof tops could be removed using existing map data, there might still be big clusters left, for instance on temporary objects, vegetation and newly built houses. In this project it is decided that the algorithm should work without help from existing map data, since it makes the algorithm more versatile. In other words, it is not limited to working inside the borders of Denmark. However if the algorithm should be used for commercial use, it is probably preferred to use existing map data, in order to ensure a high success rate when filtering areas with large low-rise buildings.

2.3.3 Problem 3: Fixed to one terrain type

In the algorithm by Matthesen and Schmidt (2014) a 7^{th} order polynomial was fitted to the point cloud in the initial selection, and a 6^{th} order polynomial in the final selection. This means that the terrain must be describable with those specific polynomials if a good result should be achieved. This can be problematic since not all areas can be described in the same way. A solution could be to optimize the algorithm and make it more intelligent, as also discussed under problem 2, section 2.3.2. A way to make the algorithm intelligent is automatic detection of the needed polynomial order and also the number of iterations necessary to get the surface down to the terrain in the point cloud. In this way the hilly terrain can be described with a complex polynomial of e.g. 7^{th} order and a very flat terrain can be described with just a 1^{st} order polynomial. If there are no non-terrain points in the fitting area, only a few iterations are needed and this can save some computing time. The polynomial order can be found with help from the standard deviation. As experienced in the project by Matthesen and Schmidt (2014) the best polynomial order is reached when the standard deviation does not change significantly, see Figure 18.

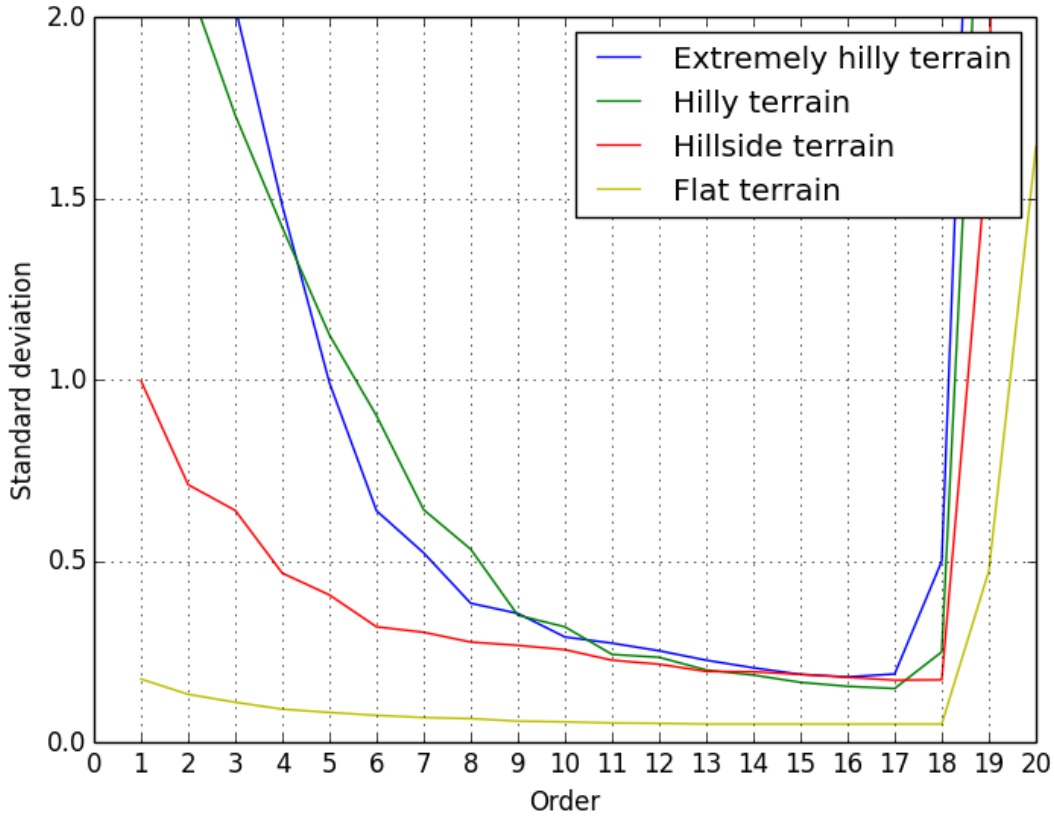


Figure 18 - The standard deviation of unit weight at different polynomial orders and with different terrain types (Matthesen og Schmidt 2014, 91)

There are other solutions to make an algorithm fit more than one type of terrain, but none of these includes the surface-based filtering approach. Both the morphological filters, slope-based filters and progressive densification evaluate a smaller area at a time and does not in the same way, as the surface-based filters, use predefined terrain shapes. The slope-based filter has the evaluation function which decides how much the terrain can increase or decrease according to the distance between two points. This means that the filter can only be used on terrain where the slope is under the allowed difference in the evaluation function. Progressive TIN densification works better in steep areas, and will to some extent also fit most terrain types. The threshold parameters for angles and the distance between a point and a TIN facet are however limiting the filter to specific terrain types. Both slope-based filtering and progressive TIN densification can be optimized in the same way as the surface-based filtering, where the evaluation function or the threshold parameters will be found automatic according to the surrounding point cloud.

2.3.4 Problem 4: Boundaries

The previous surface fitting algorithm by Matthesen and Schmidt (2014) is working inside slightly overlapping squares covering the point cloud. Inside each square, a surface is iteratively fitted to the points and gradually moving downwards until it is expected to represent the terrain.

The fitted surface gives a good representation of the terrain in the middle of the polygon surrounding the points. However, near the boundaries of the polygon, the surface might give a bad description of the terrain. The reason is that a non-terrain object will be mistaken as a hill if it is placed right at the boundary of the polygon surrounding the points, since the non-terrain object is not surrounded with terrain points. To solve this problem, the squares are overlapping, and only points from the middle of the square are used in the classification. The problem is, however, only solved if the polygon surrounding the points inside a square is the same size as the square. If e.g. the polygon surrounding the points are only covering half the square, possible non-terrain objects might be selected by the algorithm. The problem is illustrated at Figure 19.

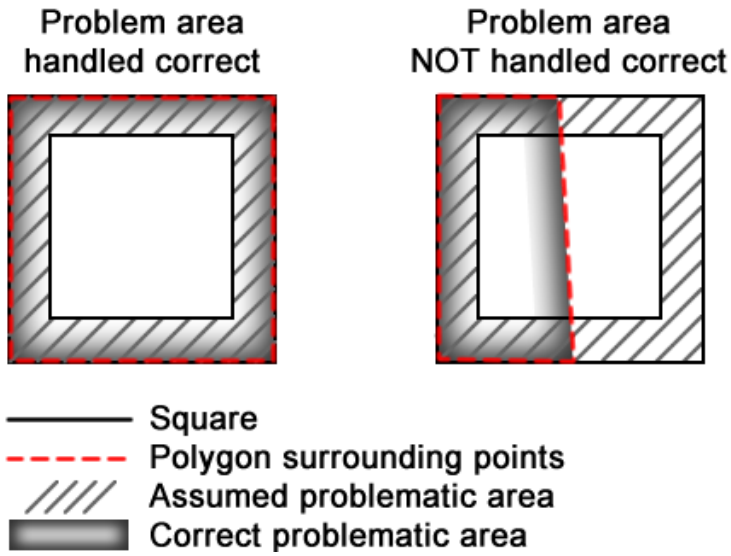


Figure 19 – Problematic areas are not handled correct in the previous algorithm, if the points are not equally distributed in the square.

As earlier mentioned, the initial selection in Matthesen and Schmidt (2014) is done using a 7th order polynomial which is quite flexible. If a less flexible low order polynomial were used, it would be less likely to have problems near the boundaries, since a low order polynomial cannot as easily bend/crawl on top of the non-terrain objects. By automatically adjusting the order of surface polynomial, as suggested in section 2.3.2, it is expected that a polynomial with an order lower than 7 will be fitted in the majority of squares. Making the previous algorithm more intelligent so it automatically detects the order needed, could therefore reduce the problems at the boundaries.

One way of making sure that the surface polynomial cannot crawl over buildings could be to use a 1st order surface polynomial. A first order surface polynomial is simply a tilt-able plane. In order to use a tilt-able plane, a constraint should be added, saying that the plane is levelled with a certain weight defined in the C matrix. If such a constraint is not added, the plane-method could end up classifying non-terrain points as terrain, see example at Figure 20.

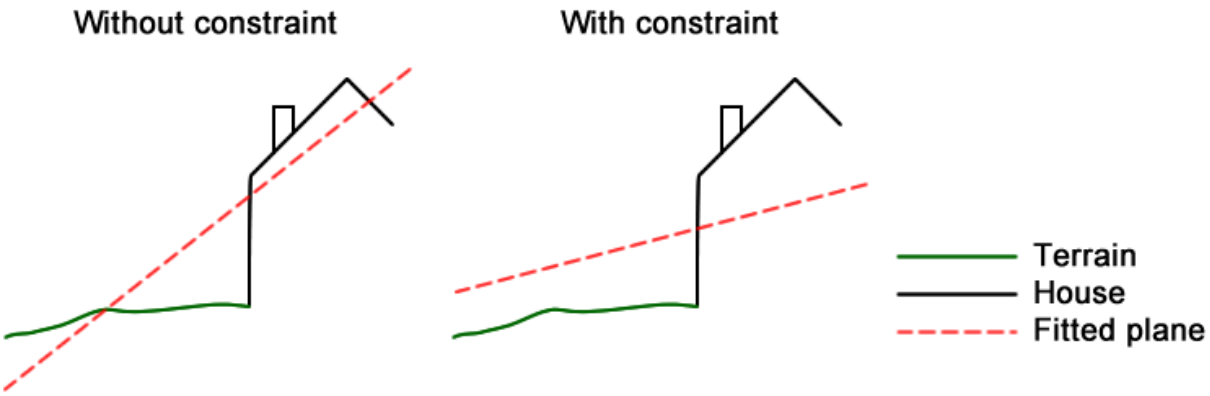


Figure 20 – Example of a plane fitted to a house placed on the boundary of the working area/square

In the plane-method, you need to compensate for the assumption that the terrain can be described as a plane. This can be done by increasing the size of the buffer used to do the initial selection. A bigger buffer size will still remove the big clusters of non-terrain points, but smaller clusters will still exist. This is probably not a problem, if small non-terrain points have to be removed afterwards anyway, as described in section 2.3.1. It is however very difficult to define the value of the weight used to make sure that the plane cannot tilt more than e.g. 40 degrees. If the weight is too high, the fitted plane will always be levelled, which will lead to misclassification of terrain points in sloped areas. If the weight on the other hand is too low, the constraint will have no effect, and the plane might end up being tilted too much, as shown at Figure 20.

As a final note, it should be mentioned that the problem with boundaries is isolated to algorithms based on surface-fitting. The problem could therefore be completely eliminated by choosing a different filtering approach like e.g. TIN densification.

2.3.5 Problem 5: Discontinuities and sharp edges in the terrain

In Matthesen and Schmidt (2014) the algorithm assumes that the terrain can be described with a smooth surface polynomial. As a result, the algorithm cannot handle discontinuities and sharp edges in the terrain. If there is a sharp edge or discontinuity, the surface-based filter will cut the upper edge off and make the edge less sharp, see Figure 21.

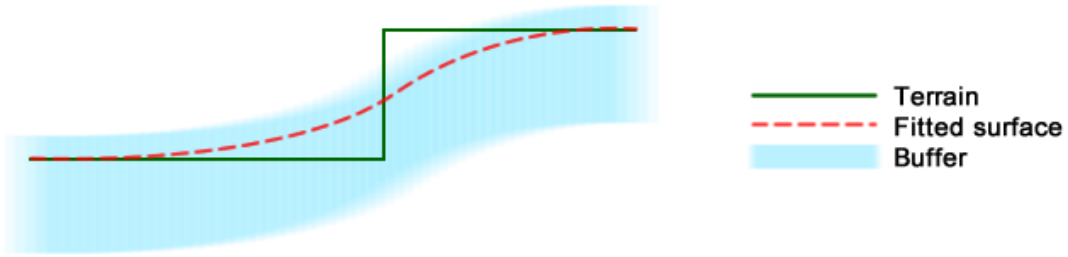


Figure 21 – Surface-based filtering will cut the upper edge.

The surface-based approach cannot handle discontinuities and sharp edges on its own. Progressive TIN densification and segment-based filtering are filtering methods which can handle discontinuities. But they also have disadvantages which should be considered.

Progressive TIN densification has a tendency to classify non-terrain points as terrain, and because of the ability to handle discontinuities it is assumed that the method have difficulties in removing low buildings with flat roofs because they look similar to a discontinuity in the terrain. A solution to that problem can be to remove clusters of non-terrain points with the surface-based filter, and afterwards remove small objects with the progressive TIN densification.

The segment-based filters first have a big challenge in how to classify the point cloud into segments. Afterwards the segments have to be filtered into terrain segments and non-terrain segments. This filtering is also a challenge since the input is not just points, but entire segments.

The discussion about discontinuities is not that important in Denmark. There are very few places with huge discontinuities, and therefore it is assessed that it may be better to solve some of the problems seen more often, than to solve this problem which only rarely occurs.

2.3.6 Problem 6+7: Memory and speed issues

In the 9th semester project by Matthesen and Schmidt (2014), memory errors were experienced if a surface was fitted to a large point cloud. When running the program at a powerful PC with 16 GB of RAM, the computer ran out of memory if more than 40,000 points were used for the least squares calculation.

It turns out that the memory issues are caused by storing the entire C matrix in memory. In fact, if a surface is fitted to a point cloud with 40,000 points, the C matrix alone takes up 11.92 GBs of RAM. In comparison, the A and b matrix together will occupy only 8.85 MBs of RAM.

There is no need to store the entire C matrix in memory, since only the elements in the diagonal of the C matrix are used. The rest of the elements are zero. Continuing the example with 40,000 points, the diagonal of the C matrix will occupy only 0.31 MB of memory instead of 11.92 GB.

An algorithm has been written, which can find the least squares estimate based on a C matrix defined only by storing the diagonal elements. Since this improved algorithm is light on the memory, it should be able to fit surfaces to much bigger point clouds compared to the previous algorithm. For speed reasons, the improved algorithm also takes advantage of the fact that the N matrix is symmetrical.

The new method should be very fast, since the calculations involving the *many* zeroes in the C matrix are skipped, and since the algorithm takes advantage of the symmetry of the N matrix. Yet, the new method turns out to slower than the previous method. This is because all matrix calculations are performed in pure Python, which makes the algorithm slow. To solve this problem, the algorithm has also been written in Cython which is a programming language used to easily write C extensions for Python. The Cython solution is very fast compared to the other solutions, see Table 7.

No. of points	Old method	New method (pure Python)	New method (Cython)
84,958 points	Out of memory	125.79 s	2.82 s
14,596 points	10.13 s	24,27 s	0.53 s

Table 7 – Time in seconds spend to fit surface before and after optimization

It is recommended that the Cython method should be used in the continued project work, since it is much faster and uses significantly less memory.

A more thorough description of the memory and speed optimizations presented in this section can be found in Appendix C.

2.4 Structure for new algorithm

The purpose of this section is to present the overall structure for a new optimized algorithm for DTM generation. In section 2.3 , a number of problems with the previous algorithm were investigated, and possible solutions were found. These solutions, together with the theory presented in section 0, and experience gained from the previous project by Matthesen and Schmidt (2014), forms a basis for the development of a new and improved algorithm for DTM generation.

The strategy of the new algorithm is to process the point cloud in *two steps*. In the first step, the big clusters of non-terrain points should be removed. The second step should be about removing the remaining non-terrain points. A two-step strategy is chosen since some filtering methods are good at removing the big clusters of non-terrain points, while others are good at removing non-terrain points on small objects. For instance, morphological filters with a small search window are good at removing small non-terrain objects, but not good at removing bigger objects like houses. A big search window makes it possible to remove houses with a morphological filter, but at the risk of removing valid terrain. It is therefore a good idea to remove the big clusters of non-terrain points before applying a morphological filter. With a surface-based filter, the situation is exactly opposite. As experienced in the previous algorithm by Matthesen and Schmidt (2014), a surface-based algorithm is good at removing most big clusters of non-terrain points (especially if it is made intelligent as described in section 2.3.2). However, the small clusters cannot be removed because a buffer is used to select points. The surface-based and the morphological methods therefore seem to complement each other nicely.

A progressive TIN densification algorithm could also be used to generate a DTM, possibly without using two steps. It is however decided not to use progressive TIN densification, since it has a bias towards classifying non-terrain points as terrain (commission error).

Segment-based filtering could also be used for the DTM algorithm. This will however still require two steps. In the first step, the point cloud is divided into segments, and in a second step, the segments representing terrain has to be found. Segment-based filtering cannot handle small clusters of non-terrain points very well. It is therefore possible that a third step is needed in order to get the desired smooth DTM. Furthermore, it is assessed that it is hard to construct

an algorithm using segment-based filtering due to complexity compared to the limited time available for this current project. It is therefore decided not to use segment-based filtering.

2.4.1 Step 1 – Intelligent surface-based filtering

As mentioned, the purpose of step 1 is to remove big clusters of non-terrain points. In order to create a smooth DTM, also the small clusters of non-terrain points have to be removed. Most algorithms good at removing small clusters are however not good at removing big non-terrain objects. Therefore this first step where big clusters are removed is needed in order to get a satisfying result.

It is decided to use surface-based filtering to remove the big clusters of non-terrain points. The reason for this choice is that surface-based filtering in general did a good job removing big non-terrain objects in the previous project by Matthesen and Schmidt (2014). Furthermore, with the suggested changes in section 2.3.6, the algorithm will run very fast, and automatically adapt its parameters to the given terrain. It is expected that an intelligent algorithm which automatically adapts itself to the terrain in general will result in lower orders of polynomials fitted to the points compared to the previous project. This again means that it is easier to remove the problematic low-rise buildings covering a large area. Furthermore, an increased cell-size for the block-minimum filter will also make it easier to remove big clusters of non-terrain points.

The decision to use surface-based filtering means that the algorithm will not be able to handle discontinuities very well. However, a big buffer size could be used, which means that most discontinuities are preserved. A bigger buffer size means that more non-terrain points have to be removed in step 2. This is however not a problem as long as only small clusters of non-terrain points remains.

2.4.2 Step 2 – Slope-based filtering with small search window

The purpose of step 2 is to remove the remaining non-terrain points, which are not removed in step 1. The remaining non-terrain points are expected to be in small clusters. Such clusters can, as described in section 0, be removed with a morphological filter, such as a slope-based filter with a small search window.

It is decided to use slope-based filtering to remove the remaining non-terrain points. The reason is that it is very simple to implement, and should very effectively be able to remove small clusters of non-terrain points. It is possible that the slope-based filter automatically have to adjust according to the slope of the terrain, in order to get a satisfying result in hilly areas.

It should be mentioned that a slope-based filter cannot handle discontinuities in the terrain. It will simply cut off some of the terrain. However, since only a small search window is used, the cut-off problem is limited to a narrow buffer near the discontinuities.

2.5 Conclusion

The purpose of these initial studies was to answer the questions in the initial problem statement. The answers are listed below:

What filtering methods exist, which can handle a photogrammetrically created point cloud?

A lot of different filtering methods exist, but in general they can be grouped into these four groups:

- Morphological and slope-based filtering
- Progressive densification
- Surface-based filtering
- Segment-based filtering

The methods in each group all have different advantages and disadvantages. This means that one of the methods can work in one area, where another method struggles in removing non-terrain points.

How can the problems experienced in the algorithm by Matthesen and Schmidt (2014) be solved, and how can the presented filter theory help solving some of the problems?

The table below shows possible solutions to the problems experienced in the previous algorithm by Matthesen and Schmidt (2014):

Problem	***Solution***
Problem 1: Remains of non-terrain points	A slope-based filtering approach can be used to remove the small non-terrain objects close to the terrain. The previous surface-based algorithm could not remove such non-terrain points because of the used buffer.
Problem 2: Large low-rise buildings	Surface-based filtering with a low order surface polynomial will make it less likely that the surface crawl on top of the buildings. Furthermore, a block-minimum filter with a large grid size should be used as input to the surface-based filter in order to ensure a uniform point distribution and smaller clusters of non-terrain points. In the previous project, a fixed high order surface polynomial was used. If the order of surface polynomial is detected automatically, it is assessed that a lower order polynomial in general will be selected. A more intelligent algorithm which automatically detects the order of polynomial will therefore reduce the problem with low-rise buildings.
Problem 3: Fixed to one terrain type	In the surface-based filter from the previous project, a fixed order of polynomial was used to describe the terrain, and a fixed number of iterations were used to outweigh non-terrain points. If the needed polynomial order and the number of iterations could be detected automatically, the algorithm would automatically adapt itself according to the given terrain. Another solution could be to choose another method such as progressive TIN densification or slope-based filtering which is not fixed to a certain terrain type in the same way as the surface-based filtering.

Problem	***Solution***
Problem 4: Boundaries	In the previous project, the surface-based algorithm worked inside slightly overlapping squares. The squares were overlapping because the iteratively fitted surface will give a bad description of the terrain near the boundaries of each working square. However, the area which gives a bad description is only in the overlapping region if the points are uniformly distributed inside the square. If the points only cover half the square, problematic areas with a bad description of the terrain are not handled correct. A solution could be to use a 1st order polynomial (a tilt-able plane). In order for this to work, constrains should be used, saying that the plane should be levelled with a certain weight. As an alternative solution, any polynomial lower than used in the previous project will help minimizing the problem, since the lower order polynomials are less bendy.
Problem 5: Discontinuities and sharp edges in the terrain	Both progressive TIN densification and segment-based filtering can handle discontinuities. Using one of these two methods could therefore solve the problem.
Problem 6+7: Memory and speed issues	The speed and memory issues experienced in the previous project can be solved by converting the Python code into Cython code and only storing the diagonal of the weight matrix in memory.

Table 8 – Suggested solutions to the problems experienced in the previous project

At a general level, how can the solutions to the problems be combined into an optimized algorithm for DTM generation?

The proposal for a new and optimized algorithm is in this chapter presented as a two-step algorithm. In step 1, clusters of non-terrain points are removed with an intelligent surface-based filter. This is done because most of the filtering principles have troubles removing big clusters of non-terrain points, if the algorithm at the same time should preserve terrain details. Step 2 is about removing the remaining non-terrain points. After step 1 only smaller non-terrain objects exist in the point cloud and those small objects should be removed with a slope-based approach.

It should be mentioned that the proposed algorithm is a compromise, which aim to solve as many of the problems as possible. Especially the problem with discontinuities is not handled very well by the proposed algorithm. One should however keep in mind that large discontinuities rarely occur in Denmark. It is therefore assessed that it may be better to solve some of the problems seen more often, like e.g. large low-rise buildings. If an algorithm handles discontinuities well, it will probably not be able to remove large low-rise buildings, since they look similar to discontinuities.

Chapter 3: Problem Statement

3.1 Problem statement

The initial studies present a proposal for a new and optimized algorithm for filtering of a DSM point cloud, generated from drone data. Now the algorithm should be developed to work on the point cloud and filtering the correct non-terrain points away from the dataset. Step 1 in the suggested algorithm is an improved and intelligent surface-based filter. Step 2 is new compared to the algorithm in Matthesen and Schmidt (2014) with a slope-based approach to remove the small non-terrain objects. This leads to the following problem statement:

How can the surface-based filtering in step 1 be developed to be intelligent and automatically adapt to the terrain?

How can the slope-based approach in step 2 be implemented in a way that removes the non-terrain points and preserve the terrain details?

How can step 1 and 2 be implemented in a user friendly program, which makes it possible to run the algorithm on a photogrammetrically produced point cloud?

How good is the quality of the developed DTM algorithm?

The next four chapters will be about answering the four questions in the problem statement. Each chapter will focus on the answer of one of the four questions. Before the next chapters the overall structure for the algorithm is presented followed by a presentation of the project area and the smaller test areas. The test areas are used when parameters for the algorithm are tested and the total project area is used when the algorithm is implemented in the user friendly program as described in question three.

3.2 Algorithm structure

The idea for a new algorithm has already been described in section 2.4 , but will now be concretized. The new algorithm will consist of 2 steps. Step 1 is an initial selection where big clusters of non-terrain points are removed. Step 2 is the final selection where the rest of the non-terrain points should be removed from the point cloud.

Figure 22 shows a flowchart for the overall structure for the algorithm.

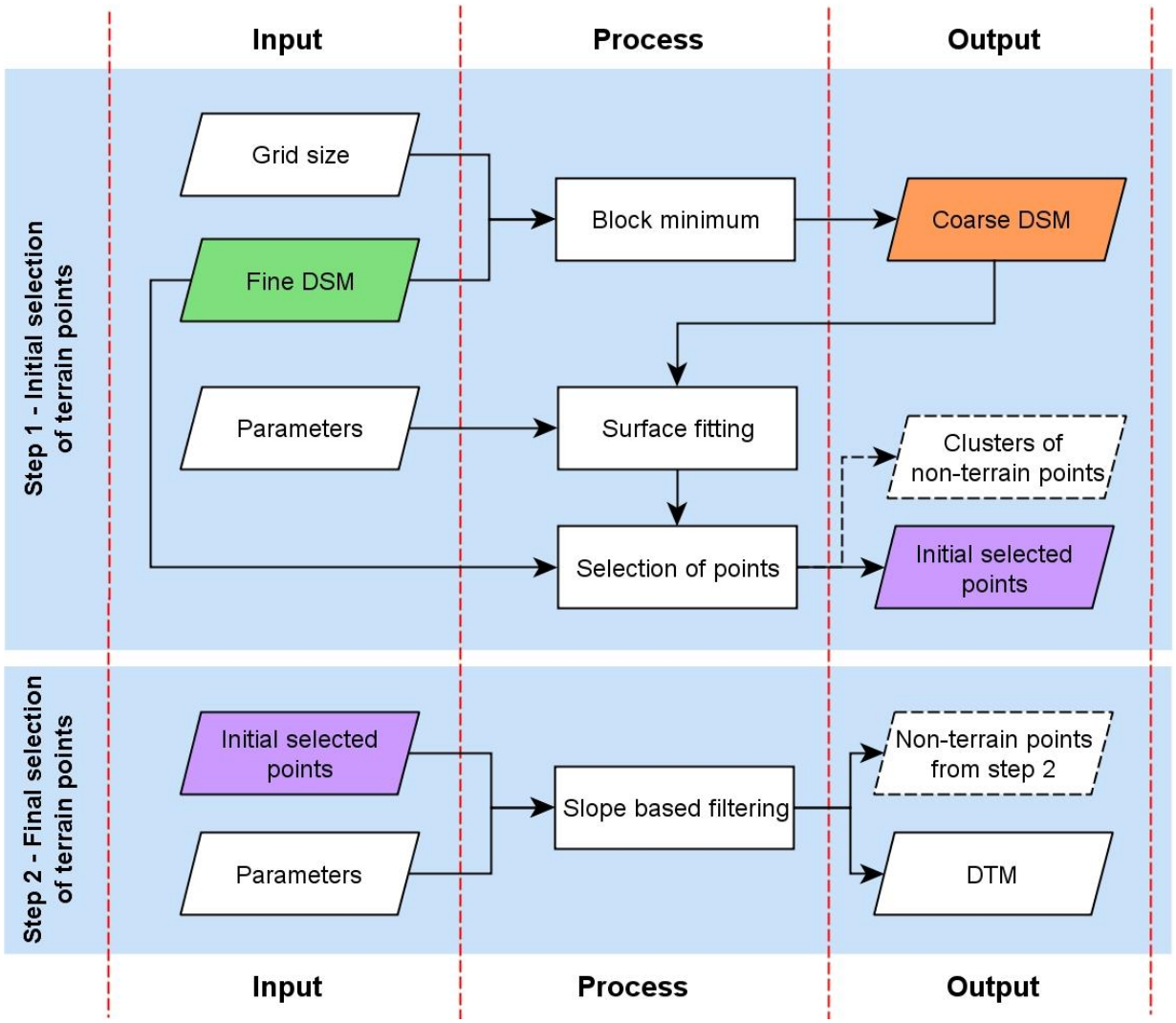


Figure 22 – Overall process for the filtering algorithm

The two steps will be further discussed, developed and tested in the following chapters, where Chapter 4 deals with step 1 and Chapter 5 deals with step 2.

3.3 Test areas

Under the development and test of the two steps, data from a project area is used. The project area is the same as in Matthesen and Schmidt (2014), and is a part of Golfparken in Aalborg, see Figure 23. The DSM can be seen in Figure 24.

Figure 23 – The project area in Golfparken in Aalborg (Matthesen og Schmidt 2014, 12)

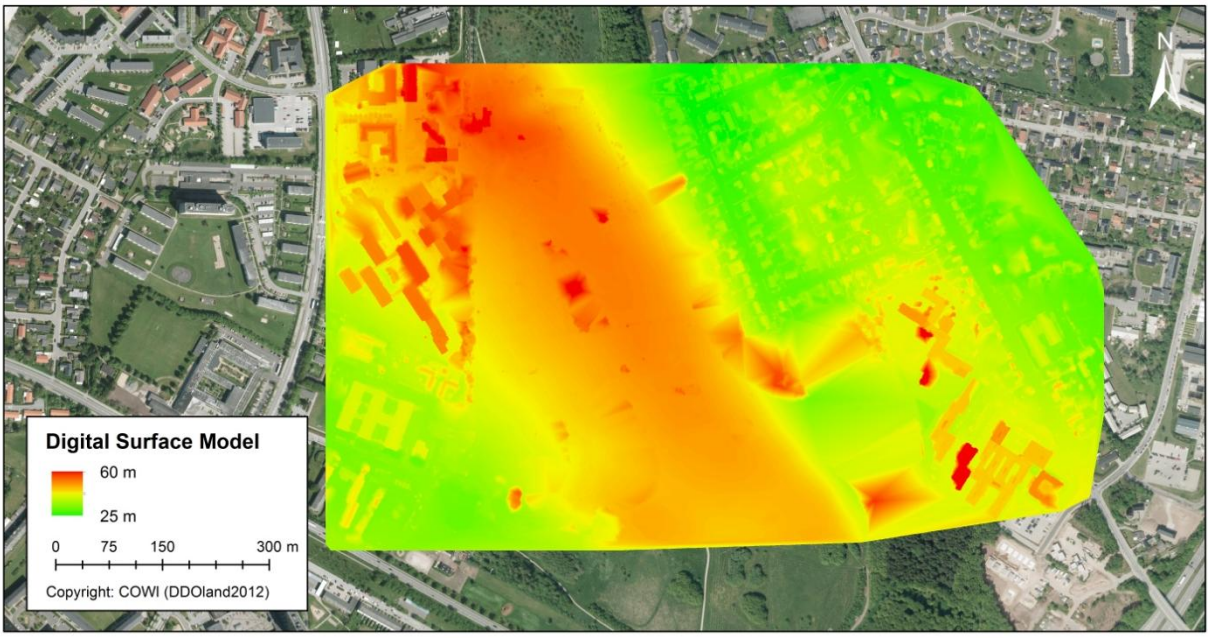


Figure 24 – DTM for project area

The project area is too big to be used for testing, which is why three subareas are selected to be used in tests, see Figure 25.

Figure 25 – The subareas used for testing

The subareas are chosen with different terrain characteristics. Subarea A contains big buildings, subarea B contains almost only terrain points and subarea C is an area with family housing. A more detailed description of the subareas can be seen in Appendix A.

Chapter 4:
Step 1

4.1 Introduction

Step 1 of the filtering algorithm is about removing the big clusters of non-terrain points. The method builds on top of the algorithm from the previous project by Matthesen and Schmidt (2014). A surface-based approach is used, but the algorithm should be developed to be intelligent and automatically find the best suited polynomial and the number of iterations.

Step 1 can be divided in two sub-steps. First a block-minimum filter is applied to the fine DSM. This is done to create a coarse DSM where the point distribution is more even than in the fine DSM. Furthermore the coarse DSM will include fewer non-terrain points, which can make it faster to fit a polynomial to the terrain. The block-minimum filtering will in the final program be referred to as step 1.1.

The other sub-step is called step 1.2 and includes the actual surface filtering. The surface-based filtering works inside smaller overlapping squares as illustrated on Figure 26. The size of the squares and the overlap is the same as in the previous project. Each square is 100x100 m and the overlap is 30 m. As mentioned the method in step 1 is similar to the method used in the previous project by Matthesen and Schmidt (2014), where a specific polynomial order was fitted to a coarse DSM. When the surface was fitted the residuals were calculated and new weights for all points were found using a weight function. This process was repeated until a specific iteration was reached, and the surface was expected to represent the terrain. When the final surface was fitted, points inside a buffer on each side of the surface were selected as initial terrain points. This is just a brief explanation of the surface-based filtering, but a more detailed description can be found in Appendix B.

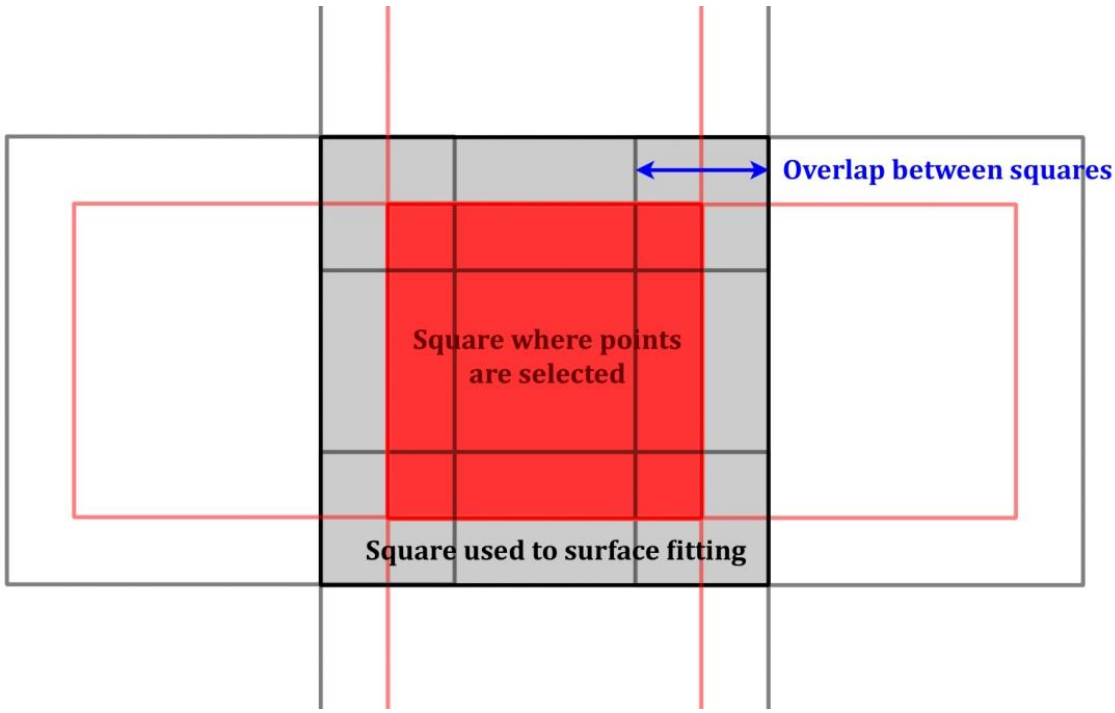


Figure 26 – The surface-based filtering works inside smaller overlapping squares. (Matthesen og Schmidt 2014, 105)

In this present project the surface based filtering should be intelligent. This means that the algorithm automatically should detect the order of the surface polynomial which should be iteratively fitted to the terrain inside each square. Furthermore, the number of iterations needed to iteratively outweigh the non-terrain points should be detected automatically.

In order to automatically detect order and iterations, some tests must be performed to find the best stop criterions to use. Even though the focus is on the automatic detection of order and iteration, the weight function should be investigated a little again. The weight function used to iteratively outweigh the non-terrain points in the project by Matthesen and Schmidt (2014) worked well with the specific parameters used in that project. However, the weight function has to be tested again in order to ensure that the weight function works in the new intelligent surface filtering. Also the buffer used when selecting points to has to be tested with the new parameters.

The flowchart in Figure 27 shows the overall structure of the algorithm in step 1.

The elements of the flowchart are explained in detail in the later subsections of this chapter. First the block minimum filter will be described, before the method for the intelligent surface-based filtering is explained. Afterwards there is an initial testing of the method before the real tests are made.

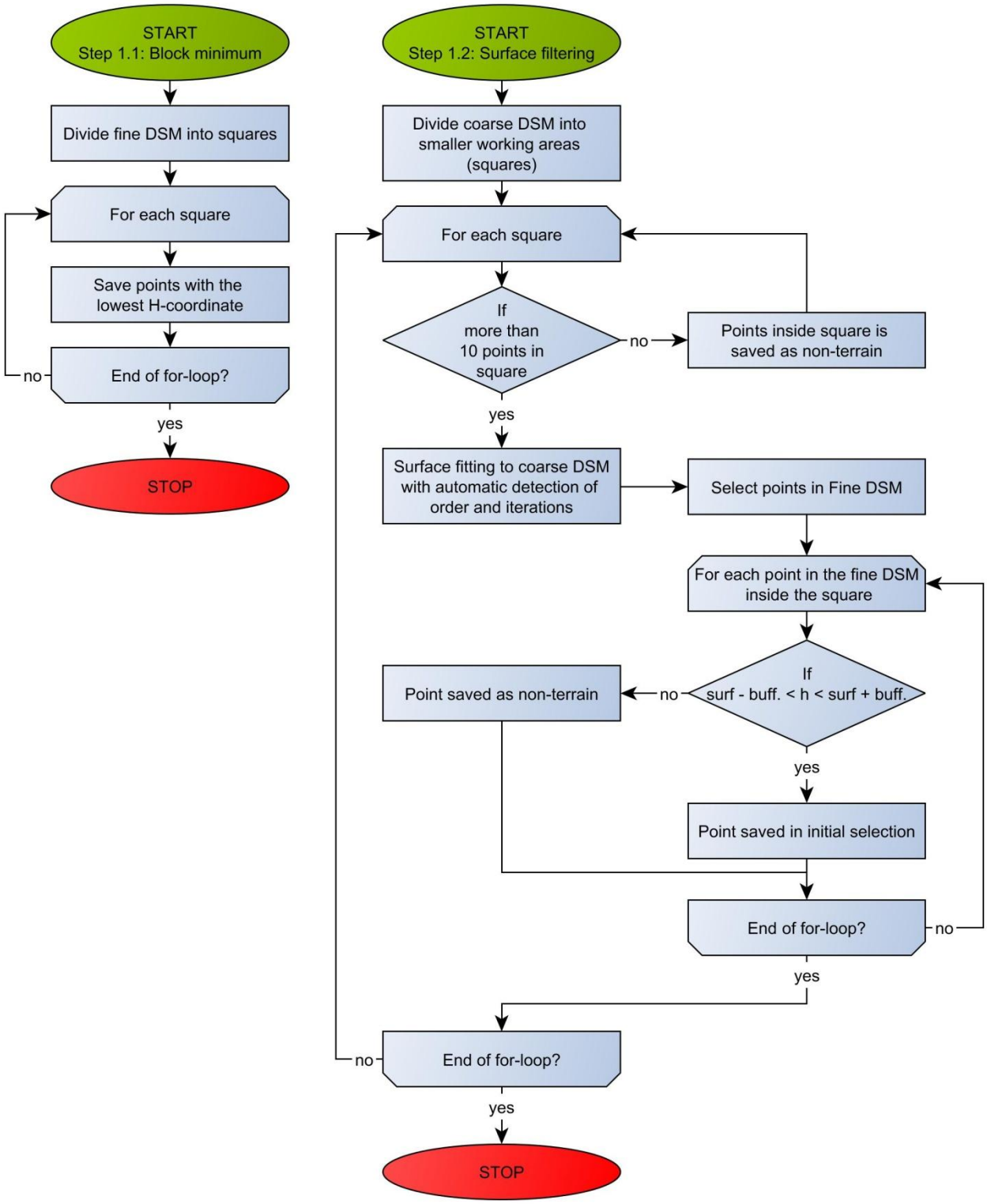


Figure 27 – Flowchart for the overall structure in step 1

4.2 Step 1.1: Block-minimum filter

In the previous project by Matthesen and Schmidt (2014), a block-minimum filter with a grid size of 5 meters was used. The block minimum filter makes it easier to filter away big clusters of non-terrain points, since fewer points are placed on e.g. the rooftops after applying the filter. Furthermore, the block-minimum filter ensures a more even distribution of the points, which will give more reliable results, since the point density is less likely to vary depending on textures and materials.

Experience from the previous project showed that it was problematic to remove large low-rise buildings. By using a bigger cell size than the 5 meters previously used for the block-minimum filter, it is easier to remove large clusters of non-terrain points. However, the cell size should not be too big, because of a risk to remove details in the terrain. It is decided that a cell size of *10 meters* should be used in this current project, since it is assessed that most terrain details are preserved, and at the same time it is easier to remove large clusters, compared to the previous project. Below is an example of the project area after a block minimum filter is applied with a grid size of 10 meters.

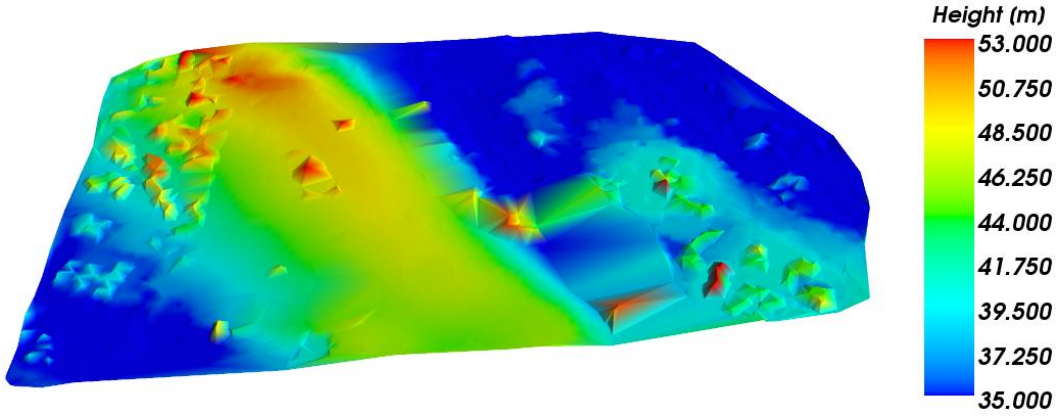


Figure 28 - Block-minimum result with a used grid size of 10 meters

The block-minimum filter in the previous algorithm was implemented in Python. As a result, it took quite a long time to run the block-minimum algorithm for the entire project area. To speed things up in this current project, the block-minimum filter is implemented in Cython instead of Python. By doing this, the block-minimum filter can run approx. 7 times faster, see Table 9.

Grid size	Python method	Cython method
5 m	1.284,65 s	184,78 s
10 m	367,64 s	54,10 s

Table 9 – Time in seconds spend to run block-minimum filter

For each subarea, Figure 29 shows the original point cloud and the point cloud after the block minimum filter is applied. Subarea A now contains a few non-terrain points on top of the buildings. In subarea B no non-terrain points exist in the block minimum data, and in subarea C it is almost the same situation. There may be a few points which are non-terrain, but it is hard to decide because of the complexity of the area.

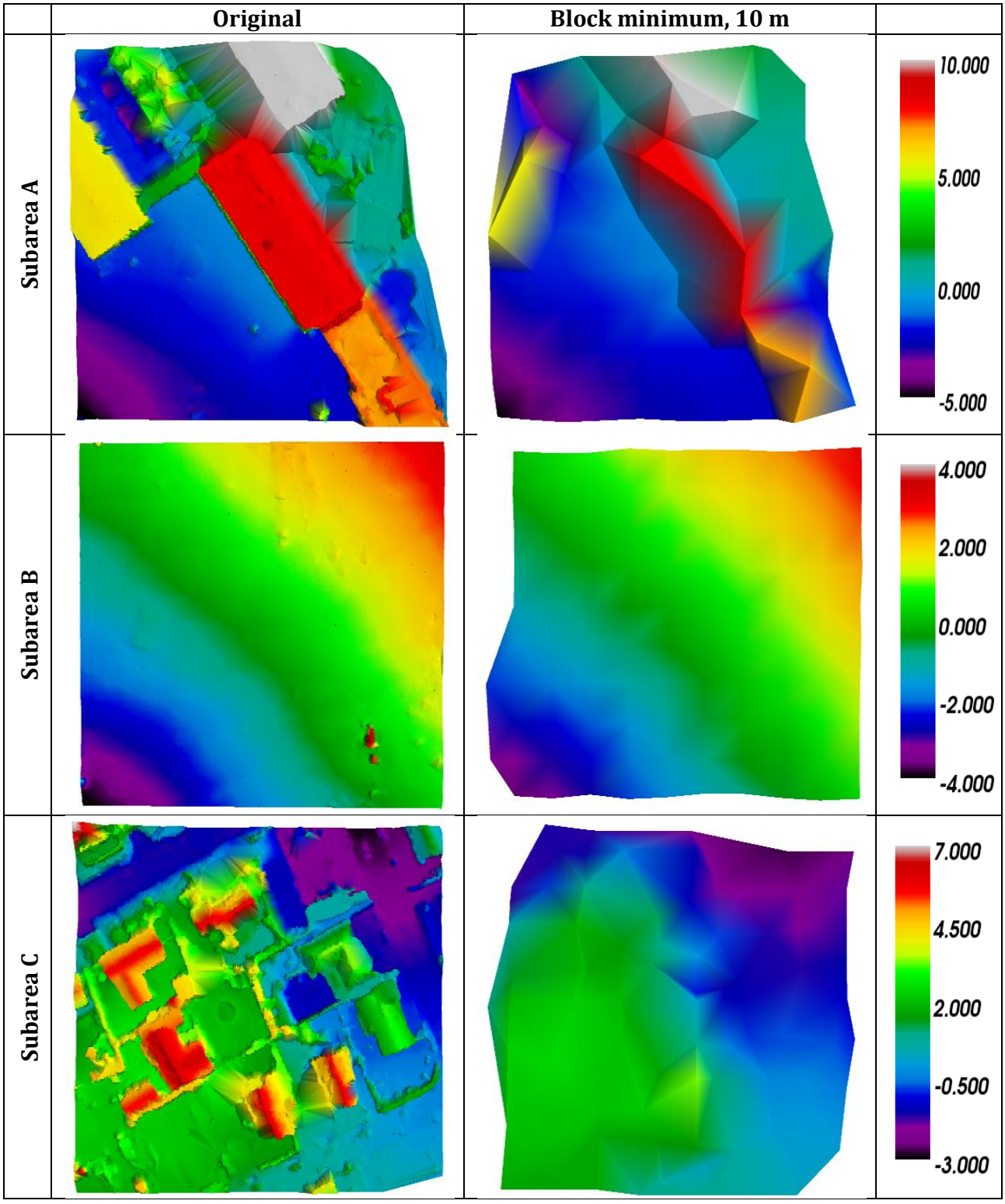


Figure 29 – Point cloud before and after applying the block-minimum filter

4.3 Step 1.2: Intelligent surface-based filtering

The idea for making the surface-based filtering intelligent is to use the standard deviation of unit weight. In the project by Matthesen and Schmidt (2014) it was found that the polynomial order which gives the best description of the terrain in a given part of the point cloud could be found by looking at the standard deviation of unit weight. When choosing a higher polynomial order does not lead to significant change of the standard deviation of unit weight, the best polynomial is found.

A similar method can be used to detect the number of iterations needed to fit a given polynomial to the terrain. When the standard deviation of unit weight does not change significantly between two iterations, it makes no point to do any extra iterations. Figure 30 shows a flow diagram for the automatic order and iteration detection.

In the previous project by Matthesen and Schmidt (2014), a fixed polynomial order and number of iterations were used for the entire project area. The fixed values for the polynomial order and the number of iterations were found by a visual look on plots of the standard deviation. With the human eye, it is easy to assess when the standard deviation has reached a steady state. However, in this project the assessment should be automated, in order to automatically find the optimal parameters according to the characteristics of the terrain in a given part of the project area.

Since the algorithm cannot visually see when a steady state has reached, a stop criterion should be used to assess when the standard deviation stops changing significantly. In this project it is decided to look at the change in percentage between two following standard deviations. When the change in percentage is below a given stop criterion, a steady state has been reached.

In order to prevent that the algorithm can go on forever with an endless loop of iterations or orders, a maximal order and a maximal allowed number of iterations are added to the algorithm.

The rest of this chapter will be about the development of the surface-based filter for step 1. The most important part of this development is to determine the parameters (stop criterions, weight function, buffer size etc.), which should be used for the filter. Various tests will be performed in the following sections in order to determine the best set of parameters.

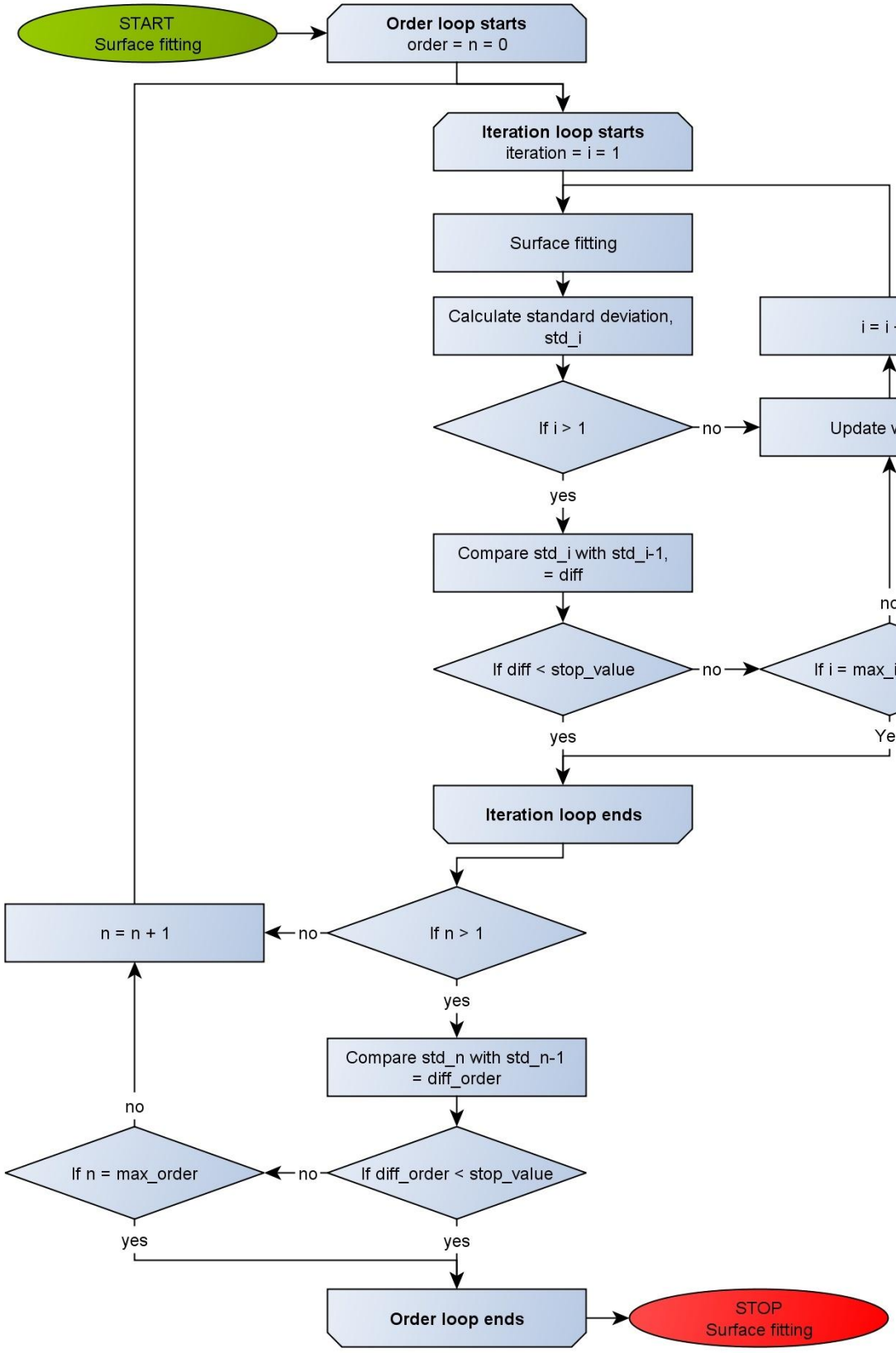


Figure 30 – Flow diagram for the automatic order and iteration detection

4.4 Initial testing

This section is about testing if the parameters for the intelligent surface-based filter can be found based on data already available from the previous project by Matthesen and Schmidt (2014). It should be noted that the used data for the surface fitting in this present project is different from the data in the earlier project because another grid size is used for the block-minimum.

Even if this initial test shows that further investigations and tests are needed, it is assessed that the test at least will give an indication if the standard deviation can be used for the automatic detection of order and number of iterations, as suggested in the previous project.

4.4.1 Stop criterion for polynomial order

When the polynomial order was chosen in the project by Matthesen and Schmidt (2014) it was from a visual look at the plot in Figure 31. For different terrain types, the plot shows the standard deviation of unit weight as a function of the polynomial order used. When the standard deviation between to orders did not change significantly the lowest order was chosen. This is a little different from how the stop criterion is implemented in this current project. In the algorithm for this project, the standard deviations between two loops are compared and if the difference is smaller than the stop criterion, then the loop is stopped and the last order is saved.

The percentage change between two following orders is calculated this way:

$$\%\ \text{change} = \frac{\text{std}_{n-1} - \text{std}_n}{\text{std}_{n-1}} \cdot 100 \tag{4.1}$$

where std_n *is the standard deviation for order n*
std_{n-1} *is the standard deviation for order n-1*

In order to find the initial stop criterion, the data from the previous project is investigated, see Figure 31. The figure shows the standard deviation when polynomials of different orders are fitted with one iteration to a point cloud containing only terrain points. The selected order where the standard deviation of unit weight has reached a steady state is used as starting point. This order is called the n'th order. Because of the differences in how the order is chosen in this project compared to the previous project, the percentage difference in standard deviation is both investigated between order n-1 and n, as well as between order n and n+1. The percentage differences are presented in Table 10.

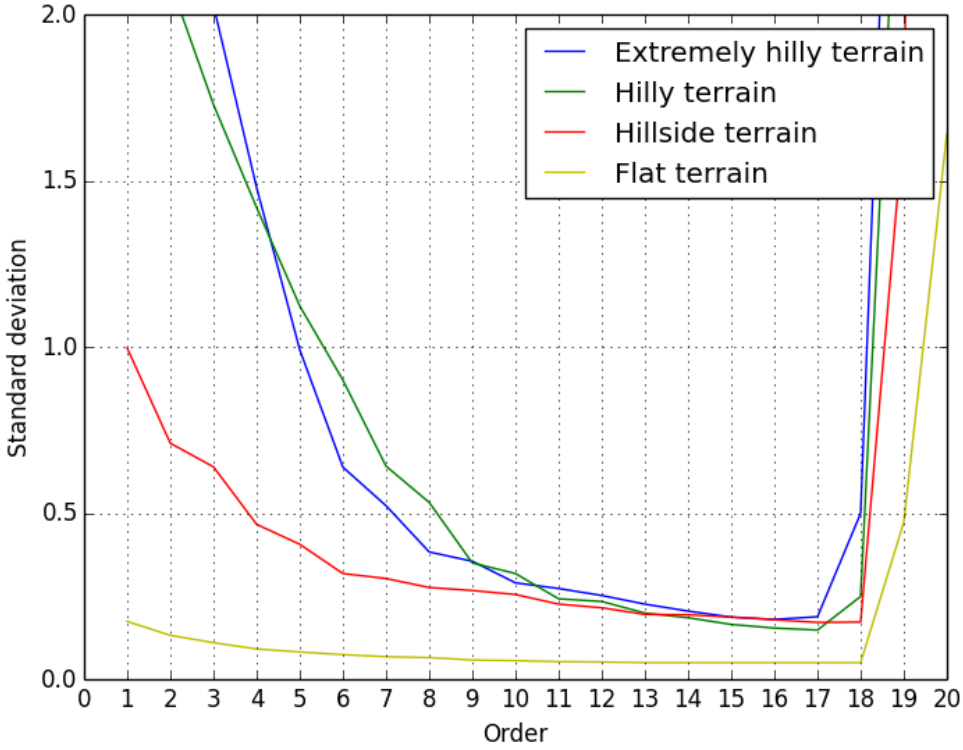


Figure 31 - The standard deviation of unit weight at different polynomial orders and with different terrain types(Matthesen og Schmidt 2014, 91)

Terrain type	**Order (n)**	**Difference in standard deviation between order n-1 and n (%)**	**Difference in standard deviation between order n and n+1 (%)**
Flat terrain	4	17.1	9.8
Hillside terrain	6	21.6	4.7
Hilly terrain	11	23.8	3.3
Extreme hilly terrain	8	26.6	7.3
Average		**22.3**	**6.3**

Table 10 – Percentage differences between standard deviations at the order where it is visually assessed that a steady state has been reached

If the algorithm should end up with the same polynomial order as shown in the table, the stop criterion should be quite high, about 22%. Such a high stop criterion could lead to a used polynomial which has a too low order, since a change of 22% in standard deviation easily can happen before a steady state has been reached. If the percentages change between order n and n+1 (about 6%) is used as stop criterion, it is more likely that the algorithm will find the best fitting polynomial order. In contrast to the previous project, the chosen order will be one order higher. If the same order should be chosen a solution could be to save the polynomial coefficients before a new order is calculated. This solution is not used since it is assessed that the higher order will only give a more detailed description of the terrain and the algorithm would be simpler without. One should also keep in mind that the automated method in general will end up using a lower polynomial order than the fixed order used in the previous project.

It is assessed that a stop criterion of 6 % maybe is a little too low, because half of the values used to calculate the average of 6 % is bigger and one is actually very close to 10 %. Therefore a stop criterion of 10 % is used as initial value. Furthermore any value below 17 % will end up with the results where n is compared to n+1 in Table 10.

4.4.2 Stop criterion for the number of iterations

The number of iterations used in the project by Matthesen and Schmidt (2014) was found by visual looks at plots of the surfaces and plots of the standard deviation for each iteration. As when choosing the order, the number of iterations was chosen as the lowest iteration when a steady state was reached. In this project the algorithm must find the order by itself by looking at percentage changes between the standard deviations from two following iterations. The percentage change is calculated this way:

$$\%\ \text{change} = \frac{\text{std}_{i-1} - \text{std}_i}{\text{std}_{i-1}} \cdot 100 \qquad (4.2)$$

where std_i *is the standard deviation for the i'th iteration*
std_{i-1} *is the standard deviation for i-1'th iteration*

The chosen iteration number from the previous project is used in the same way as when the initial order stop criterion was found above. The percentage change is both calculated between iteration i-1 and i, as well as between iteration i and i+1. The results can be seen in Table 11.

In the previous project by Matthesen and Schmidt (2014) three different weight functions were used when the number of iterations was investigated. The notations *"low"*, *"med"* and *"high"* in Table 11 refers to the used weight function with either low, medium or high aggressiveness.

Subarea	**Iteration (i)**			**Difference in standard deviation between iteration i-1 and i (%)**			**Difference in standard deviation between iteration i and i+1 (%)**		
	low	*med*	*high*	*low*	*med*	*high*	*low*	*med*	*High*
A	9	6	6	10.7	3.5	2.1	-1.0	0.0	0.7
B	2	2	2	0.0	4.3	19.6	0.0	0.0	0.0
C	4	6	6	3.1	1.4	1.1	0.3	0.5	0.0
Average				**5.1**			**0.05**		

Table 11 - Percentage differences between standard deviation for two following iterations

From the table it can be seen that the difference between the standard deviation for iteration i and the standard deviation from the previous iteration (i-1) in average is 5.1%. When an iteration is chosen automatically in the new algorithm the standard deviation of one iteration is compared to the standard deviation of the previous iteration. If this percentage change is under the stop criterion the last iteration is chosen. A percentage difference of 5.1% seems acceptable. Therefore the initial stop criterion for the iterations is chosen as 6%. It is decided to round up the 5.1% to 6% because this will lead to fewer iterations which will speed up the algorithm.

Furthermore the fitted polynomial order does not need to totally fit the terrain, because a big buffer is used when the terrain points are selected.

4.4.3 Weight function

The chosen weight function in Matthesen and Schmidt(2014) can be described in this way:

$$weight_i = \begin{cases} 1 & r_i \leq a \\ 0.5 \cdot \cos\big((r_i - a) \cdot b\big) + 0.5 & a < r_i \leq \frac{\pi}{b} + a \\ 0 & \frac{\pi}{b} + a < r_i \end{cases} \quad (4.3)$$

where r_i *is the residual for the i'th point*
a *is the shift value*
b *is the steepness*

The two parameters 'a' and 'b' are adjustable and can be used to make the weight function more or less aggressive. 'a' is the shift value used to move the function along the x-axis and 'b' controls the steepness.

In Matthesen and Schmidt (2014) a weight function with medium aggressiveness was used, see Figure 32. This function worked well and therefore it is decided to use it as the weight function for the initial test.

Medium aggressiveness (a = -0.3 , b = 2**)**

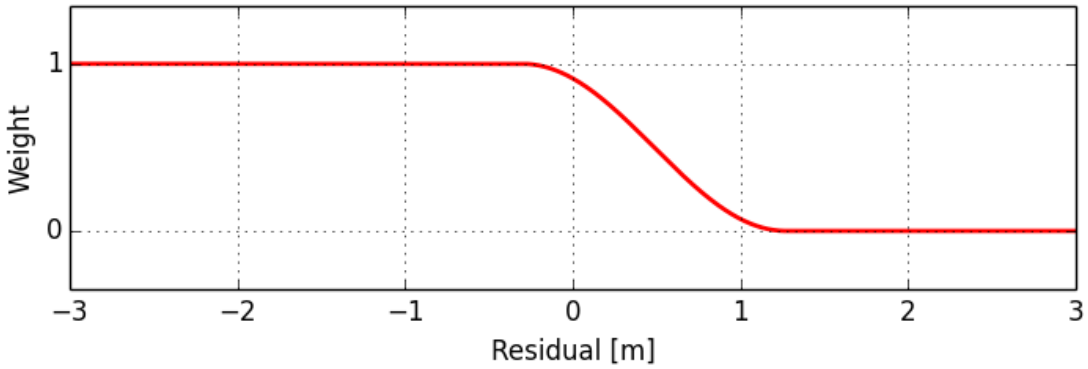


Figure 32 - Weight function with medium aggressiveness

4.4.4 Initial test of the algorithm

To see how the algorithm works with the automatic detection of order and number of iterations the algorithm is tested with the initial values found in the previous sections. The final order and number of iterations detected for each subarea is:

Subarea	Final polynomial order	Iterations
A	2	6
B	3	2
C	2	7

Table 12 – Final order and iteration in the initial trial of the algorithm

The automatically detected polynomial order does not look as expected. Subarea B is relatively simple compared to subarea A and C and it was therefore expected that the order for subarea B should be lower than for the other subareas. The order for subarea A and C are in general quite low, and significantly lower than expected. In order to investigate why these results appear, the standard deviation for every order in the subareas are plotted when the ideal iteration are reached according to the iteration stop criterion. See plot in Figure 33. For subarea B it nearly looks as expected with the standard deviation decreasing with a higher order. For subarea A and C it is another story. Here the standard deviation bounces up and down, and therefore it is not possible to choose the best polynomial from a change in the standard deviation.

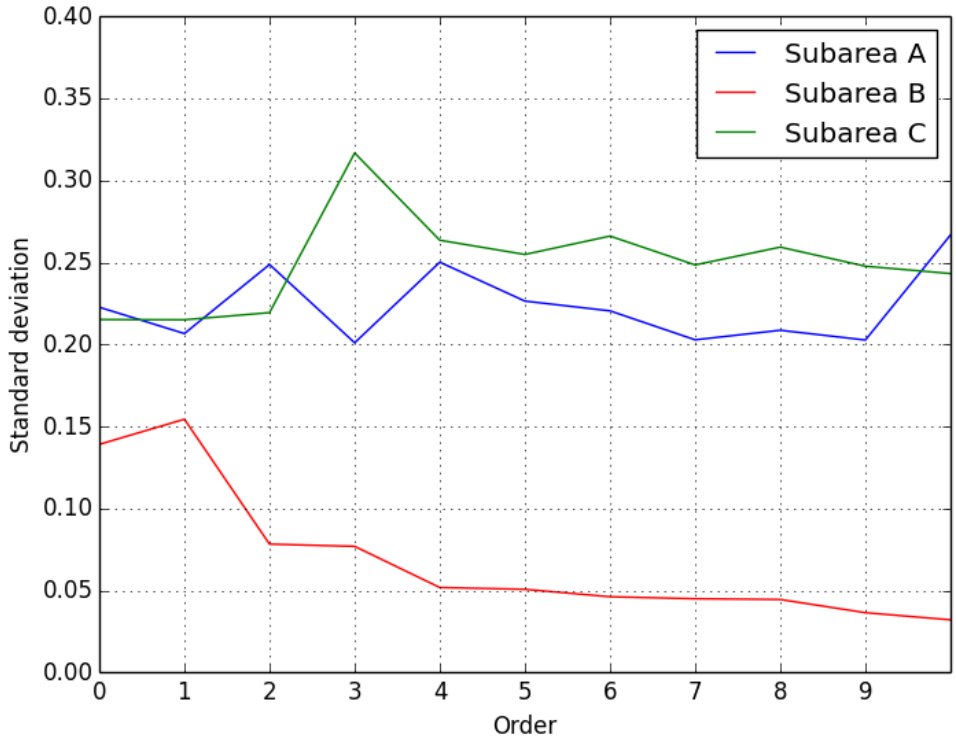


Figure 33 – Standard deviation for each order in the subareas when the optimal iteration is reached.

A reason for this behavior of the standard deviation could be the weight function. The shift value 'a' is negative which is really good to ensure that the surface always goes downwards for each iteration. With a negative shift value, a point which is directly on the surface is weighted

down and the surface will therefore continue downwards. This is especially a problem when the polynomial order is low as e.g. a plane. The plane only require 3 points to be defined and because of that, it can in principle move itself downwards until only three points are used to define the plane, see Figure 34.

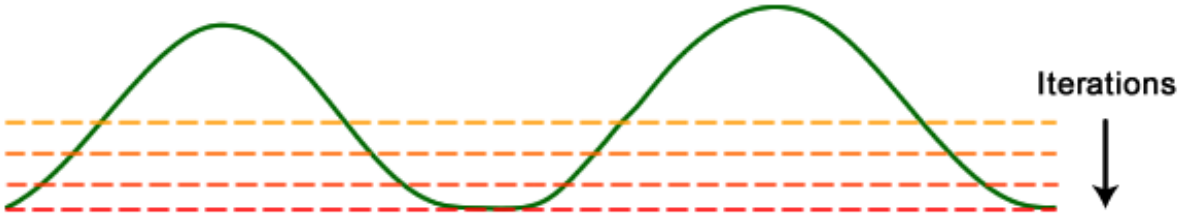


Figure 34 – A plane will just continue to move downwards with the initial weight function

Another reason why it is a problem that the surface always moves downwards is that the use of a 10 m block minimum before the surface fitting means that for some areas there is no non-terrain points left in the block minimum dataset, see section 0. Therefore the selected polynomial should just describe the terrain and not necessarily move downwards to remove non-terrain points.

It should be noted that a polynomial which perfectly describes the terrain cannot continue to move downwards no matter how negative the shift value 'a' is. The reason is that all points in this ideal example will have the exact same (low) weight, since they all have a residual which is zero. In other words, the problem only occurs if a low order polynomial is used, which cannot perfectly describe the terrain.

The reason why the weight function worked in the previous project by Matthesen and Schmidt (2014) is that a high order was already decided to be used and there were more non-terrain points which should be removed because of the smaller block minimum grid size.

This initial testing reveals some problems which should be further investigated. The biggest problem is assessed to be the weight function which is clearly not the best suited for this surface fitting. Because of the importance of the function and the direct influence on the number of iterations the weight function is one of the first things to be tested and clarified. Furthermore the stop criterions for both order and iterations should be tested.

4.5 Tests

After the initial testing some problems were found. Especially problems with the weight function. Because the weight function has an influence on the number of iterations needed to fit the surface polynomial to the terrain, the optimal weight function should be found before the stop criterion for the iterations can be determined.

After the weight function is found, the stop criterions for the iterations and the order of polynomial has to be found. The algorithm iterates through the different orders and for every order a number of iterations are made to iteratively outweigh non-terrain points. The final

standard deviation for a specific polynomial order depends on the number of iterations made. If the surface is not fitted to the terrain, because not enough iterations have been made to outweigh the non-terrain points, the std. dev. of unit weight might not decrease as expected. Therefore the parameters regarding the iterations should be determined before the parameters regarding the polynomial order. When both the weight function, parameters for iterations and parameters for polynomial order is found, the surface fitting can be done. The last step is to select the initial terrain points with a buffer on both sides of the fitted polynomial.

This section will include the following tests:

- **Weight function:** What parameters should be used for the weight function?
- **Iterations:** What is the maximal allowed iteration and what should the stop criterion be?
- **Polynomial order:** What is the maximal allowed order and what should the stop criterion be?
- **Buffer:** What should the buffer be in order to remove the big clusters of non-terrain points?

4.5.1 Weight function

The weight function should ensure that the surface polynomial moves downwards in places where there are non-terrain points.

The weight function used for the initial testing had a negative shift value. This is problematic since valid terrain might end up being outweighed if a low order polynomial is used, which does not perfectly describe the terrain. Instead, a new weight function should be designed, which gives points on the surface or a little above the full weight. In other words, new parameters for the steepness and the shift value of the weight function should be found.

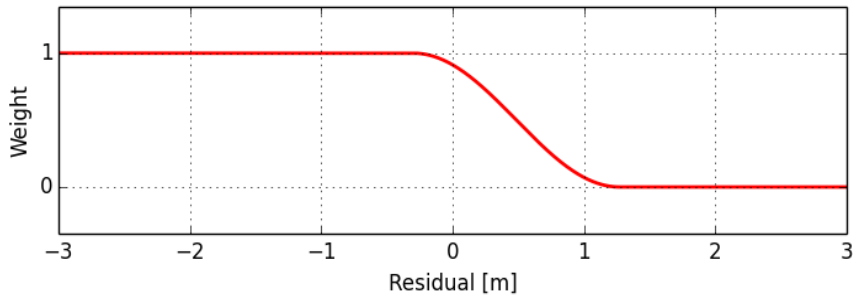
The precision of the DTM created in Matthesen and Schmidt (2014) was on well defined places better than 10 cm. Three times the precision seems as a good starting point for a new shift value. The steepness is chosen in a way that will outweigh non-terrain points. A building is normally more than 2 m high and therefore the weight function should assign a weight of zero to points more than 2 m above the fitted surface. Different values of steepness and shift value has been tested, but if the weight function fulfills the above mentioned conditions then the exact values are not that important. Based on the arguments presented, it is decided that the shift value in a new weight function should be 0.3 and the steepness should be set to 1.7. Figure 35 shows the weight function for the initial testing together with the new weight function.

It should be mentioned that a dynamic weight function also could be a solution. This is however not investigated in this project due to limited time. The main focus is to make the algorithm intelligent by automatically finding the best polynomial order and the number of iterations needed.

The surface fitting is tested again with the new weight function to see if the results are better than in the initial testing. The test is done in the three subareas and the standard deviation for each order is plotted in Figure 36. As it can be seen the standard deviation behaves more like

expected, since it gradually decreases as the order increases. There is still some bouncing up and down which mean that the order test should take into account what to do if the standard deviation gets bigger from one order to another.

Weight function for initial testing (a = -0.3 , b = 2**)**



New weight function (a = 0.3 , b = 1.7**)**

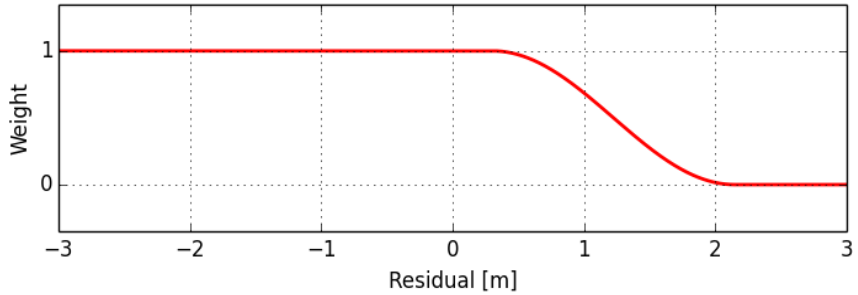


Figure 35 – Initial and new weight function

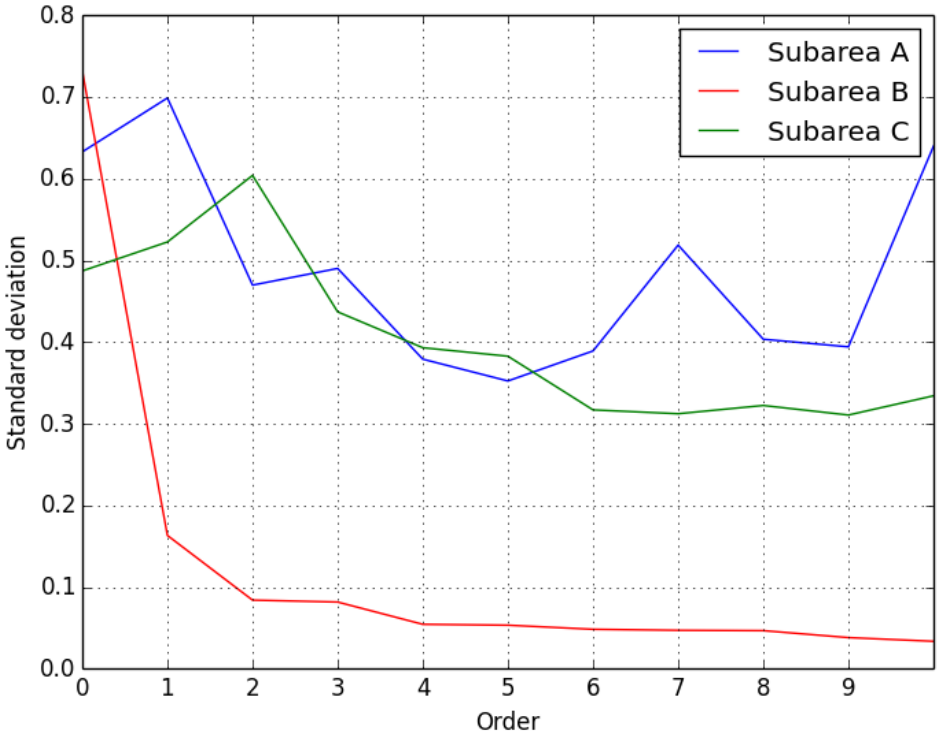


Figure 36 - Standard deviation for each order in the subareas (the number of iterations for each order is decided according to the stop criterion, of 6%, used in the initial testing)

4.5.2 Iterations

The tests about iterations deals with the maximum allowed number of iterations and what the stop criterion should be when a polynomial with a given order is fitted to the point cloud. The maximum number of iterations is used if a polynomial for a given order fits so badly to the points that the standard deviation never reaches a steady state. The stop criterion is used to decide if a new iteration should be made or if the fitted polynomial should be used.

4.5.2.1 Maximum allowed number of iterations

The maximum number of iterations is found through a simple test. In the test, the new parameters found for the weight function in the previous section (4.5.1) are used. For each subarea, surface polynomials of order 0 to10 are iteratively fitted to the point cloud. The number of iterations to use is automatically found by setting the stop criterion to 1 %. This choice is made since it in practice makes no sense to do any more iterations if the change of std. dev. between two consecutive iterations is below 1%. The number of iterations used before the stop criterion was reached is, for every order and for every subarea, shown in Figure 37.

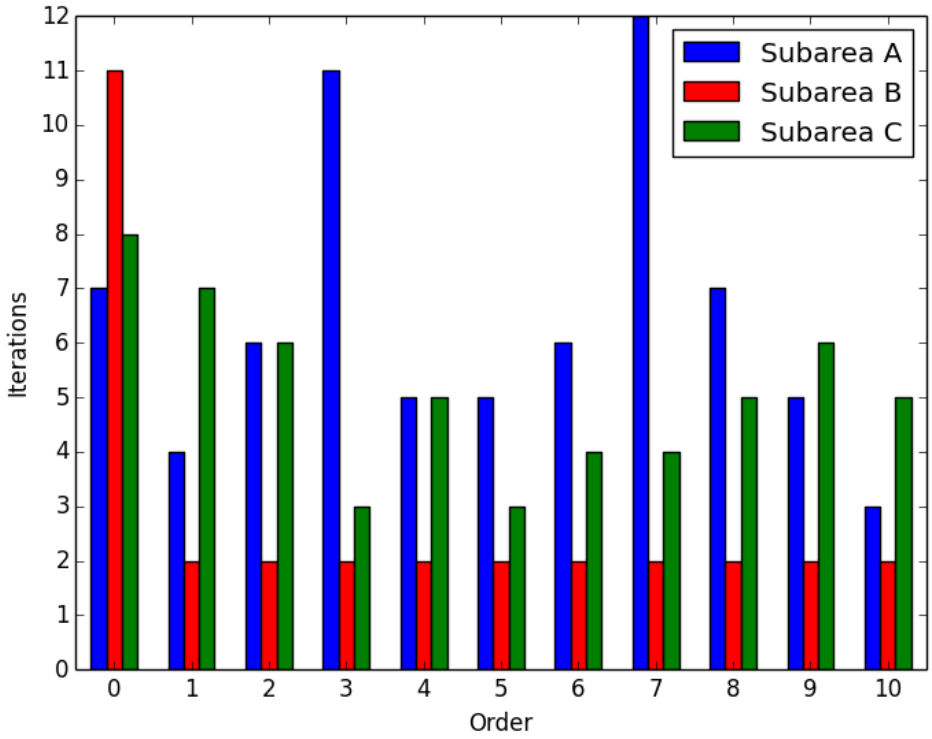


Figure 37 – Maximum iterations used to fit different orders of polynomials to the subareas when the iteration stop criterion is 1%

The main purpose of the maximum allowed number of iterations is to prevent an endless loop of iterations if the standard deviation of unit weight for some reason never reaches a steady state. This means that the maximum number of allowed iterations should be equal to the number of iterations which are experienced in the worst case scenario. From the diagram at Figure 37, it can be seen that in the worst case scenario, 12 iterations are needed in order to reach the stop criterion. Therefore, it is decided that *12 iterations* should be the maximum

allowed number of iterations. It should however be noted that Figure 37 also shows that the algorithm in most cases stops after fewer than 6 iterations.

4.5.2.2 Iteration stop criterion

To find the optimal stop criterion for the number of iterations, a test has been made. Again, the new parameters for the weight function are used in the test. In the test, a polynomial with order 0 to 10 is fitted to each subarea. For each polynomial, 12 iterations are made and the standard deviation is saved. Based on the test, plots of the standard deviation as a function of the number of iterations can be made for each subarea. Figure 38 shows the standard deviation for subarea A, which is the most interesting area in this test, because it contains most non-terrain points. Plots for subarea B and C can be found in Appendix D.

A visual look at the plots can tell where the standard deviation stops changing significantly. For each order, the iteration where no visual changes were found is also listed in Table 13. The change in percent between the two last iterations is also calculated in the table.

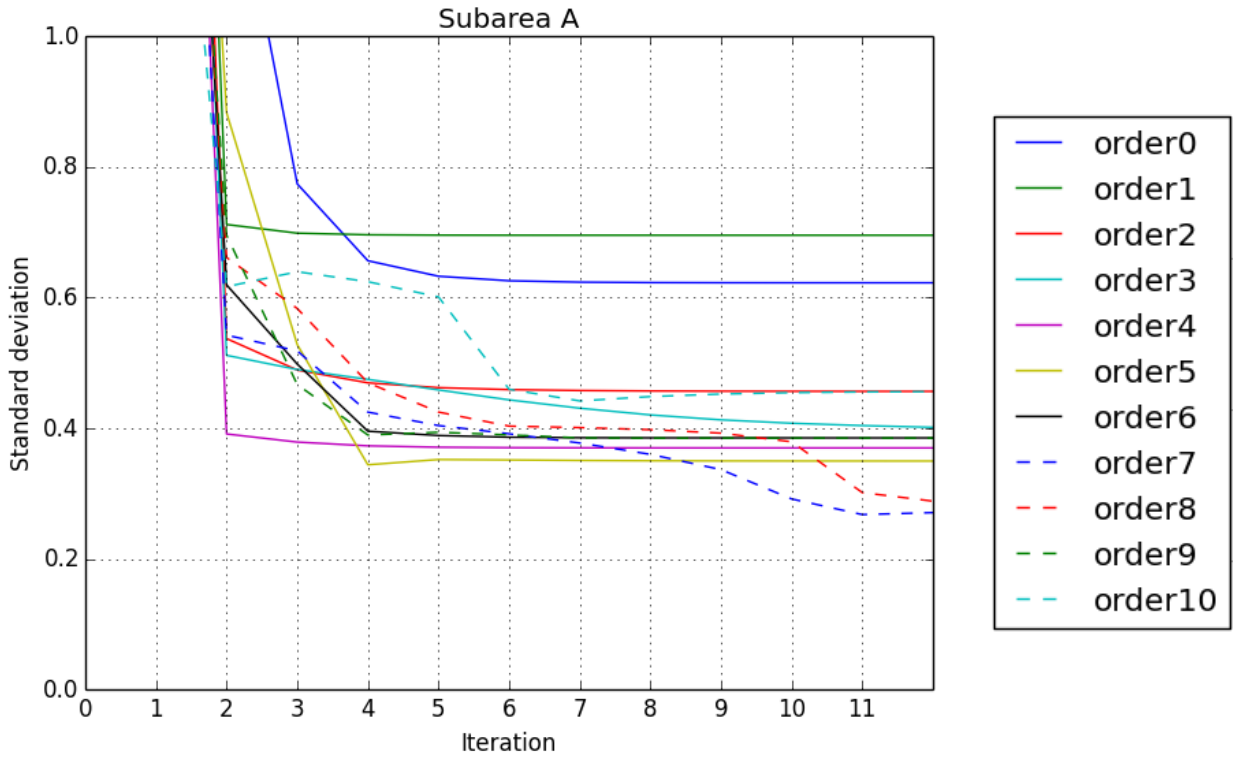


Figure 38 – Standard deviation for each iteration in subarea A. Plots for subarea B and C are in Appendix D

	Iteration (i)			Difference in standard deviation between iteration i-1 and i (%)		
Subarea / Order	*A*	*B*	*C*	*A*	*B*	*C*
0	5	8	6	3.6	2.2	3.6
1	3	2	5	1.9	0	2.6
2	4	2	4	4.0	0	2.2
3	6	2	3	3.3	0	0.6
4	3	2	3	3.2	0	4.4
5	5	2	3	(-2.3)	0	0.6
6	5	2	3	1.7	0	3.3
7	5	2	3	4.9	0	5.2
8	6	2	4	5.1	0	2.2
9	5	2	4	(-1.1)	0	3.5
10	4	2	3	2.4	0	2.0
Average of positive differences				**3.3**	**-**	**2.8**
Average for subarea A and C				**3.0**		

Table 13 – Iteration where the standard deviation stops changing significantly and the percentage change

The results from Table 13 can be used to find the iteration stop criterion. Results from subarea B will not be used since the percentage difference is 0 for all orders but one. For subarea A and C the average is 3 %. It is however not assessed that this is a good stop criterion. The stop criterion should be a little bigger because many of the percentage changes are more than 3 %. A stop criterion of 5 % will in two cases result in extra iterations compared to the table above. It will also mean that maybe some of the orders will stop at an iteration previous the one listed here. It is assessed that this is okay, considering the fact that the surface does not have to fit 100 % to the terrain since it is planned to use a quite large buffer for the classification. Another advantage is that fewer iterations will speed up the algorithm. Based on the considerations presented, it is decided to use a stop criterion of 5 %.

As it can be seen in Figure 38 and Table 13, sometimes the standard deviation grows from one iteration to the next. This leads to a negative percentage change. Sometimes the increase in standard deviation can be okay, because when a steady state is reached some random noise is expected which lead to slightly different standard deviations. When the standard deviation increases a lot between two iterations it might be because the given order of polynomial cannot describe the given terrain properly. The solution is that if the percentage change is a little negative the algorithm continues as always, but if it gets too negative the algorithm should continue to the next iteration.

The question is now how negative the percentage change can be before it is not accepted. With the 10th polynomial order in subarea A, the standard deviation increases between iteration 2 and 3. The percentage change is -3.7 %. It is assessed that this change cannot be explained as random noise. Therefore this 3rd order should not be used since it might give a bad description of the terrain. In other words a percentage change of -3.7 % cannot be accepted. However percentage changes of -1.1 and -2.3 %, as found in Table 13, should be accepted. The reason is

that these values are accepted in the visual assessment of when the steady state was reached. The lower limit should be placed between -2.3 and -3.7 %. A compromise is made and the lower limit is chosen to be -2.5 %.

One can think of the upper and lower stop criterions as a definition of an “accept area”. The accept area is illustrated at Figure 39, where the iterations will stop at iteration 5, because the standard deviation at this point is inside the accept area.

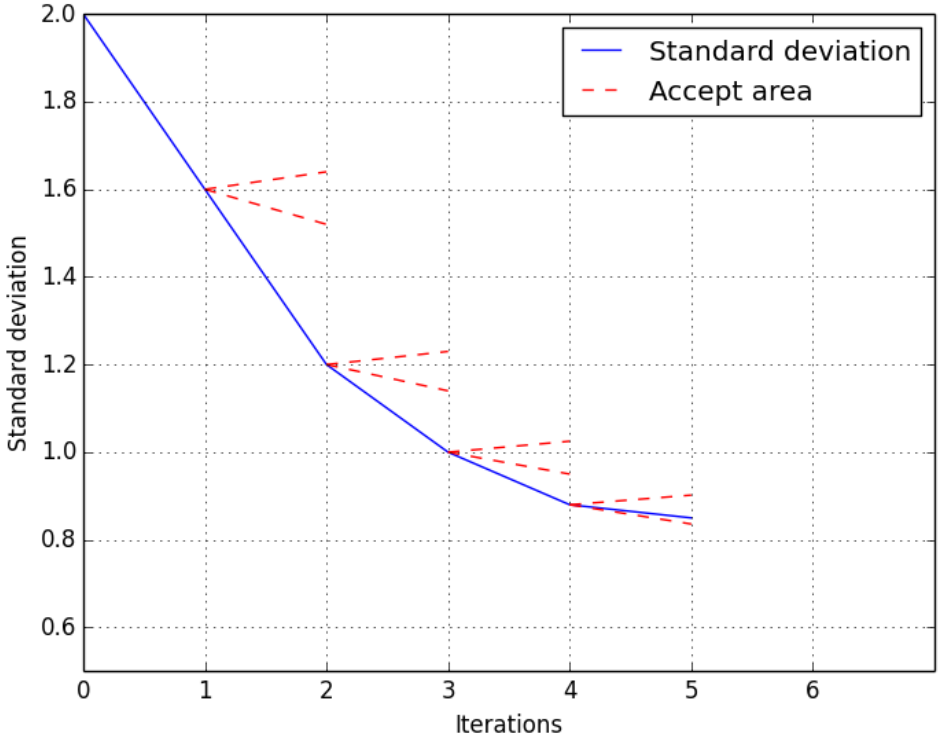


Figure 39 – Illustration of the chosen stop criterion. It can be seen that the accept area decrease when the standard deviation decrease.

When the standard deviation increases between two iterations with a percentage change of more than -2.5 %, then the next iteration is also calculated. Here a change is made since the comparison of the standard deviation should not be with the previous iteration, but with the iteration before the previous. If the standard deviation continues to grow for several iterations it is always the last time where the standard deviation decreased that is the base of the comparison. The principle is shown in Figure 40, where the red line indicates the standard deviations which should be compared.

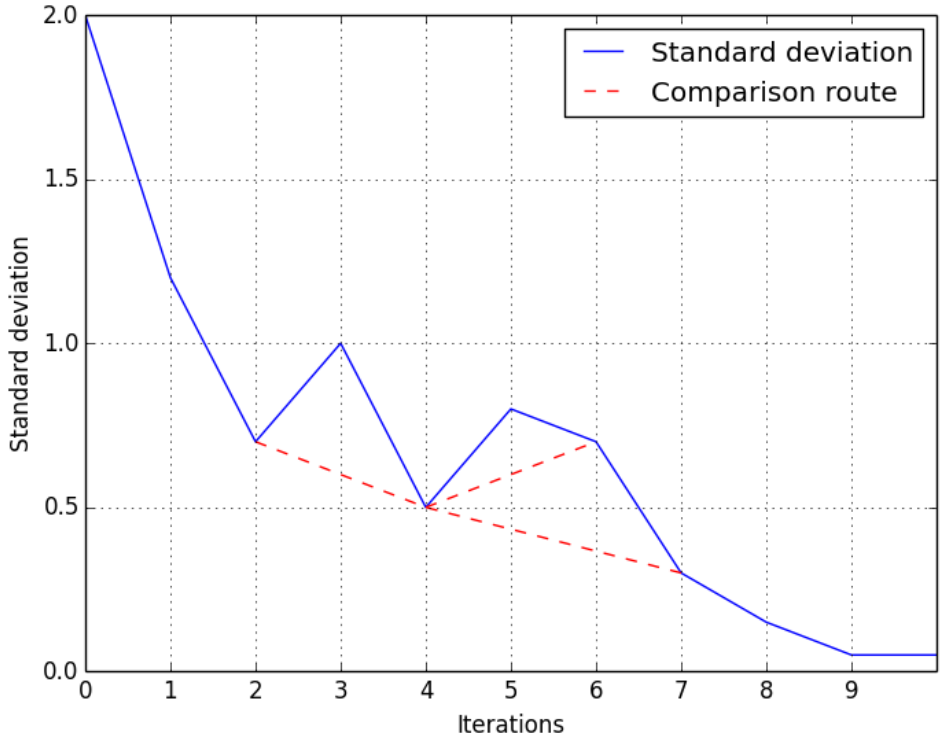


Figure 40 – Illustration of comparison route when the standard deviation increases between two iterations

4.5.3 Polynomial order

Like the iteration test, two parameters should be found when it comes to the polynomial order. The first parameter is the *maximal order*, which have something to do with the number of points in the sample. The other parameter is the *stop criterion*, which decides if a polynomial order should be used or if a polynomial with a higher order should be calculated.

4.5.3.1 Maximal polynomial order

The maximal polynomial order which can be used has something to do with the number of points in the square working area. The squares are 100 x 100 meters and with a block minimum of 10 m the number of points in a square is about 100. Sometimes it may be a little less due to possible holes and the borders in the point cloud. When fitting a polynomial to the point cloud it is the coefficients which are the unknowns. For every unknown in the polynomial there must be at least one point in the subarea. However, more points than unknowns (redundancy) will give a more reliable result. Furthermore, if there are some non-terrain points in the square there should be more points than unknowns in order to fit the surface to the terrain.

The number of unknowns in a polynomial can be found using the following formula (Cederholm [2] 2004):

$$\text{number of unknowns} = 1 + 2 \cdot n + n \cdot \frac{n-1}{2} \tag{4.4}$$

where n *is the order of the polynomial*

Based on Equation 4.4, the plot at Figure 41 can be made. The plot shows the relationship between the polynomial order and the number of unknowns. The 100 points in a square means that in theory a 12 order polynomial can be used. However, not every square contains 100 points due to borders and holes in the inputted point cloud. Furthermore, some of the points might be non-terrain points. The non-terrain points are weighted out in the calculations, which mean that the polynomial order should be lower than what would work theoretically.

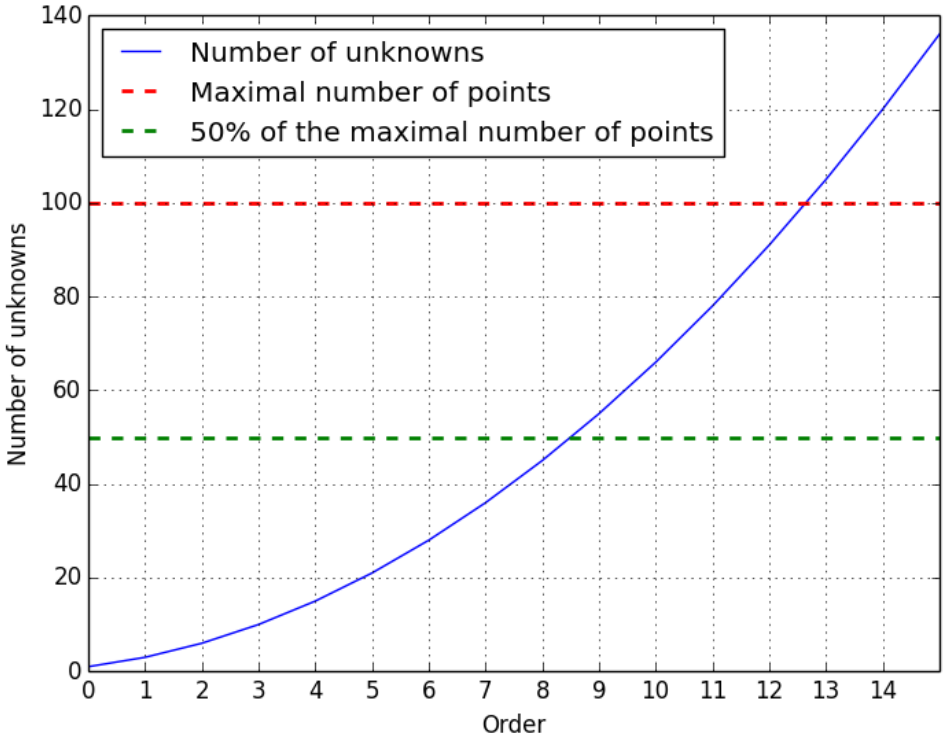


Figure 41 – Number of unknowns for polynomial orders

The maximal allowed order is implemented in the script as a varying factor which changes according to the number of points in the subarea. To make sure that there are more points than unknowns, it is decided that there should be two points in the subarea for every unknown in the polynomial. If there is less than 10 points in the sample, the square should be skipped. With 10 points it is possible to fit a plane and still have a number of outliers.

4.5.3.2 Order stop criterion

To find the optimal stop criterion for the polynomial, a test has been made. Again, the new parameters for the weight function are used in the test. In the test, a polynomial with order 0 to 10 is fitted to each subarea. The number of iterations for each order is found using the iteration stop criterion found in the previous section. Based on the test, plots of the standard deviation as a function of the polynomial order can be made for each subarea, see Figure 42. Plots of the fitted surfaces can be seen in Appendix E.

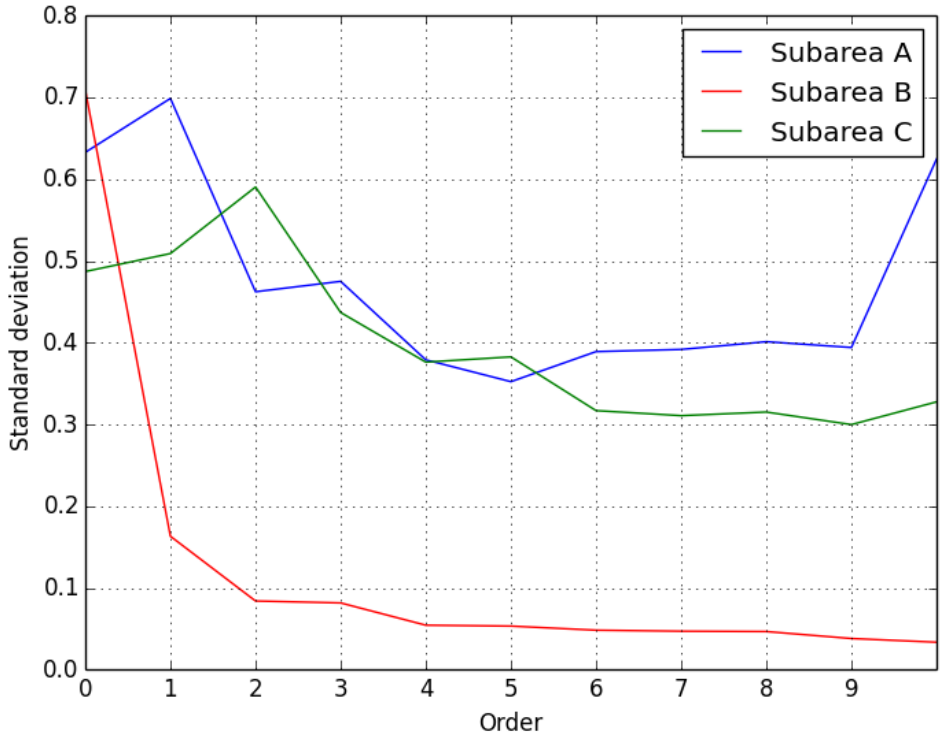


Figure 42 - Standard deviation for each order in the subareas when the optimal iteration is reached.

The order where a steady state has been reached is visually assessed from plots of the surfaces in Appendix E, and from the plot in Figure 42 of the standard deviation. The visual inspection of the surface plots gives an interval of polynomial orders which fits the terrain in each subarea. A specific order inside the interval is visually chosen as the place where the standard deviation has reached a steady state on Figure 42. The percentage change between the chosen order and the previous order is shown in Table 14.

Subarea	**Order from pictures in Appendix E**	**Order from plot in Figure 42 (n)**	**Difference in standard deviation between order n-1 and n (%)**
A	3-5	5	7.1
B	2-10	3	2.8
C	6-7	7	1.9

Table 14 – Selected orders from surface plots and plot of standard deviation

On Figure 42 it can be seen that for subarea A and C the standard deviation increases between two consecutive orders, especially in the beginning. For subarea A it is furthermore between order 2 and 3 and the percentage change is -2.8 %. This could be okay since the interval adopted from the pictures gives an interval between order 3 and 5.

In subarea C the standard deviation increases between order 4 and 5 where the percentage change is -1.7 %. The change is smaller than for subarea A, but it cannot be accepted since order 5 in subarea C is outside the selected interval from 6 to 7. This means that the percentage change should be between 0 and -1.7 %. As a compromise it is decided to use a lower stop

criterion of -0.5 %. If a percentage change is more negative than the lower stop criterion, then the next order, n, is calculated and the standard deviation is compared to order n-2.

An upper limit should also be decided. The percentage changes in Table 14 shows that the biggest change is in subarea A, where a change of 7.1 % is needed in order to select order 5. The upper stop criterion should therefore be a little more than 7 %. It is decided to use a stop criterion of 8 %.

4.5.4 Results with selected iteration and order parameters

In order to test the chosen parameters, the algorithm is applied to the three subareas. With the new weight function and the stop criteria for iterations and order, a polynomial order is automatically selected for each subarea. Figure 43 shows the results for all three subareas.

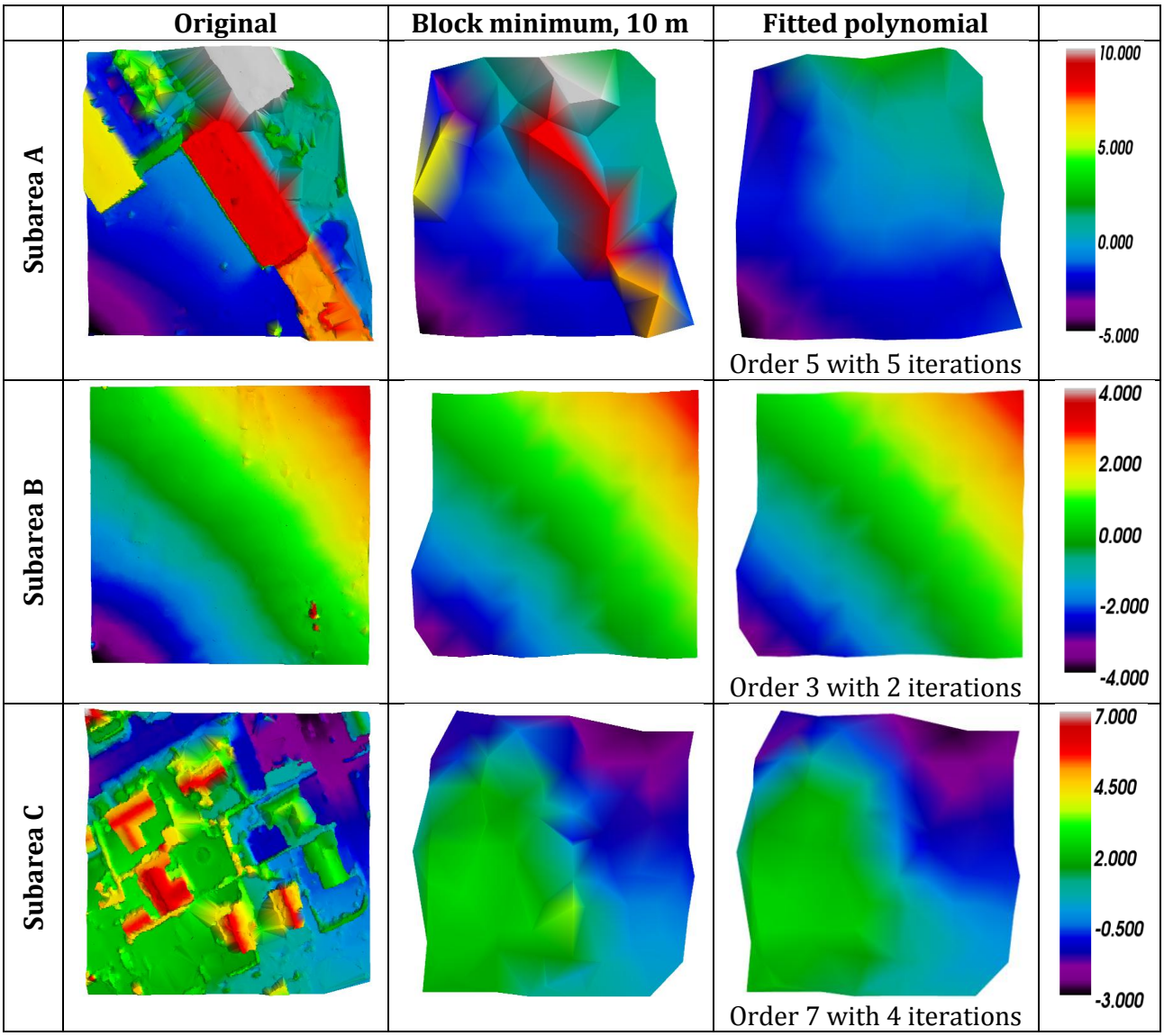


Figure 43 - Selected polynomial order in comparison with the original point cloud and the block minimum point cloud

4.5.5 Buffer

Points in the original point cloud are selected inside a buffer on each side of the fitted surface. The buffer is an asymmetrical buffer. The buffer above the surface is called the upper buffer and the buffer below the surface is called the lower buffer. Figure 44 shows an illustration of the buffer principle, where points inside the gray area will be selected as initial terrain points.

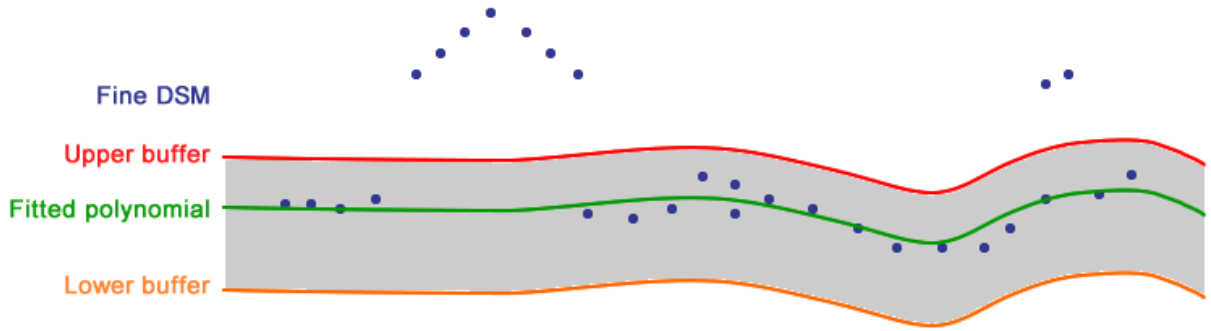


Figure 44 – Illustration of the buffer principle, with an upper buffer above the surface and a lower buffer below the surface

Non-terrain points are expected only to appear above the fitted surface. Therefore the lower buffer could in theory be infinite. However in order to remove possible low outliers a lower buffer of 2 m is used as proposed in the project by Matthesen and Schmidt (2014).

The next thing to decide is the size of the upper buffer. The purpose of step 1 is to remove the big clusters of non-terrain points, so a buffer should be selected in a way that removes the clusters and at the same time preserves the terrain details and smaller discontinuities in the terrain. It should be noted that it is okay that there are non-terrain points in the point cloud after step 1, because the non-terrain points which are not clustered are to be removed in step 2.

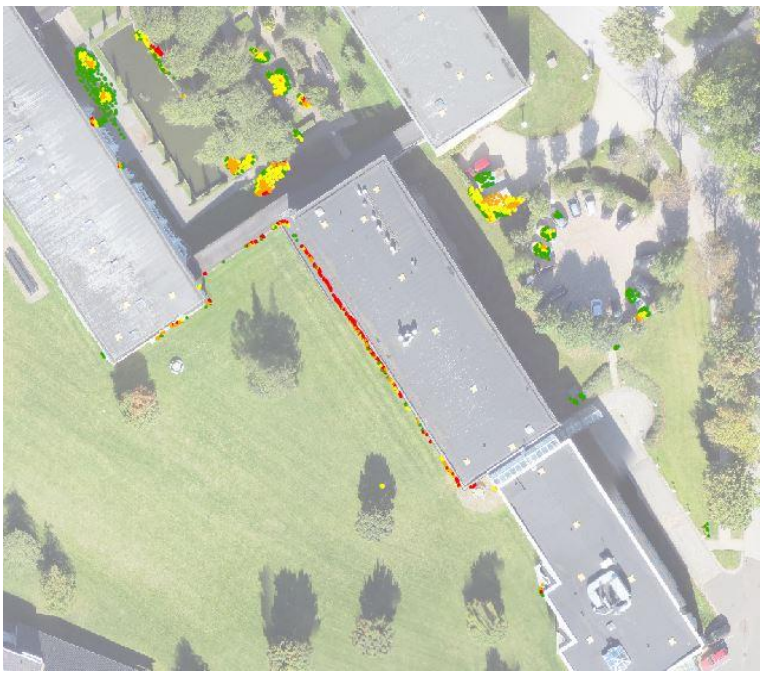
Different sizes of the upper buffer are tested for subarea A and C. Subarea B is not interesting for the buffer test since almost none non-terrain points exist in that area. The results with a buffer of 1, 2 and 3 m are shown in Appendix F.

From the figures in Appendix F it can be seen that a buffer of 1 m is too small. Especially in subarea A it looks like some terrain points are removed. A buffer of 3 m seems a little too big. This can especially be seen in subarea C where buildings start to appear with the 3 m buffer. The upper buffer should therefore be around 2 m. In order to more closely examine the size of the upper buffer, each point is saved together with the residual from the surface fitting. Thereby it is possible to import the points into GIS and easily visualize the points in a useful way. The visualization can be found in Figure 45 where points selected with buffers between 1.5 and 2.5 m are shown.

In subarea A no big clusters are selected no matter the buffer size. Furthermore, none of the visualized points are at the terrain. In other words, by looking isolated at subarea A, the buffer could be selected as any number between 1.5 and 2.5 m.

In subarea C there are more points to study. Especially the orange cluster of points in the middle (marked with a black circle at Figure 45) should be noticed. These points are between 2 and 2.25 m above the fitted surface and they are all non-terrain points. Such big clusters should be removed in step 1, and therefore the upper buffer should definitely be lower than 2 meters.

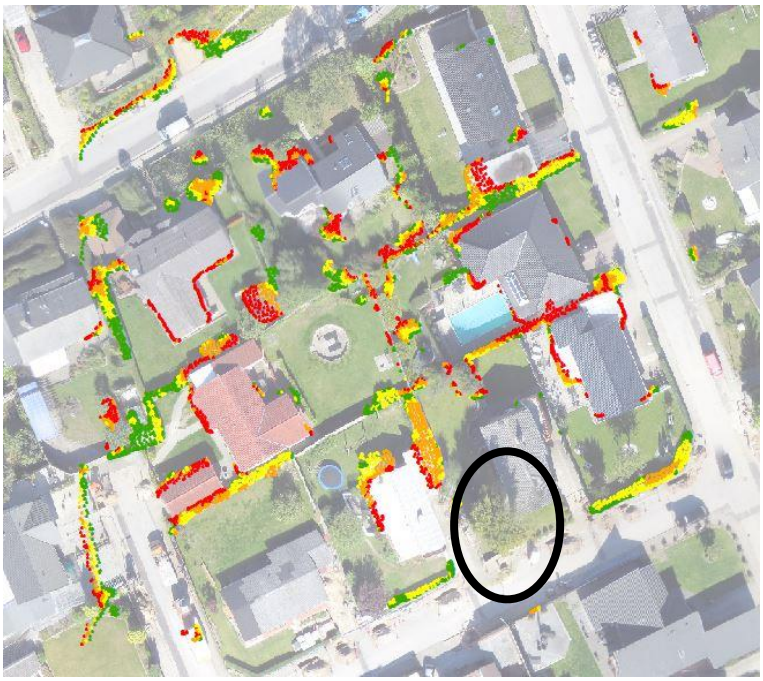
Based on the observations made, it is now known that a buffer of 1 meter cuts valid terrain, and a buffer higher than 2 meters results in selection of clustered points. As a compromise it is therefore decided to use a buffer of 1.5 meters. This also looks like a good choice according to Figure 45 where it seems like all points above 1.5 m are non-terrain points.



Subarea A

Distance above polynomial:

- 1.5 - 1.75 m
- 1.75 - 2 m
- 2 - 2.25 m
- 2.25 - 2.5 m



Subarea C

Distance above polynomial:

- 1.5 - 1.75 m
- 1.75 - 2 m
- 2 - 2.25 m
- 2.25 - 2.5 m

Figure 45 - Buffer between 1.5 and 2.5 m in subarea A and C

4.6 Conclusion

The purpose of this chapter is to develop the first step of the algorithm, and thereby answer the 1st question from the problem statement:

How can the surface-based filtering in step 1 be developed to be intelligent and automatically adapt to the terrain?

Step 1 is split into two sub-steps with a block minimum filter as the first and surface-based filtering afterwards.

The block-minimum filter is applied to the fine DSM before the surface filtering, in order to make it easier to filter away big clusters of non-terrain points, because the block minimum ensures that fewer points will be on e.g. rooftops. In a photogrammetrically produced point cloud the point density varies depending on the texture in the photos. This means that areas with a good texture will influence the surface fitting more than areas with bad texture. The block minimum ensures that this won't happen since it gives a more even distribution of points throughout the total area.

Only one parameter, the grid size, is used for the block minimum filter, see Table 15.

The second part of step 1 is the surface-based filtering. The surface based filtering use the coarse DSM produced by the block minimum filter and works inside smaller slightly overlapping squares. The surface fitting works by using least squares to fit a surface polynomial to the point cloud and iteratively outweigh the non-terrain points using a weight function.

The surface-based filtering is made intelligent, meaning that the algorithm automatically chooses a polynomial order which fits the terrain. This is done by iterating through different polynomial orders starting from order 0 and continuing until a certain stop criterion is reached. The stop criterion is based on the standard deviation of unit weight.

For every order a number of iterations are done where the non-terrain points are gradually outweighed. When a new iteration does not lead to any significant change of the fitted polynomial, the iterations are stopped. An insignificant change is determined by a given stop criterion. Again, the stop criterion is based on the standard deviation of unit weight.

After the best polynomial is found, and the non-terrain points are iteratively outweighed, the last part of the algorithm is to select points from the fine DSM which are inside a buffer around the fitted polynomial.

The parameters used for the surface fitting are through testing found to be as shown in Table 16.

Step 1.1	
Block minimum	
Grid size	10 m

Table 15 – Parameter used in step 1.1

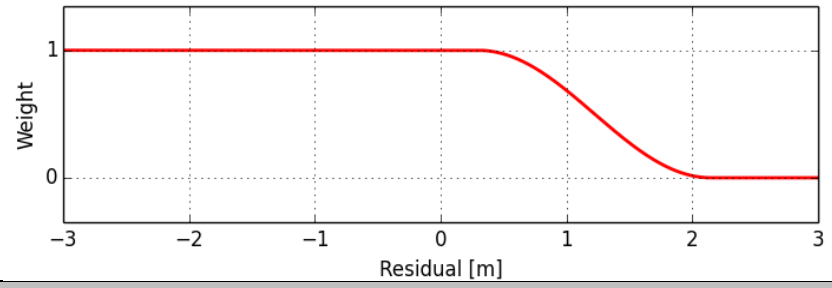
Step 1.2	
Squares	
Size of square	100 x 100 m
Overlap	30 m
Weight function	
Shift value and steepness	a = 0.3 , b = 1.7 
Iterations	
Maximum number of iterations	12
Lower stop criterion	-2.5%
Upper stop criterion	4%
Polynomial order	
Maximal order	Varying according to the number of points in the subarea. At least two points for every unknown.
Lower stop criterion	-0.5%
Upper stop criterion	8%
Buffer	
Lower buffer	2 m
Upper buffer	1.5 m

Table 16 – Parameters used in step 1.2

Chapter 5: Step 2

5.1 Introduction

Step 2 of the filtering algorithm is about removing the rest of the non-terrain points, which are still present in the dataset after the initial filtering in step 1. In other words, step 2 is about the *final filtering* of the point cloud. Slope-based filtering methods will be used for the final filtering, as decided in section 2.4 . This means that for a given evaluation point, an evaluation function and the neighboring points are used to determine if a given point is a terrain or non-terrain point.

Since the neighboring points has to be found for each point in the n points big dataset, the algorithm has to iterate through the point cloud n^2 times, if no indexing is used. The point cloud used in this current project contains approx. 2 million points. This means that the algorithm has to iterate through the point cloud 4 trillion[2] ($4 * 10^{12}$) times. In other words, the algorithm will be extremely slow if no spatial indexing is used. The first section of this chapter will therefore be about the creation of a Cython class which can be used for spatial indexing. The rest of the chapter will be about the development of a slope-based filter.

A simple implementation of a slope-based filter works by simply calculating the height difference and the 2D distance to the neighboring points. Afterwards, the height differences are compared to an evaluation function, which defines the maximum allowed height difference as a function of the 2D distance. This simple implementation works well if the project area is totally flat, but if the area is sloped, it is not a good solution, since it can give misleading results, see Figure 46.

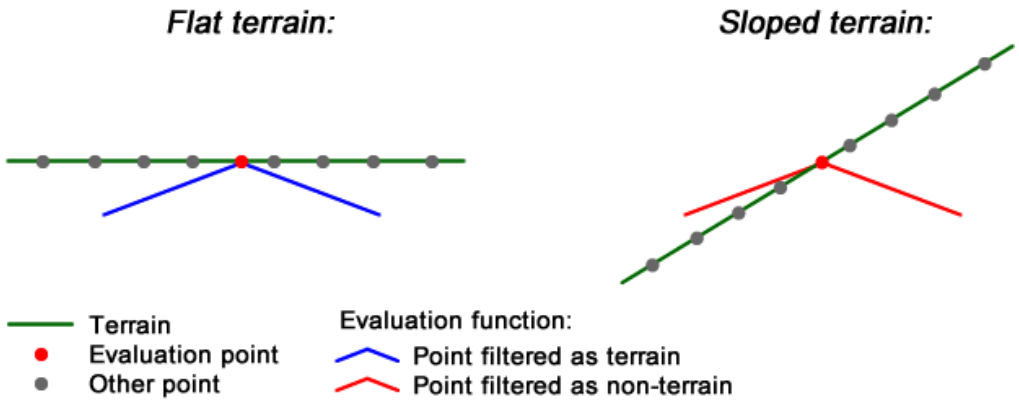


Figure 46 – The simple implementation of the slope-based filtering algorithm can lead to misleading results

A better solution is, for each evaluation point, to fit a plane to the terrain using the neighboring points. All points can then be transformed into a coordinate system where the plane spanned by the EN axes is parallel with the fitted plane. Afterwards, the evaluation function can be applied to the points in this new coordinate system, and thereby the influence of the mean slope has been removed. The advantage of this implementation is illustrated at Figure 47.

[2] Corresponding to the Danish term 'billion'

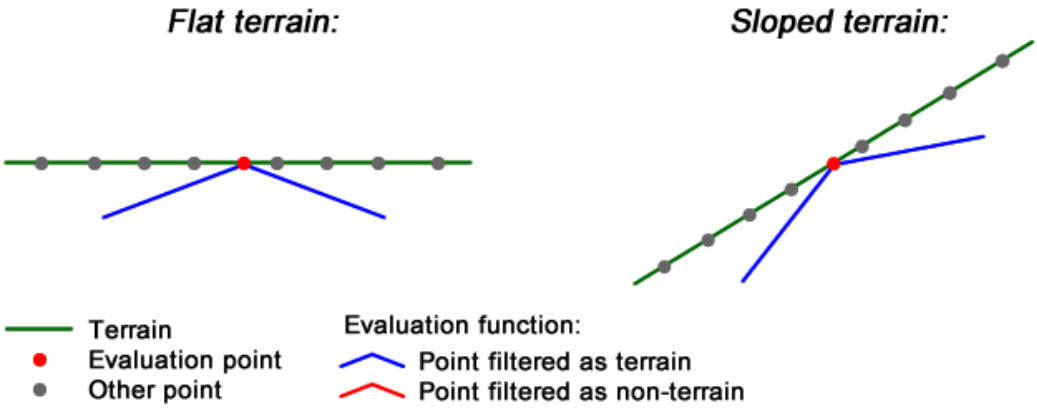


Figure 47 – For the best results, the evaluation function should be adjusted according to the mean slope

It is decided to use the proposed implementation of the slope-based filter where the dataset is corrected for the mean slope by fitting a plane. This is decided since there are some quite hilly areas on the project area. Furthermore, this more advanced implementation makes the algorithm more versatile and intelligent, because it automatically adapts itself based on the characteristics of the inputted dataset.

The search radius used to find the neighboring points should be small in order to avoid cutting valid terrain. At the same time, the search area should be bigger than the largest object which should be filtered away. Cars and other vehicles are the biggest objects expected to exist in the dataset after the initial filtering. It is assessed that a *search radius of 3 meters* is sufficient to remove the remaining non-terrain objects.

It should also be noted that a minimum of 3 neighboring points are needed in order to fit a plane. In order to get a reliable result, it is however assessed that a *minimum of 10 points* should be used when fitting a plane, since this ensures redundancy. If there are less than 10 points within the 3 meter radius from a given evaluation point, it is decided that the evaluation point should simply be skipped and marked as non-terrain. The reason for this choice is that the evaluation point in that case most likely is part of an isolated cluster of points which cannot be trusted as terrain points, since they might be measured on top of trees, see Figure 48. However, it should be noted that this choice might lead to a wrong classification of isolated terrain points measured on the ground inside e.g. a clearing in a forested area. Such errors are acceptable since it is assessed that it is better to remove valid terrain points compared to the risk of keeping non-terrain points in the final DTM.

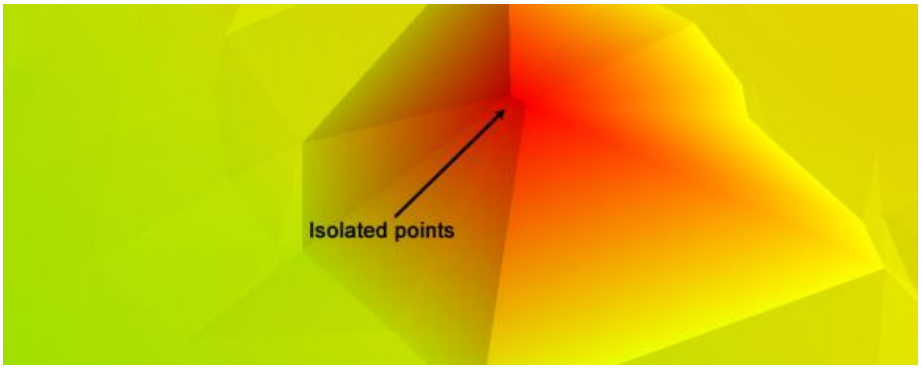


Figure 48 - Isolated points

The flow diagram below shows the overall structure of the slope-based filter, which should be developed:

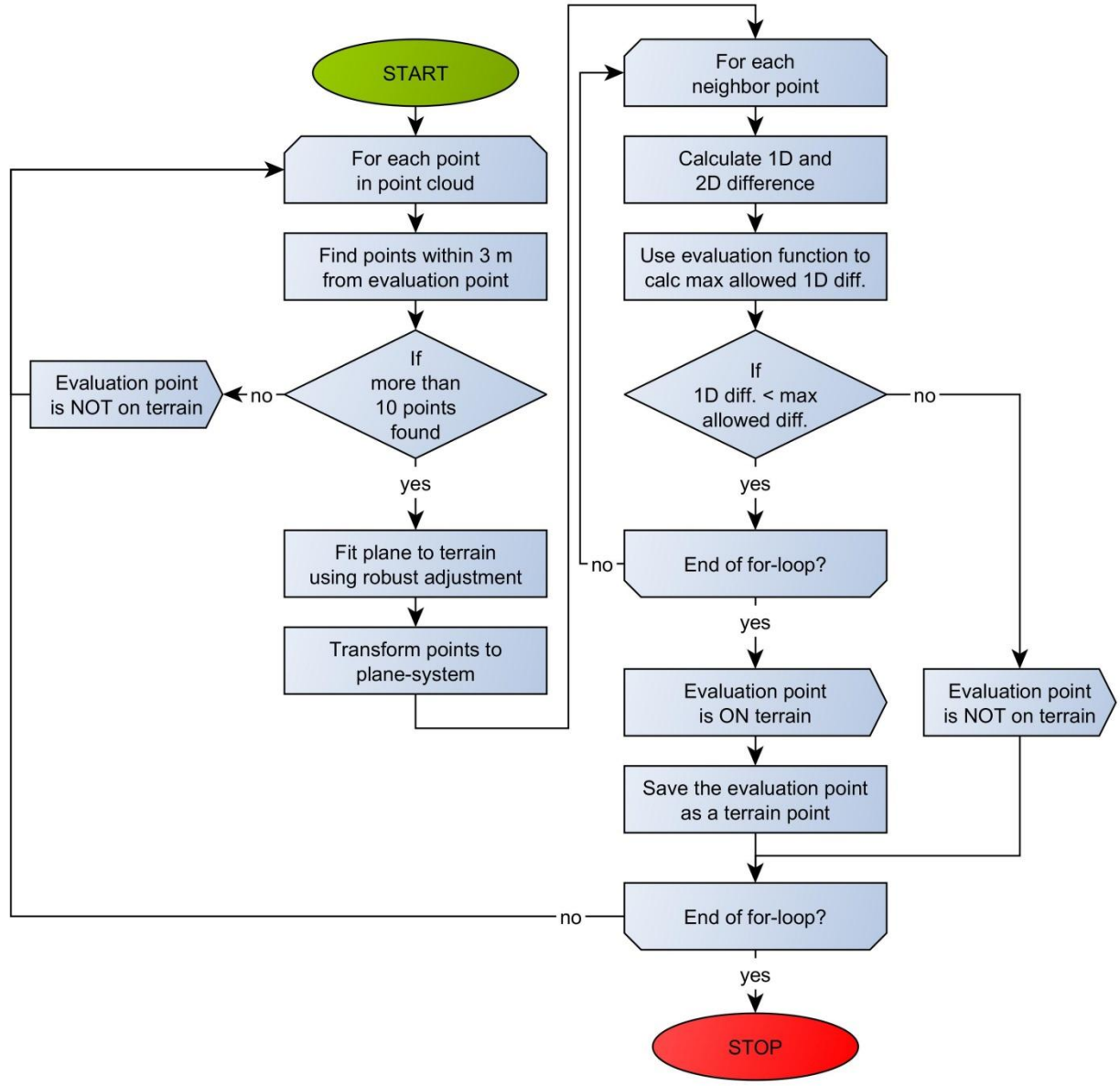


Figure 49 – Flowchart showing the overall structure for the slope-based filtering algorithm

The elements of the flow diagram are explained in detail in the later subsections of this chapter. However, as earlier mentioned, the first section will be about spatial indexing. The next section will be about how to correct for the slope of the dataset. Finally, the chapter is ended with a description of the used evaluation function, and a test to find the optimal parameters.

5.2 Spatial indexing

In a slope-based filtering algorithm, for each point in the point cloud, the neighboring points within a given search radius have to be found. If no indexation of the point cloud is used, the algorithm has to calculate the 2D distance to every single point in the point cloud for each evaluation point. Such a procedure is very time consuming since the point cloud consists of

millions of points. In order to test the algorithm properly, it is important that the algorithm run reasonably fast. If the algorithm takes days to complete, you will not try out new and maybe better sets of parameters, since it simply takes too much time. In order to try to avoid such problems, it is decided to write a class in Cython, which can be used for indexing of the point cloud.

Multiple methods for indexing of spatial data exist. One of the simplest methods is *grid-based spatial indexing*. A grid-based index works best if the points are uniformly distributed, otherwise too much time will be spend going through empty index cells. More advanced methods also exist, but in this project it is decided to keep it simple and use a grid-based index.(Shekhar og Chawla 2003, 96-99)

Grid-based indexing works by simply putting a grid over all the points, and then store the points sorted by the cell which they are placed in. When searching for neighboring points within a given radius from an evaluation point, it is now possible to look only at the points in the cells close to the evaluation point. Without indexing, the algorithm has to go through the entire dataset. This is illustrated at Figure 50.

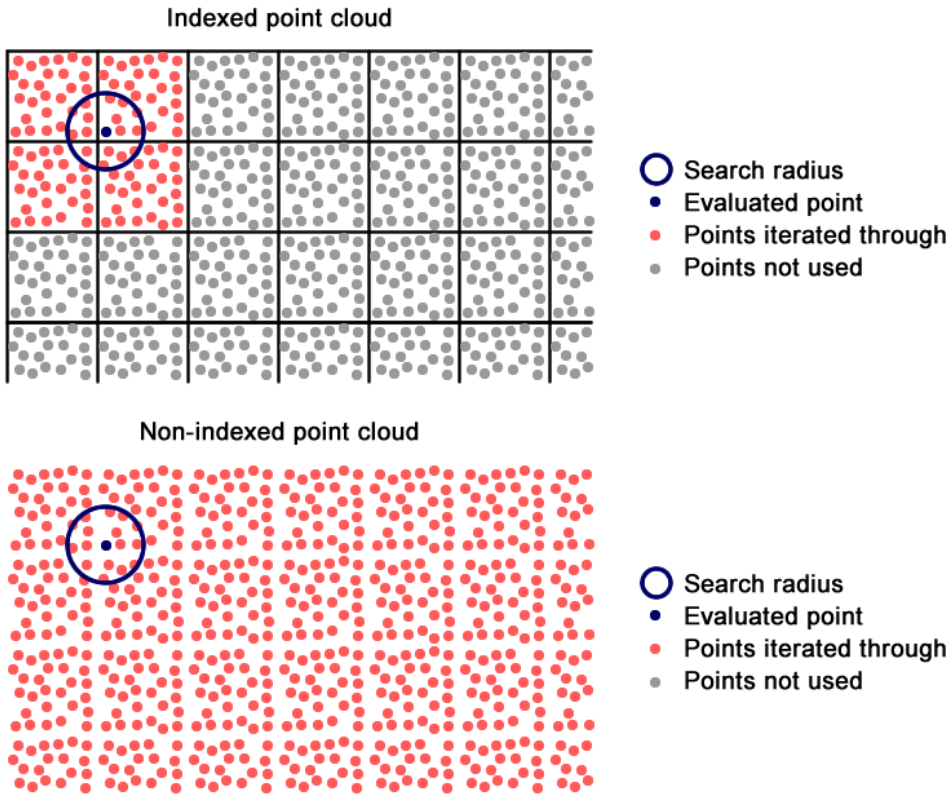


Figure 50 – Points used to find neighboring points within a given radius with and without indexing.

In this project, the grid-based indexing is implemented by placing a grid over the point cloud. Each cell in the grid has a number, from zero and upwards, see Figure 51. The points in the initial point cloud are stored in a random order. Based on the grid, it is possible to sort the points according to the cell in which they appear. This means that a new sorted list is created, where the points in cell 0 are stored first, followed by the points in cell 1, which again are followed by the points in cell 2 and so on. An index array is also created, which for each cell

stores the point number from the sorted list which corresponds to the first point in the given cell, see Figure 51. With this structure, the index array can be used to quickly find the points inside a given cell, without the need of going through the entire point cloud.

Figure 51 – Illustration of the used implementation of grid-based indexing

The index array can be used to quickly find the points within a given cell. However, all points within a given search radius from an evaluation point should be found. This means that not only the points from one cell should be used, but sometimes also points from the neighboring cells. In order to find out which cells to look in, the 2D distance from the evaluation point to the center of each cell is calculated. If this distance is within a given max distance, the points in the cell are used. The max distance is illustrated at the figure below:

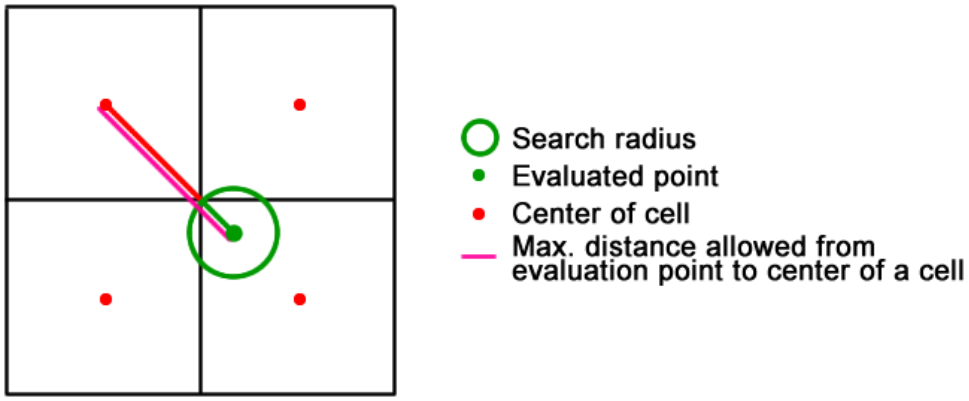


Figure 52 - Max distance used to find the correct cells when searching for points within a given search radius

The max distance can be described using the following formula:

$$\text{Max\_dist} = \sqrt{\left(\frac{\text{gridsize}}{2}\right)^2 + \left(\frac{\text{gridsize}}{2}\right)^2} + search\_radius \qquad (5.1)$$

where *'gridsize' is the grid size used for the indexing*
'search_radius' is the search radius used for selection of nearby points

As described in section 0, the search radius should be 3 meters. The grid size is however still undecided. In order to find the best grid size, the *time used* to find the points within a given search radius is tested with different grid sizes. In the tests, the points within a search radius of 3 meters from a random evaluation point in Subarea B are found. For each grid size, the test is repeated 100 times with a new random point and in the end the average time used for the given grid size is found. It is important to use a new random point in each iteration, since the position of the evaluation point relative to the grid decides the number of cells that the algorithm should go through. Furthermore, the impact of different point densities inside Subarea B is reduced by using of a new random point for each iteration. The graph at Figure 53 shows the test result.

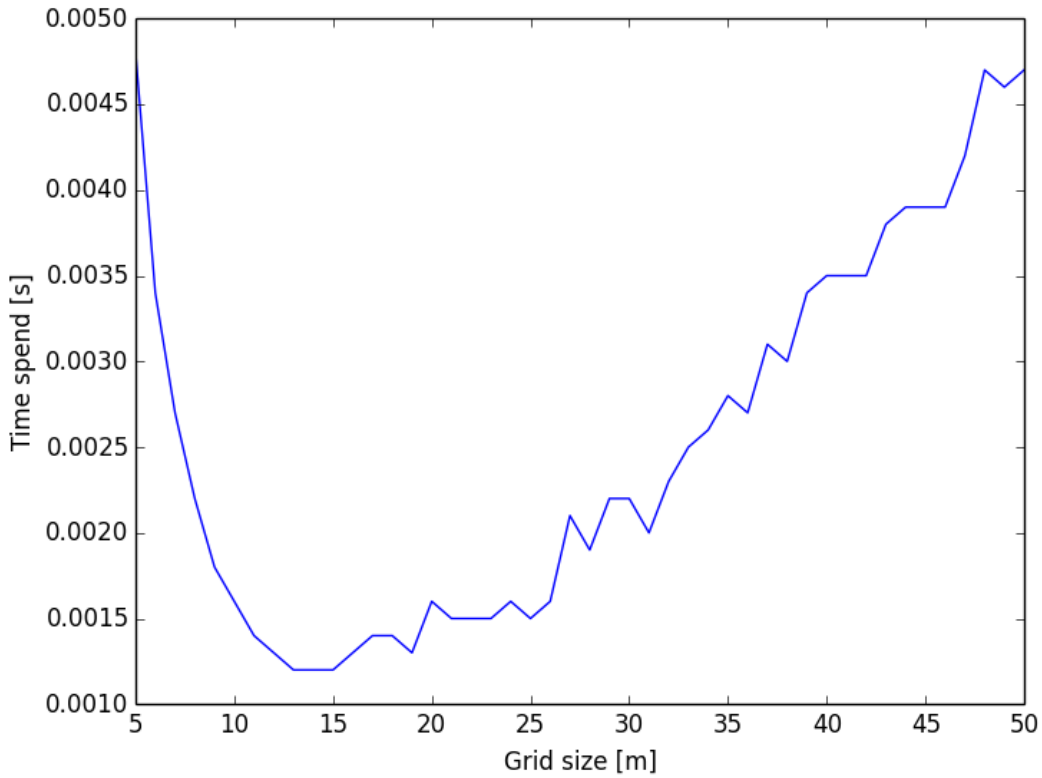


Figure 53 – For different grid sizes, the graph show the time spend to find the points inside a search radius of 3 meters from a random evaluation point inside Subarea B.

From Figure 53, it can be seen the best performance is achieved with a grid size of 14 meters. It is therefore decided to use a grid size of *14 meters* in this project. It should however be mentioned, that the test results are only valid with a search radius of 3 meters and for point clouds with a point density similar to the density found in Subarea B.

One might ask if the indexation gives any significant performance boost compared to not using indexing at all. In order to test this, a grid size of 14 meters is compared to a grid size of 10,000 meters. The grid size of 10,000 meters is so big that only one cell covers the entire project area and thus put the indexing out of action. As before, 100 random points inside Subarea B is used for the test where the points within a search radius of 3 meters are found. The results can be found in Table 17.

	Grid size [m]	Time spend [s]
With indexing	14	0.0012
No indexing	(10,000)	0.2868

Table 17 – Average time spend to find points within 3 meters from an evaluation point

From Table 17, it can be seen that the use of indexing is 0.2868s/0.0012s= *239 times* faster compared to no indexing. In other words, the use of indexing will significantly improve the performance of the algorithm developed in this current project.

To put the numbers in Table 17 into perspective, just finding the surrounding points near each point in a point cloud containing 2,000,000 points will take 0,67 hours for an indexed point cloud versus 159,33 hours for a non-indexed point cloud.

Below is an example of the output from a nearest neighbor query made to the indexing class. The points within a 3 meter radius are found by the indexing module:

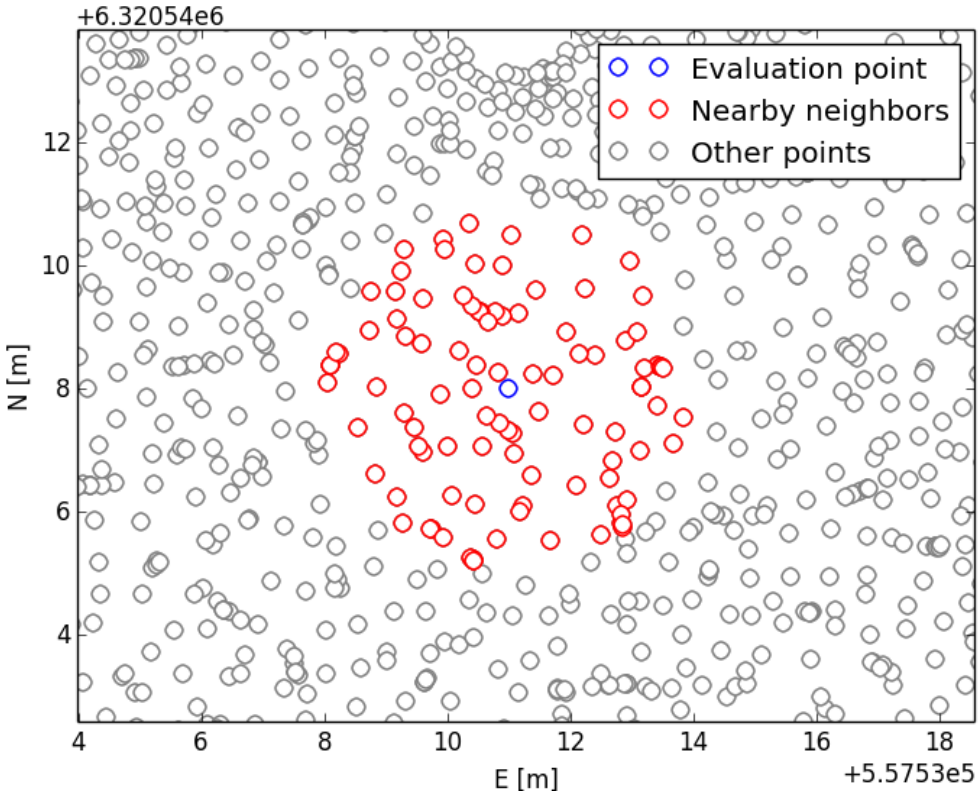


Figure 54 – Example of output from a nearest neighbor query made to the indexing class

5.3 Slope correction

For any given evaluation point, it is possible to find the neighboring points within a radius of 3 meters by using the indexing module described in section 5.2 . The mean slope of the terrain should be removed from the neighboring points, before an evaluation function can be used to classify points as either terrain or non-terrain points. This section is about how to fit a plane to the terrain, and then - based on the fitted plane - transform the points into a new coordinate system where the mean slope is removed.

5.3.1 Fitting of plane using robust adjustment

It is assessed that the terrain within the used search radius of 3 meters from the evaluation point can be described with a plane. By fitting a plane to the points surrounding the evaluation point, it is possible to calculate the slope and correct for it. It is decided to fit the plane using *robust adjustment*, where the influence of outliers (non-terrain points) is reduced.

The used method for robust adjustment is *iteratively reweighted least squares* (IRLS), where the residuals are minimized according to a weight function(Cederholm 2013). This is similar to the surface fitting method used for the initial selection of terrain points. However, a different (and symmetrical) weight function is used, which does not differentiate between positive and negative residuals. This is chosen due to the problem described in section 0, where it can be seen that the fitted plane does not give an adequate description of the terrain, since the plane keeps moving downwards until, in worst case, only 3 points are used to describe the plane.

It has been considered if a constraint should be added, saying that the plane should be levelled with a certain weight. Such a solution will help avoiding that the plane is wrongly fitted near large non-terrain objects. It will however require further investigations in order to decide the weight which should be used for the constraint. Due to limited time it is decided not to implement a constraint in this current project. It could however be a subject for further improvement of the algorithm.

For any given EN coordinate, the height in a plane can be described using the following formula:

$$H(E,N) = a_0 + a_1 \cdot E + a_2 \cdot N \tag{5.2}$$

where *H is the elevation*
E and N are the 2D coordinates
$a_0, a_1, and\ a_2$ *are the unknown coefficients*

In order to avoid numerical problems, it should be noted that the E, N and H coordinates used for the least squares calculations should be reduced by extracting the mean coordinates as described by the following formulas:

$$\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_m \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_m \end{bmatrix} - \frac{\sum_{i=1}^{m} E_i}{m} \tag{5.3}$$

$$\begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_m \end{bmatrix} = \begin{bmatrix} N_1 \\ N_2 \\ \vdots \\ N_m \end{bmatrix} - \frac{\sum_{i=1}^{m} N_i}{m} \tag{5.4}$$

$$\begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_m \end{bmatrix} = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_m \end{bmatrix} - \frac{\sum_{i=1}^{m} H_i}{m} \tag{5.5}$$

where *m is the number of observations*
e_i, n_i and h_i are the reduced coordinates used for the least squares calculations
E_i, N_i and H_i are the original coordinates in a national coordinate system

If *m* points are found near the evaluation point, then it is possible to setup *m* observation equations. The *observation equations* describe the relation between a measured point and the plane equation:

$$\begin{gathered} h_1 = a_0 + a_1 e_1 + a_2 n_1 - r_1 \\ h_2 = a_0 + a_1 e_2 + a_2 n_2 - r_2 \\ \vdots \\ h_m = a_0 + a_1 e_m + a_2 n_m - r_m \end{gathered} \tag{5.6}$$

Using matrix notation, it is possible to rewrite the observation equations in the form $\boldsymbol{b} = \boldsymbol{A} \cdot \boldsymbol{x} - \boldsymbol{r}$: (Cederholm [1] 2000, 22)

$$\begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_m \end{bmatrix} = \begin{bmatrix} 1 & e_1 & n_1 \\ 1 & e_2 & n_2 \\ \vdots & \vdots & \vdots \\ 1 & e_m & n_m \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} - \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{bmatrix} \tag{5.7}$$

$$b = A \cdot x - r$$

where *b is a vector with the H-coordinates of the measured points*
A is the design matrix
x is a vector with the unknown coefficients
r is a vector with the residuals

Each observation is given a weight according to the weight matrix C:

$$C = \begin{bmatrix} c_1 & & & \\ & c_2 & & \\ & & \ddots & \\ & & & c_m \end{bmatrix} \tag{5.8}$$

The weight matrix is a diagonal matrix, where each observation initially is equally weighted. In other words, the C matrix is initially an identity matrix.

Using least squares, an estimate of the unknown coefficients in the x vector can be found as: (Cederholm [1] 2000, 22)

$$\hat{x} = (A^T C A)^{-1} A^T C b \tag{5.9}$$

Afterwards, the residuals can be calculated as: (Cederholm [1] 2000, 23)

$$\hat{r} = A\hat{x} - b \tag{5.10}$$

The residuals can be used to correct the weight matrix according to the L_p-norm. The weight of the i[th] diagonal element in the weight matrix is updated according to the size of the i[th] residual using the following weight function: (Cederholm 2013)

$$c_i = |r_i|^{p-2} \tag{5.11}$$

where r_i *is the residual of the i[th] observation*
p *is the norm used to correct the weights*

If p=1, and the residual is zero, then the above formula leads to division by zero, since $|r_i|^{-1} = \frac{1}{|r_i|}$. A workaround to this problem is too add a very small number to the residual:

$$c_i = (100\ eps + |r_i|)^{p-2} \tag{5.12}$$

The value *eps* used in the formula above is the so-called *machine epsilon*. The machine epsilon is defined as "*the smallest positive number which, when added to 1, gives a number different from 1.*"(Duraiswami 2009, 6) In order to be sure not to divide with zero, the number is multiplied with 100 (Cederholm 2013, 24). In python, the machine epsilon can be found using the function `spacing(1)` from the NumPy extension module.

It can be discussed what norm to use in Equation 5.12. The choice of norm determines how robust the adjustment is. Typically the norm should be between 1 and 2. The norm should not be exactly 2, because this will not result in a robust adjustment, but just a normal least squares adjustment, since all corrected weights still will be equal to 1 ($|r_i|^0 = 1$). An L_1 estimate, on the other hand, has a very strict weight function, which very aggressively outweighs observations. At the lectures at Aalborg University, an $L_{1.3}$ estimate has been recommended. (Cederholm 2013)

From the weight function at Figure 55, it seems like an $L_{1.3}$ estimate is also suitable in this current project, since it is expected that the points fits a plane with residuals of a few cm. It should be noted that the weight function is symmetrical around zero. This ensures that the fitted plane cannot move downwards in an uncontrolled manner, as earlier mentioned.

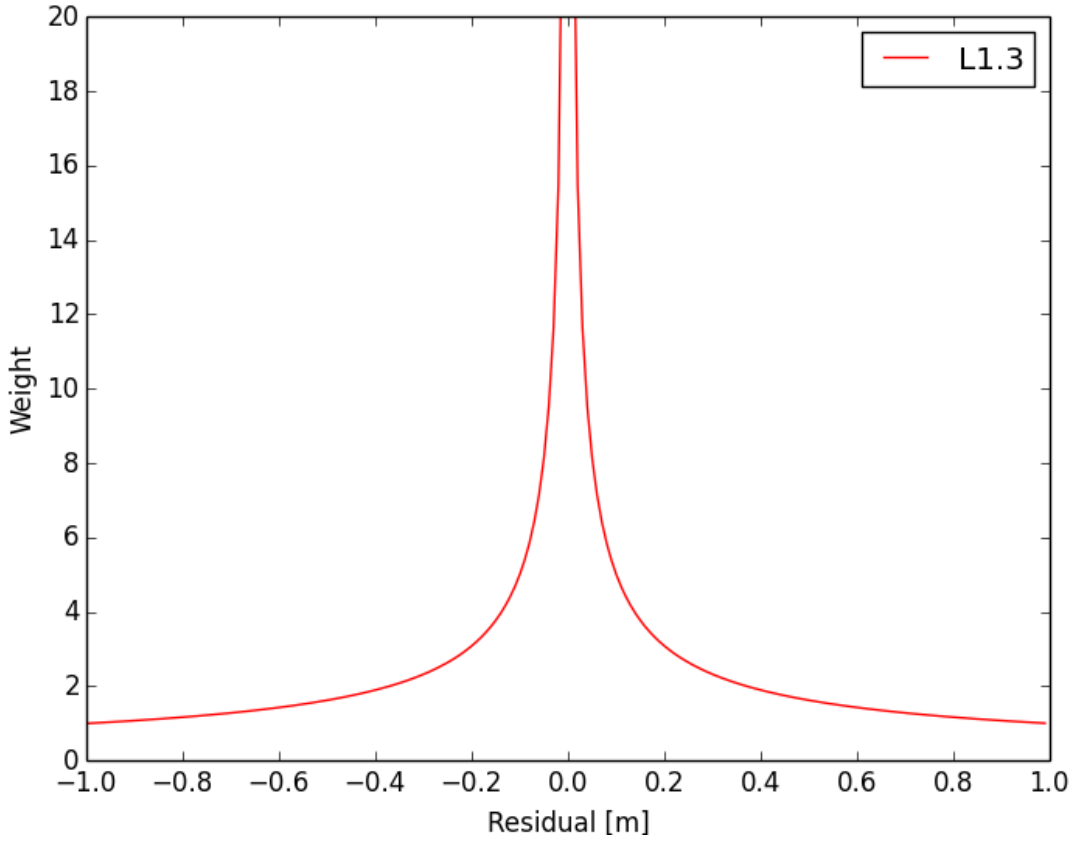


Figure 55 - The weight function for the $L_{1.3}$ estimate

After the weight matrix is corrected using the weight function, a new least squares estimate can be found and new residuals can be calculated. The residuals can again be used to update the weight matrix. The procedure can be iterated until convergence or a max number of iterations, as illustrated at Figure 56.

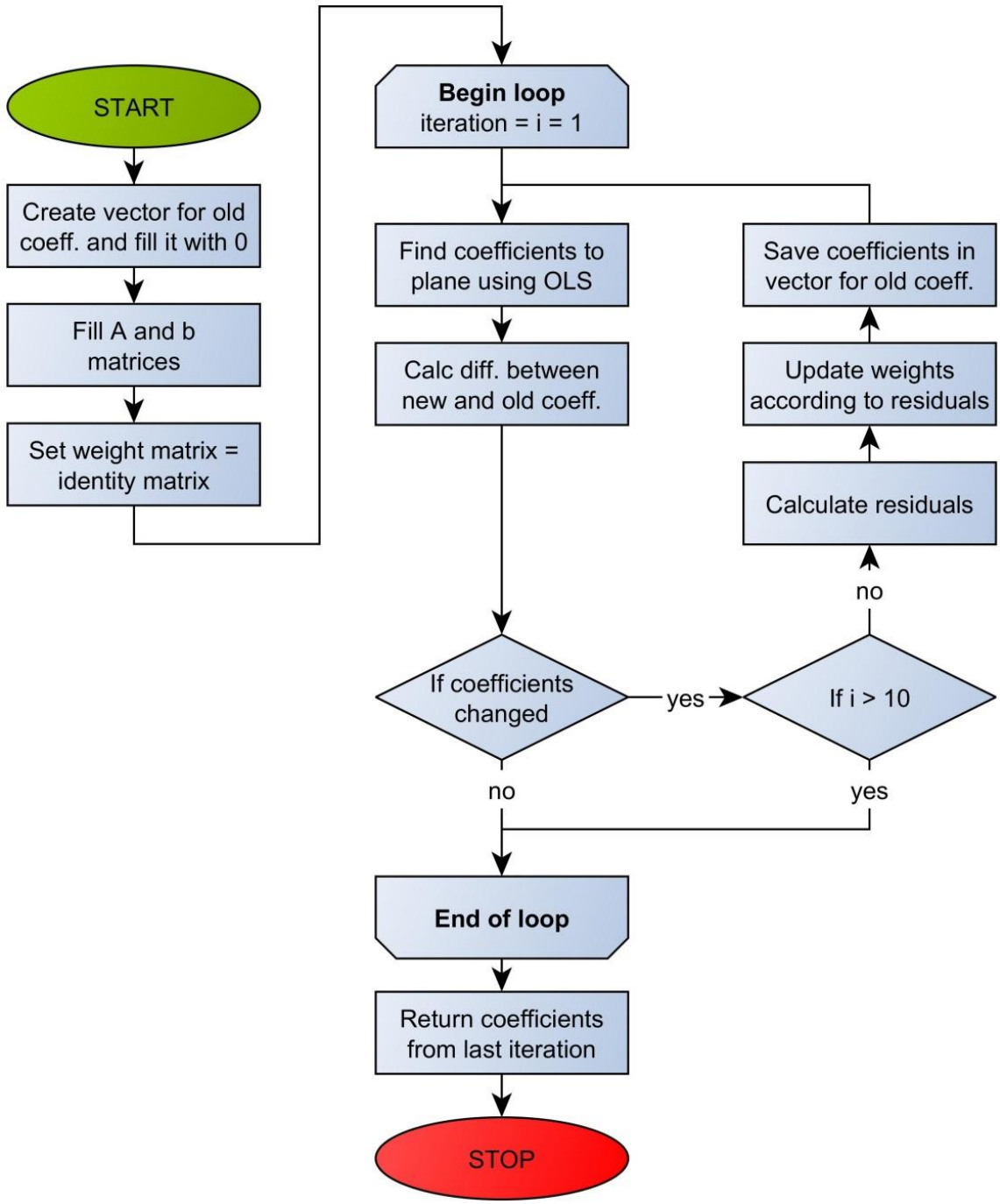


Figure 56 - Flow diagram showing the steps involved in robust fitting of a plane to a subset of the point cloud

A Cython module has been created, which can find the coefficients to a plane using an $L_{1.3}$ estimate. The Cython module takes advantage of the methods presented in Appendix C. This means that only the diagonal of the C matrix is stored in memory, and furthermore the algorithm takes advantage of the fact that the N matrix is symmetrical. For more details, see the Cython module at '*DVD:\python\program\module_final_selection\plane_fit_module\plane_fit.pyx*'.

In order to test if the robust adjustment works as expected, some tests are made using the python script '*DVD:\python\slope_based\plane_fit_plot_for_report.py*'. In the script, a test dataset is created based on a grid of E and N coordinates. The H coordinates are assigned using the following plane equation:

$$H(E,N) = 1 + 0.2 \cdot E + 0.4 \cdot N \tag{5.13}$$

The created dataset represents an ideal situation without any outliers, meaning that it should not be problematic to fit a plane and find the coefficients of the plane.

To find out how the plane fitting algorithm handles outliers, a noisier dataset has also been created. The second dataset is based on the same plane equation, but outlier points are added. The ideal and the noisy dataset can be seen at Figure 57.

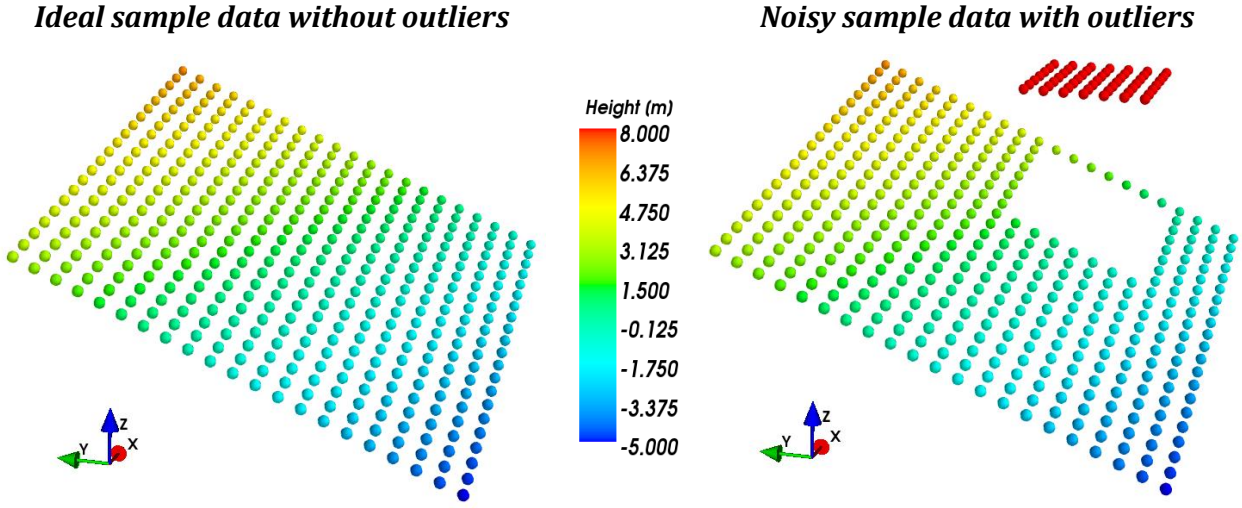
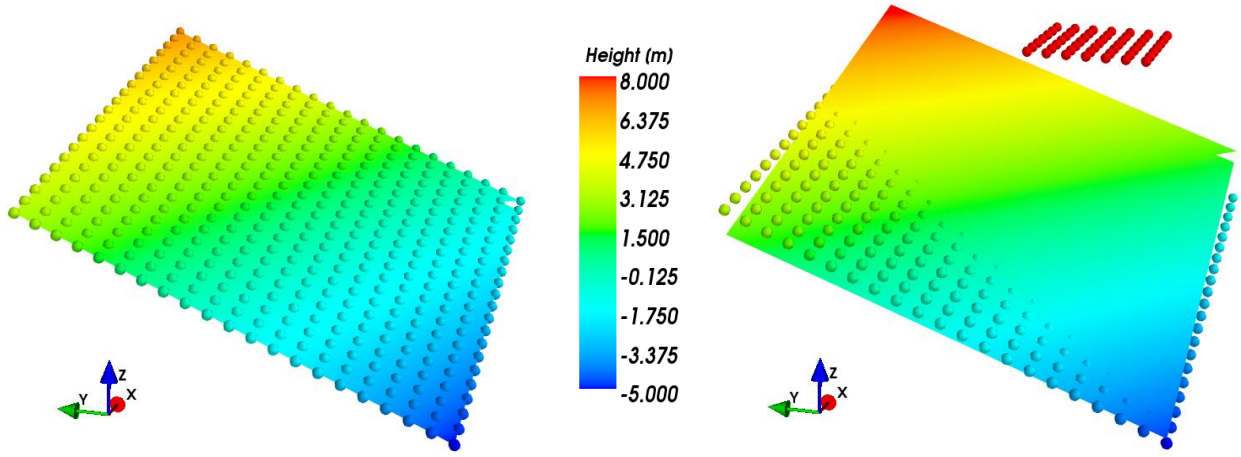


Figure 57 – Generated sample data used for testing of the robust plane fitting algorithm

A plane is fitted to the two datasets using both the $L_{1.3}$ and the L_2 norm. This way it is possible to test if the robust adjustment gives a more reliable result. The results can be seen at Figure 58 and Figure 59.

Figure 58 and Figure 59 show that the robust adjustment works as expected. It can be seen that the $L_{1.3}$ estimate is more robust since it can fit the correct plane, even with big outliers in the dataset. It can also be seen that the first coefficient is not equal to 1 as in the equation used to create the test data. The reason is that the coordinates are reduced before the plane is fitted. It is possible to correct the first coefficient, but it is not needed in this project since the slope is described only by the 2nd and 3rd coefficient.

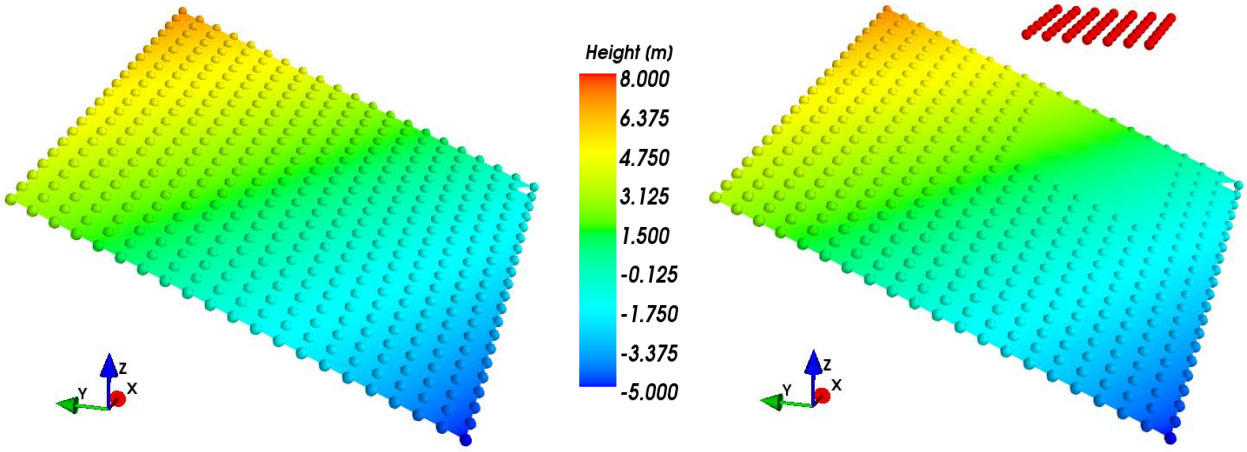
L_2 estimate for ideal data ***L_2 estimate for noisy data***



H(E,N)=0+0.2*E+0.4*N **H(E,N)=0+0.325*E+0.332*N**

Figure 58 – Planes fitted to points using least squares with equal weights

$L_{1.3}$ estimate for ideal data ***$L_{1.3}$ estimate for noisy data***



H(E,N)=0+0.2*E+0.4*N **H(E,N)=-0.728+0.208*E+0.396*N**

Figure 59 – Planes fitted to points using robust adjustment

5.3.2 Transform points to plane-system

A plane can be fitted to the points surrounding a given evaluation point, as described in the previous section. The next step is to remove the mean slope of the dataset based on the fitted plane. This can be done in different ways. A simple solution is, for each point, to simply extract the height which can be found using the plane equation found by fitting the plane. Another solution is to transform the points into a new coordinate system by rotating the dataset according the slope of the fitted plane. Figure 60 below illustrates the difference between the two suggested methods.

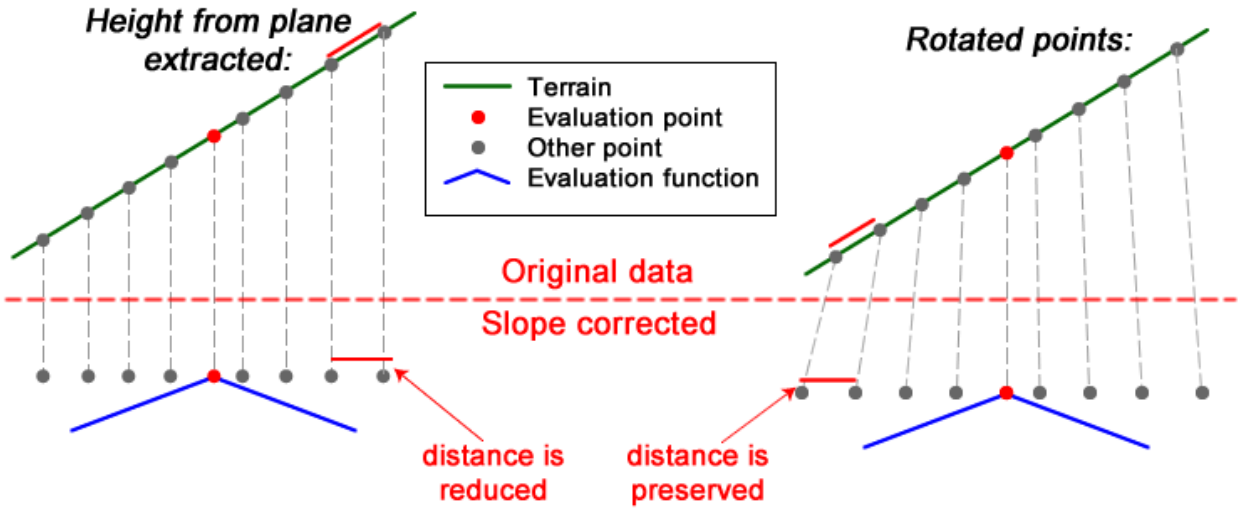


Figure 60 – Illustration of the different suggested methods for slope-correction.

It is assessed that the best solution is to rotate the points, since the distance along the terrain should be preserved after the mean slope is removed. It is therefore decided to rotate the points instead of simply extracting the heights according to the found plane equation.

It should however be mentioned that it is a lot more math intensive to rotate the points compared to the simple method, where heights are simply extracted according to the plane equation. The chosen method will therefore result in increased calculation time. If the slope of the terrain is small, the error caused by using the simple method is probably negligible. Further investigations could therefore be made, in order to clarify if it is overkill to use the method where the points are rotated.

The slope of the fitted plane can be described using the normal vector to the fitted surface. A normal vector to a plane can be found as: (Weisstein n.d.)

$$n = \begin{bmatrix} a_1 \\ a_2 \\ -1 \end{bmatrix} \tag{5.14}$$

where a_1 *is the 2nd coefficient to the fitted plane*
a_2 *is the 3rd coefficient to the fitted plane*

Figure 61 below shows an example of the normal vector, $n = [a_1, a_2, -1]$, to a plane described using the plane equation $H(E,N) = a_0 + a_1 \cdot E + a_2 \cdot N$.

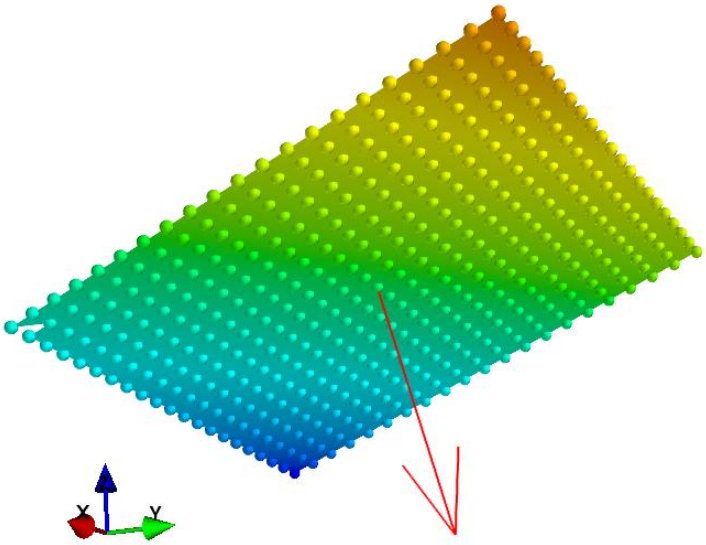


Figure 61 – Normal vector to a plane

The normal vector can be pictured in 2D:

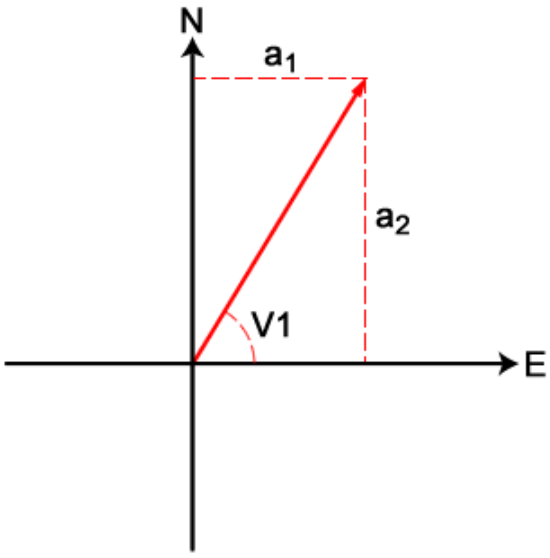


Figure 62 – The normal vector and its 2D components

The direction of the steepest slope can be found based on the normal vector in 2D:

$$V1 = \tan^{-1}\left(\frac{a_2}{a_1}\right) \tag{5.15}$$

When using the formula above, you have to correct the result according to the quadrant where the normal vector's end point is positioned. This is, as in most major programming languages, automatically done in Python by using a function called `atan2(a2,a1)`.

The steepest slope of the plane can also be found based on the normal vector, as illustrated at Figure 63 below:

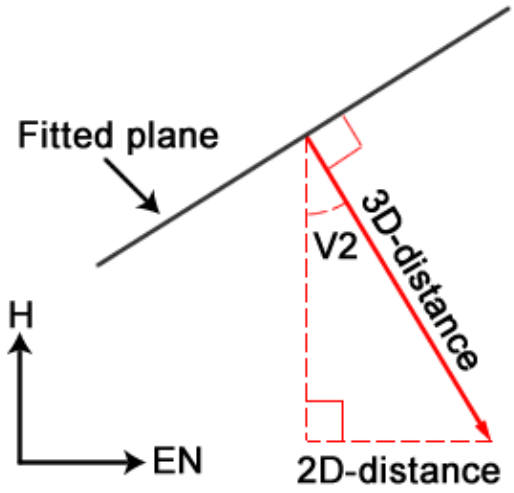


Figure 63 – The normal vector and the steepest slope of the fitted plane

From the figure above, it can be seen that the steepest slope, V2, can be found as:

$$V2 = \sin^{-1}\left(\frac{2Ddistance}{3Ddistance}\right) \tag{5.16}$$

where $2Ddistance$ *is* $\sqrt{{a_1}^2 + {a_2}^2}$

$3Ddistance$ *is the length of the normal vector found as* $\sqrt{{a_1}^2 + {a_2}^2 + (-1)^2}$

Based on the angles V1 and V2, it is possible transform the points into a new coordinate system where the mean slope of the dataset is removed. Before doing this, the E, N and H coordinates should be reduced in order to avoid numerical problems. The points are reduced by extracting the evaluation point's coordinates as described by the following formulas:

$$e_{Fi} = E_i - e_{eval} \qquad n_{Fi} = N_i - n_{eval} \qquad h_{Fi} = H_i - h_{eval} \tag{5.17}$$

where E_i, N_i *and* H_i *are the coordinates of the i'th nearby point*

e_{eval}, n_{eval} *and* h_{eval} *are the coordinates of the evaluation point*

It is decided to reduce the coordinates based on the evaluation point instead of the mean coordinates. This is chosen since it will simplify later calculations of 2D distances and thereby speed up the algorithm. A distance from the evaluation point to any point can now be calculated as $\sqrt{{E_i}^2 + {N_i}^2}$ instead of $\sqrt{(E_i - e\_eval)^2 + (N_i - n\_eval)^2}$.

Next step is to transform the points into a new coordinate system where the mean slope of the dataset is removed. The math involved depends on the used definitions of coordinate system and rotations. Therefore, the used coordinate system and the definition of rotations will be described in the following, before continuing with a description of the transformation of the points.

When talking about rotations and coordinate systems in this project, it is important to note that a right-handed three-dimensional Cartesian coordinate system is used. The coordinate system

consists of three mutually perpendicular axes. The 1st axis (E), the 2nd axis (N) and the 3rd axis (H) all have the same origin. The positive directions of the axes are defined according to the *right-hand rule*. When holding the right hand as shown at Figure 64, the 1st axis is positive along the thumb, the 2nd axis is positive in the direction along the forefinger, and the 3rd axis is pointing in the direction of the middle finger. (SAE International 1994)

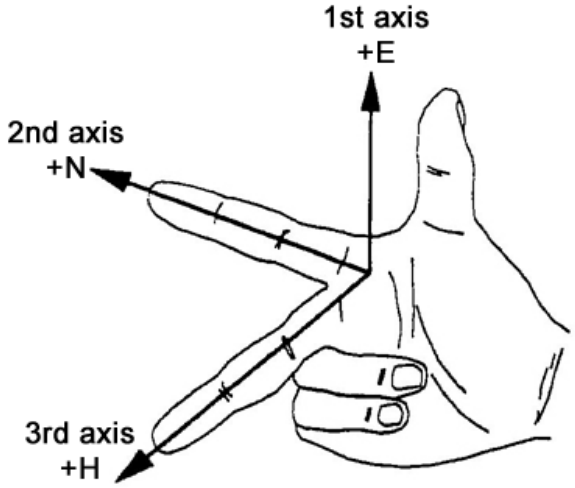


Figure 64 – Definition of axes according to the right-hand rule (SAE International 1994)

The direction of positive angular motion is defined according to the *right-handed screw rule*. When grasping with the right hand at any axis with the thumb pointing in the positive direction of the axis, as shown at Figure 65, then the curl of the other fingers points in the direction of positive angular motion. (SAE International 1994)

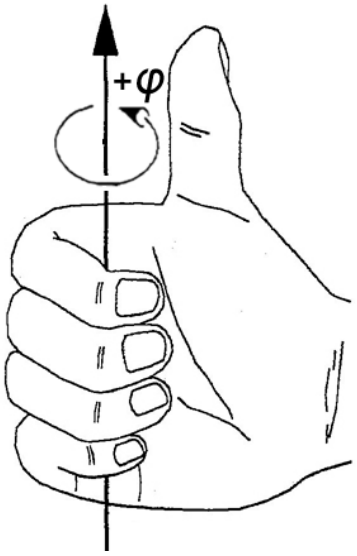


Figure 65 – The direction of positive angular motion, as defined by the right-handed screw rule (SAE International 1994)

A point (e_F, n_F, h_F) can be rotated around the 3rd and 2nd axis by using the following transformation equation (Jensen 2005, 8):

$$\begin{bmatrix} e_T \\ n_T \\ h_T \end{bmatrix} = R_2 \cdot R_3 \cdot \begin{bmatrix} e_F \\ n_F \\ h_F \end{bmatrix} = R \cdot \begin{bmatrix} e_F \\ n_F \\ h_F \end{bmatrix} \tag{5.18}$$

where R_2 *is the rotation matrix for the rotation around the 2nd axis (the N axis)*
R_3 *is the rotation matrix for the rotation around the 3rd axis (the H axis)*
R *is the combined rotation matrix*

The order of the rotation matrices is important, since the rotation should be around the 3rd axis first, and then around the 2nd axis.

The rotation matrices R_2 and R_3 contains the following elements (Jensen 2005, 8):

$$R_2 = \begin{bmatrix} \cos(\varphi_2) & 0 & \sin(\varphi_2) \\ 0 & 1 & 0 \\ -\sin(\varphi_2) & 0 & \cos(\varphi_2) \end{bmatrix}, \qquad R_3 = \begin{bmatrix} \cos(\varphi_3) & -\sin(\varphi_3) & 0 \\ \sin(\varphi_3) & \cos(\varphi_3) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.19}$$

where φ_2 *is the rotation around the 2nd axis in a right-handed coordinate system*
φ_3 *is the rotation around the 3rd axis in a right-handed coordinate system*

The two rotation matrices above can be rewritten into a combined rotation matrix:

$$R = \begin{bmatrix} \cos(\varphi_2) * \cos(\varphi_3) & -\cos(\varphi_2) * \sin(\varphi_3) & \sin(\varphi_2) \\ \sin(\varphi_3) & \cos(\varphi_3) & 0 \\ -\sin(\varphi_2) * \cos(\varphi_3) & \sin(\varphi_2) * \sin(\varphi_3) & \cos(\varphi_2) \end{bmatrix} \tag{5.20}$$

Using the rotation matrix above, it is possible to transform the points into a new coordinate system where the mean slope of the dataset is removed. In order to do this, the points should first be rotated around the 3rd axis according to the angle V1. Afterwards, the dataset should be rotated around the 2nd axis according to the angle V2. From the right-hand screw rule, it can be seen that the rotations in the combined rotation matrix should have the following values:

$$\varphi_2 = V2, \qquad \varphi_3 = -V1 \tag{5.21}$$

Based on the rotation angles above and the transformation equation (Equation 5.18), it is possible to transform all points into a system where the mean slope is removed.

In order to test if the transformation algorithm works as expected, some tests are made using the python script '*DVD:\python\rotation_test\rot_test.py*'. In the script, sample data is created using the plane equation $H(E, N) = 1 + 0.2 \cdot E + 0.4 \cdot N$. The sample data is similar to the sample data without outliers used in section 5.3.1. Below is the test results, showing the rotations performed in the algorithm.

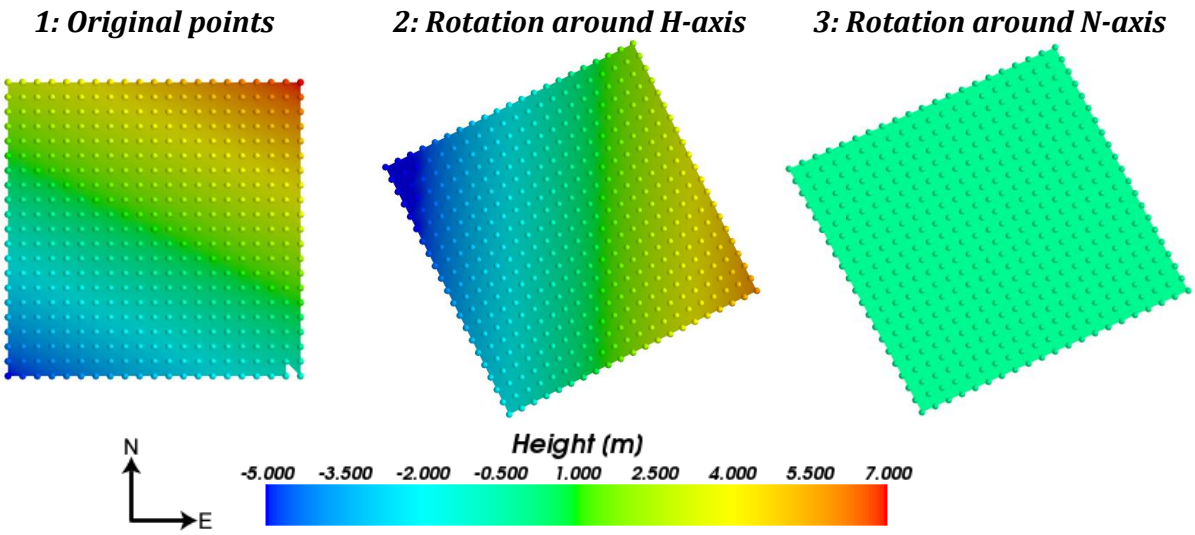


Figure 66 – Test of rotations performed by the algorithm

Figure 66 shows that the algorithm for transformation of the points works as expected, since all sample points has the same height after the transformation. At Figure 66, the rotations are showed as two steps for better understanding of the algorithm, but it should be noted that both rotations are performed in only one step in the program by using the combined rotation matrix.

5.4 Evaluation function

When using the slope-based approach an evaluation function is needed in order to decide if a point is a terrain point or a non-terrain point. In this project it is decided to use a simple function, a straight line:

$$\Delta H_{max} = ax + b \tag{5.22}$$

where ΔH_{max} *is the maximal allowed height difference between two points*
x is the 2D distance between the two points
a is a constant which control the slope of the line
b is a constant which control where the line cross the y-axis

After the slope correction the evaluation point and the nearby points are transformed to a local coordinate system. The evaluation point is placed in (0, 0, 0) which make it simple to calculate the 2D distance to the i'th point in the search area from the coordinates of the i'th point:

$$\text{2D distance} = x_i = \sqrt{e_{t_i}^2 + n_{t_i}^2} \tag{5.23}$$

where e_{t_i} *is the e-coordinate of the i'th point*
n_{t_i} *is the n-coordinate of the i'th point*

The actual height difference between the evaluation point and the i'th point can also be obtained from the i'th point's coordinates. In order to make the evaluation function work

correctly, points with a lower elevation than the evaluation point should give a positive height difference. Therefore the height difference is defined as:

$$\Delta h_i = -h_{t_i} \tag{5.24}$$

where Δh_i *is the actual height difference between the evaluation point and the i'th point*
h_{t_i} *is the h-coordinate of the i'th point*

The height difference and the evaluation function are illustrated in Figure 67. It should be noted that the evaluation function is turned around at the figure, because it is how it works in practice.

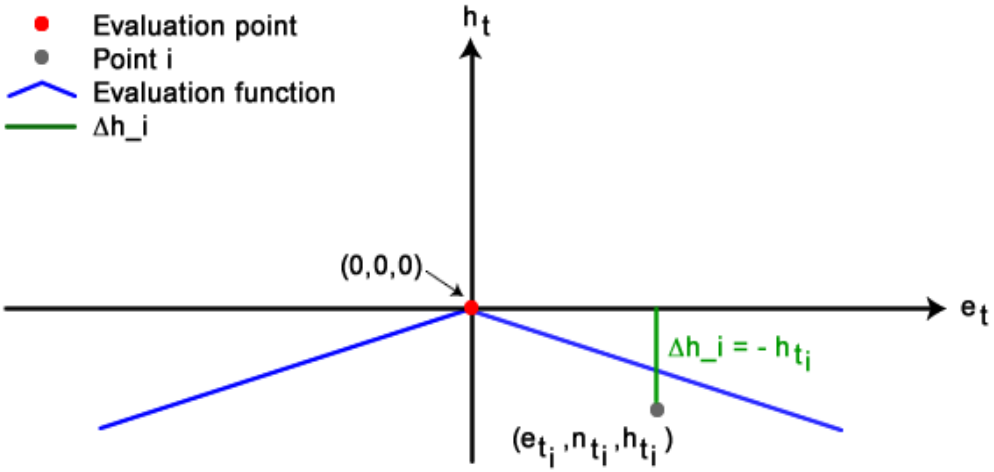


Figure 67 – Illustration of the evaluation function and the actual height difference. In this case the evaluation point is classified as non-terrain.

The maximum allowed height difference is compared to the actual height difference for every nearby point. If just one height difference is above the maximum allowed height difference the evaluation point is classified as non-terrain.

$$\begin{gathered}\text{Terrain point if all nearby points fullfill: } \Delta h_i \leq \Delta H_{max} \\ \text{Non-terrain point if one of the nearby points fulfill: } \Delta h_i > \Delta H_{max}\end{gathered} \tag{5.25}$$

where Δh *is the actual height difference between the evaluation point and the i'th point*
ΔH_{max} *is the maximal allowed height difference between two points*

If the height difference for one point is bigger than ΔH_{max}, then the height differences for the rest of the nearby points are not investigated further. This is time saving since the algorithm can skip some calculations.

The constants in the evaluation function have to be tested to find the best set of constants for filtering away the non-terrain points. It can be discussed if the constant b should be in the equation at all. The argument for a b-constant is that it preserves terrain details a little. The argument against a b-constant is that it will work a little like the buffer used in the previous project by Matthesen and Schmidt (2014).The buffer allowed points on e.g. building walls to

remain in the dataset. The b-constant can do the same, but the effect will of cause depend on the size of b. In all cases b should be a positive number. If b gets negative, then valid terrain points will be removed.

Because the b-constant works a little as the buffer, and the project focus on solving the buffer problem, it is decided to use a b-constant of 0.

With 'b' out of the picture only the constant 'a' should be tested. Five different values for 'a' are decided to be used in the test. Plots of the evaluation function with the different values of 'a' is shown in Figure 68.

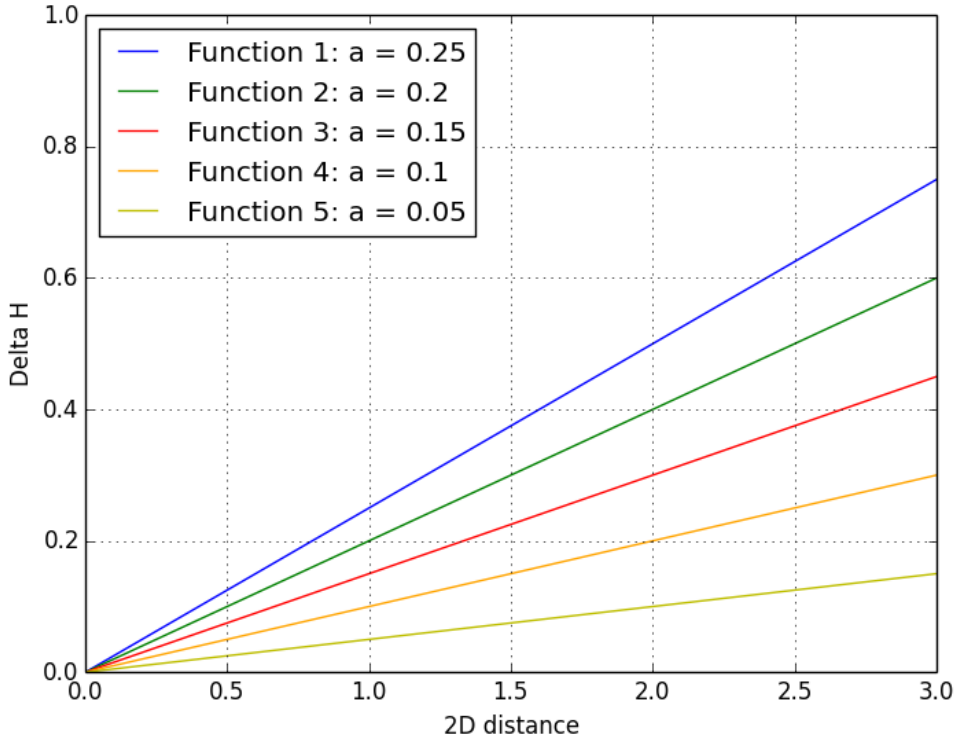


Figure 68 – Evaluation functions which is to be tested

The evaluation functions are tested and the results for subarea A and C are shown in Appendix G. Subarea B is not used here because after step 1 there are no non-terrain points left. The evaluation function should be designed in a way that removes non-terrain points, but at the same time preserves some of the terrain details. When deciding which evaluation function to use it is interesting to look in subarea C. Especially the area marked with a black ellipse on Figure 69. This area contains a slope, which only contains terrain points. This slope should be preserved in the final algorithm. By looking at the plots in Appendix G of the different evaluation functions it can be seen that 'a' must not be smaller than 0.15.

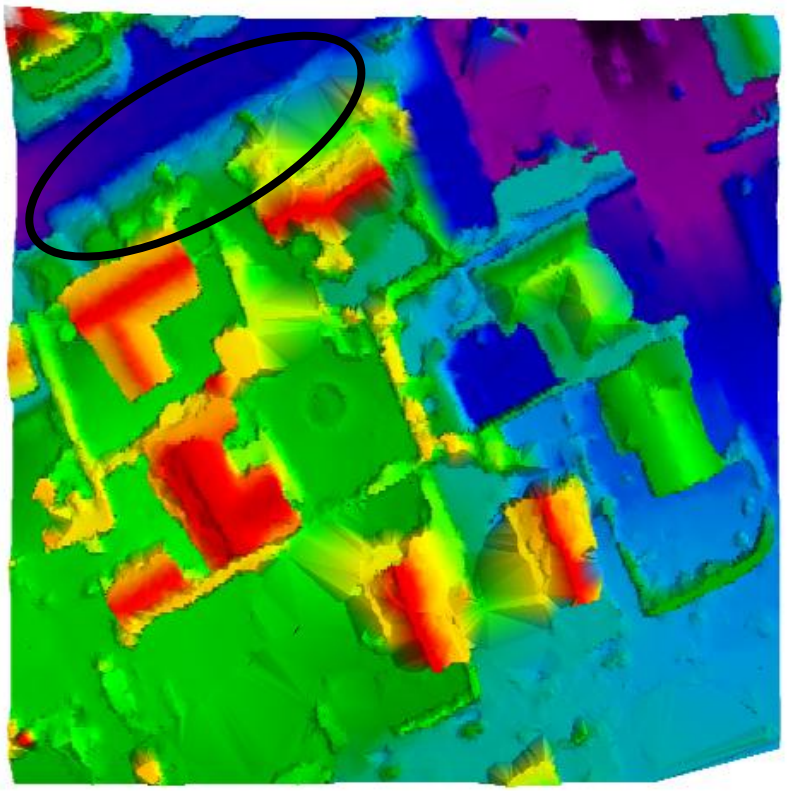


Figure 69 – Subarea C, with an interesting slope marked with the black ellipse

On the other hand the evaluation function should remove the non-terrain points. From the plots in Appendix G it is clear that 'a'-values of 0.25 and 0.20 are too big and some buildings starts to be visible in the point cloud. A value of 0.15 is maybe also a little too big, while the value of 0.10 seems to remove non-terrain points, but at the cost of removing some valid terrain points.

This means that the 'a'-value should be 0.15 to preserve the terrain, but 0.10 to totally remove all non-terrain points. The final choice is a compromise between preserving the terrain and removing the non-terrain points. Therefore the value for 'a' is decided to be between 0.15 and 0.10 and end on 0.13, see function in Figure 70. The results for subarea A and C are shown in Figure 71.

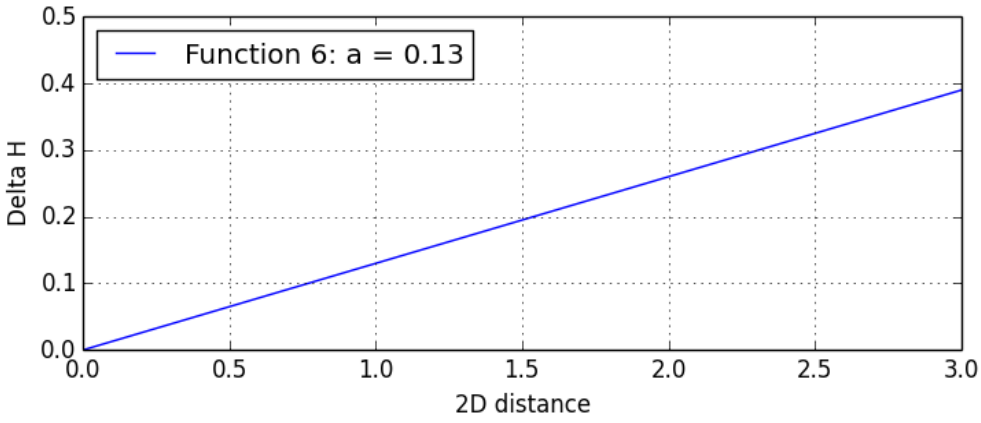


Figure 70 – Final evaluation function

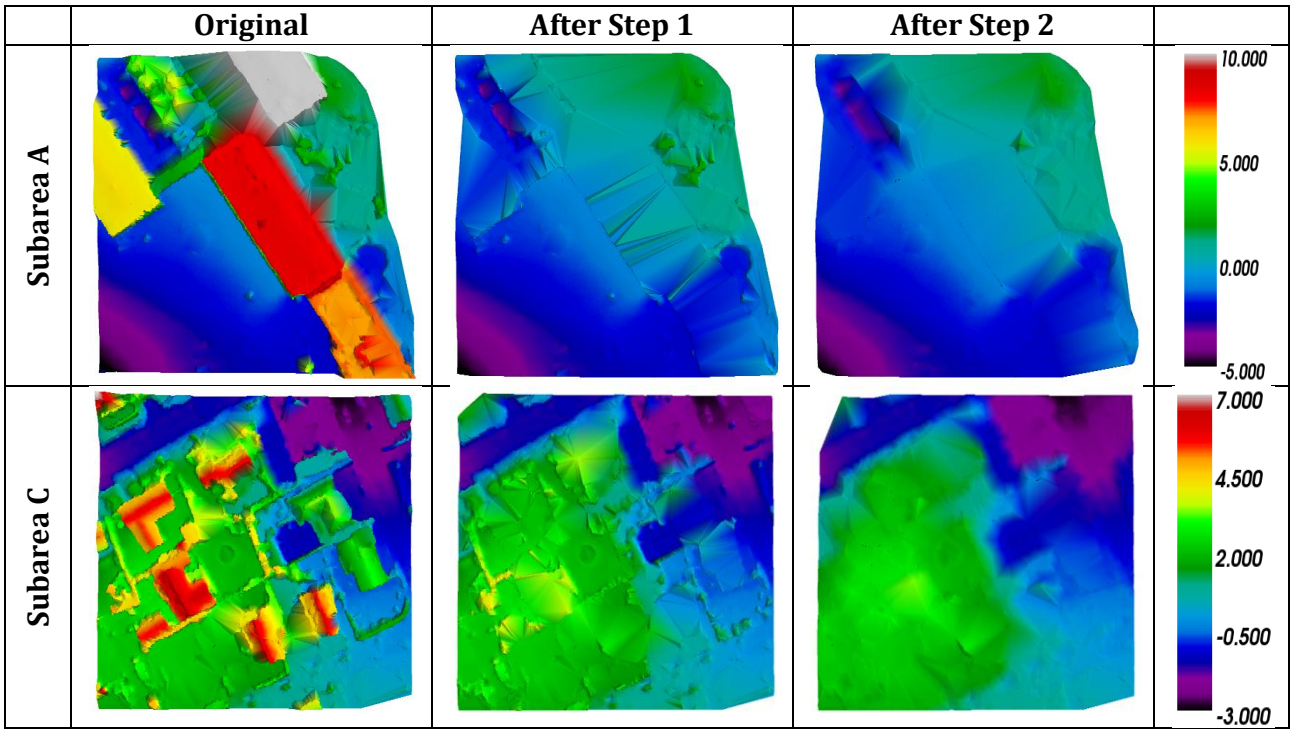


Figure 71- Final results for subarea A and C

5.5 Conclusion

The purpose of this chapter is to develop a slope-based filtering algorithm, and thereby answer the 2nd question from the problem statement:

How can the slope-based approach in step 2 be implemented in a way that removes the non-terrain points and preserve the terrain details?

In a slope-based filtering algorithm, for each point in the point cloud, the neighboring points within a given search radius have to be found. Since the neighboring points has to be found for each point in a n points big dataset, the algorithm has to iterate through the point cloud n^2 times, if no indexing is used. The point cloud used in this current project contains approx. 2 million points. This means that the algorithm has to iterate through the point cloud 4 trillion[3] ($4 * 10^{12}$) times. In other words, the algorithm will be extremely slow if no spatial indexing is used. To overcome this problem, a grid-based spatial indexing class has been written in Cython. A test has been made, which shows that the neighboring points can be found 239 times faster with the developed indexing class, compared to no indexing. This is fast enough for this current project, but it is however assessed that the indexing could be further optimized if needed.

A simple implementation of a slope-based filter works by simply calculating the height difference and the 2D distance to the neighboring points. Afterwards, the height differences are compared to an evaluation function, which defines the maximum allowed height difference as a function of the 2D distance. Such an implementation works well in flat areas. However, with this

[3] Corresponding to the Danish term 'billion'

implementation, valid terrain will be cut away if the terrain is sloped. Since terrain details should be preserved also in sloped terrain, it is chosen not to use the simple implementation.

To overcome the problems in sloped terrain, it is decided to fit a plane to the terrain using the neighboring points. The plane is found as a robust $L_{1.3}$ estimate in order to avoid the influence of remaining non-terrain points. Next, the mean slope can be removed by rotating the points according to the coefficients to the fitted plane. Afterwards, the evaluation function can be applied to the rotated points where the influence of the mean slope has been removed.

For simplicity and speed, it is decided to use a relatively simple evaluation function, which is simply a straight line going though (0, 0). The optimal steepness of the line has been found though a series of tests, and is a compromise between removing the non-terrain points and still preserving the details of the terrain.

Throughout the chapter, a set of parameters has been chosen both regarding indexing and slope-based filtering. Table 18 shows the chosen parameters for Step 2.

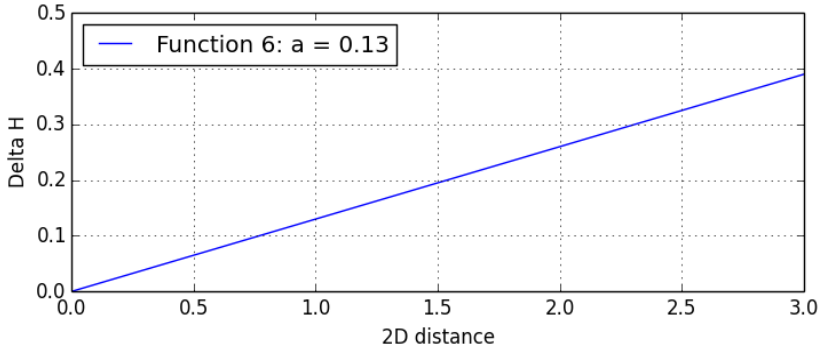
Indexing	
Grid size	14 m
Search area	
Shape	Circle
Radius	3 m
Evaluation function	
Function	$\Delta H_{max} = 0.13 \cdot x$ 

Table 18 – Parameters for Step 2

Chapter 6: DTM Creator 2.0

6.1 Introduction

A graphical user interface (GUI) for the algorithms created in Chapter 4 and Chapter 5 has been created. This chapter will mainly be a presentation of the developed multi-threaded GUI program *DTM Creator 2.0*. Along with the presentation of the program, the actual project dataset will be processed, and examples of different outputs from the program will be showed.

After the project dataset has been processed, some of the outputted documentation files are processed by GIS software in order to generate maps which can reveal if the used algorithms have performed as expected.

6.2 The final GUI program

The GUI program is created in the programming language *Python*, and relies on a number of *extension modules*. Extension modules are basically external libraries which enhance the functionality of native Python. The most important of the used extension modules can be seen in Table 19 below.

Name	**Description**
Cython	Used to speed up Python scripts by compiling them into C extensions.
matplotlib	Used to emulate the 2D plotting facilities in MATLAB.
Mayavi	Used to create advanced 3D plots.
Numpy	Used for matrix calculations etc.
PySide (Qt)	This module is used for GUI programming. The module contains bindings to Qt, which is a cross-platform GUI library originally developed by Nokia.
PyTables	Used to store the point cloud in the binary HDF file format.

Table 19 - Extension modules used in the final program (Matthesen og Schmidt 2014, 118)

In order to run the created program, a python environment with the used extension modules listed in Table 19 is required.

As described in Chapter 3 the algorithm consists of two steps. Step 1 is about removing the big clusters of non-terrain points. Step 2 is about removing the remaining non-terrain points using a slope-based filtering approach. Based on these two overall steps, it is decided to split the final program into four steps, see Figure 72. The numbers 0, 1.1, 1.2 and 2 describing the four steps are chosen in order to ensure that a uniform naming strategy is used throughout the project:

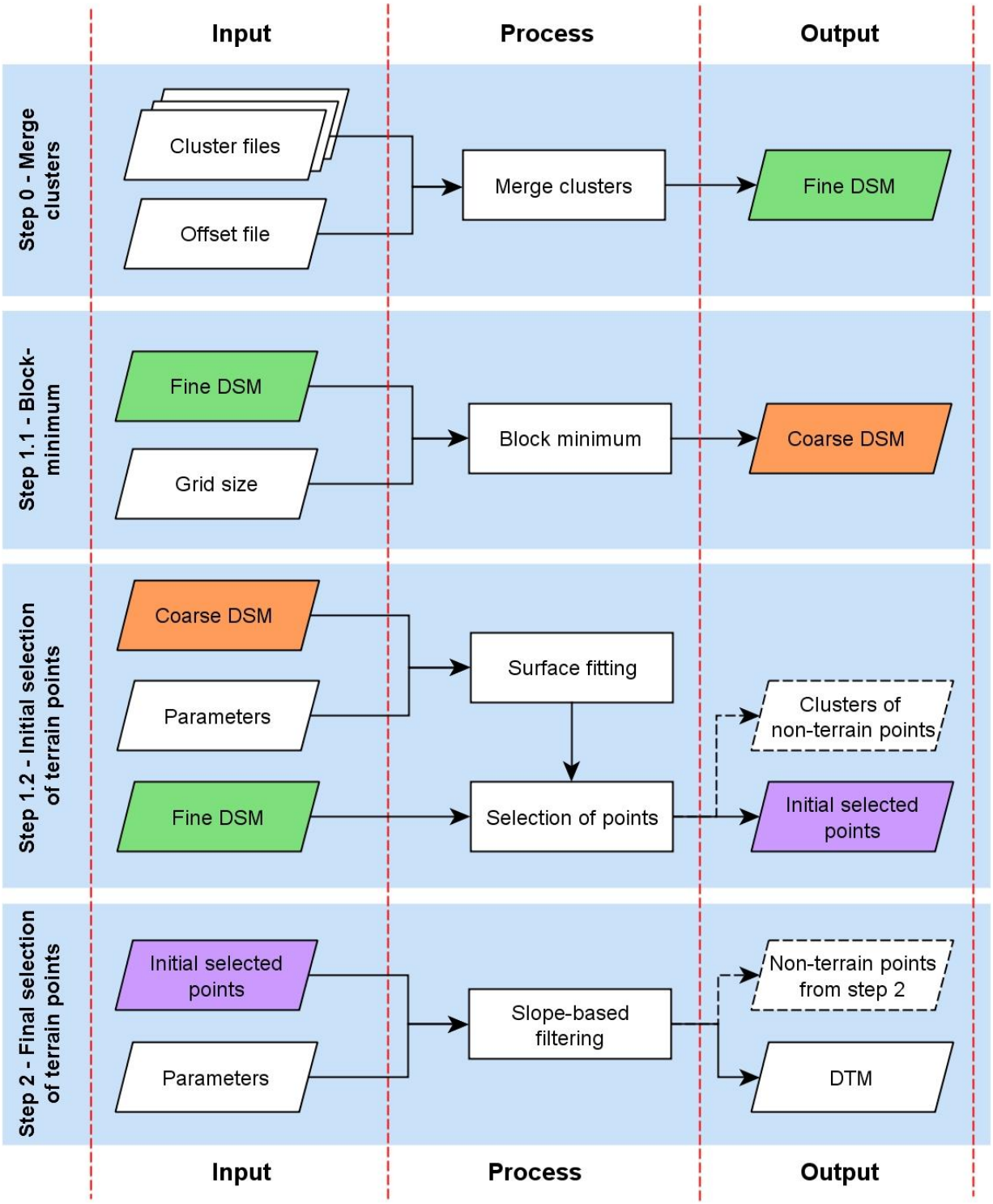


Figure 72 - Steps in the final program. The relations between input/output files from the different steps are marked with colors. For simplicity, the generated documentation files are not showed.

Each step can be run individually and creates output files which can be used by the next step or analyzed for a better understanding of the algorithm. The possibility to analyze the different outputs is one of the main reasons for the choice of splitting the algorithm into several steps. Furthermore, this choice makes it possible to re-run a single step with new parameters without running all the previous steps.

One might wonder why a Step 0 is added to the algorithm. This step is purely a technical step which is converting the multiple point cloud files from the photogrammetry software used in the previous project into a single binary HDF file.

6.2.1 Front page of program

The final program is called *DTM Creator 2.0*. When running the script *'DVD:\python\program\dtm_creator.py'*, the program is started, and the user is presented with the following front page:

Figure 73 – Front page of the final program called DTM Creator 2.0

The first thing the user should do is to check if the current *working directory* is correct. This can be done by simply looking at the title of the program, where the path is displayed. The working directory determines where the output folders and files should be saved. If the working directory needs to be changed, this can be done both from the toolbar and the menu bar, see Figure 74.

By storing the current working directory in a txt-file, the program remembers the working directory even if the program is restarted. If a non-existing path for the working directory is discovered when starting the program, the user is prompted with a folder dialog forcing to select a working directory before the program can start.

The buttons on the toolbar can be used not only to change the working directory, but also to open the current working directory in File Explorer, going to the front page, and closing the program.

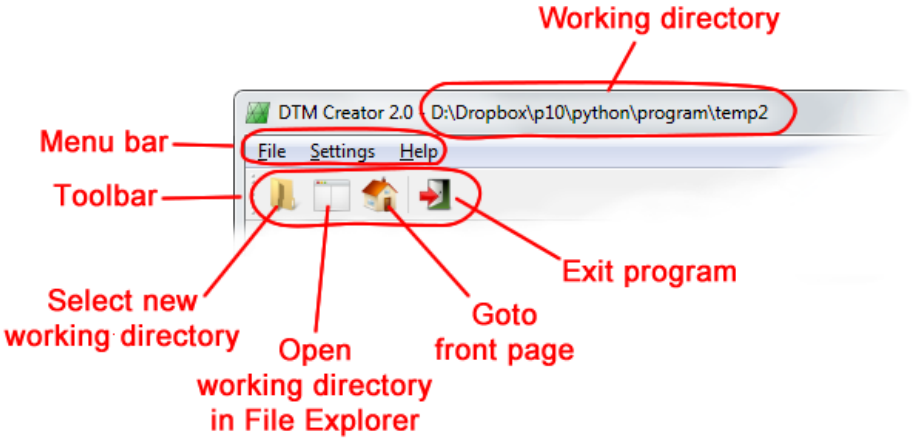


Figure 74 – The basic functions are found in the menu and the toolbar placed in the top of the window.

After the user has ensured that the desired working directory is used, the different steps of the filtering algorithm can be run by clicking at the four command link buttons. It is obviously not possible to for instance run Step 1.1 before Step 0, since the output from Step 0 is needed in Step 1.1. The user should not worry about this, since the program always checks if the output files from the previous step exists. In other words, the program makes sure that the steps are performed in the correct order.

6.2.2 Step 0 – Merge clusters

The first step in the program is performed by clicking at the button '*0 – Merge clusters*' from the front page. After clicking at this button, the following page is displayed:

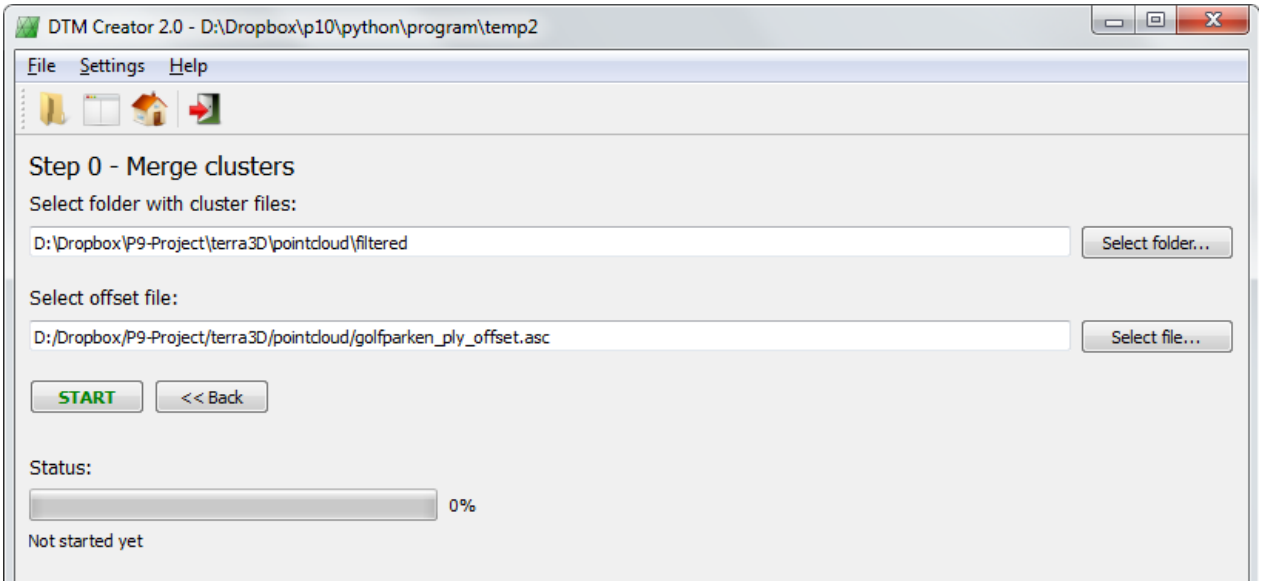


Figure 75 – In Step 0 the cluster files from the photogrammetry program are merged into a single HDF file

The purpose of the 'Merge clusters'-page is purely a matter of conversion of the format of the raw point cloud. In this current project, the raw point cloud used is the so-called filtered point cloud generated by the photogrammetry software used in the previous project by Matthesen and Schmidt (2014).

The raw point cloud generated by the photogrammetry software used in the previous project can be found in the folder *'DVD:\pointcloud\filtered\'* as multiple ASCII files. Each ASCII file contains a cluster of the raw point cloud. These cluster files should be merged into a single binary HDF file in order to make it easier to use and analyze the raw point cloud. Therefore, the user has to select the folder where the cluster-files are placed.

The user also has to select an offset-file, which can be found at '*DVD:\pointcloud\golfparken_ply_offset.asc*'. This file contains the offset values which the photogrammetry software used to reduce the E, N and H coordinates in the cluster files. The points in the cluster files are corrected according to this offset before they are saved in the HDF file.

When the user presses the 'START'-button, a new thread will be started where the selected cluster files will be merged into a single HDF file. The progress of the merging process can be followed in the green progress bar. The user is notified with a pop-up window when the merging process in completed, see Figure 76.

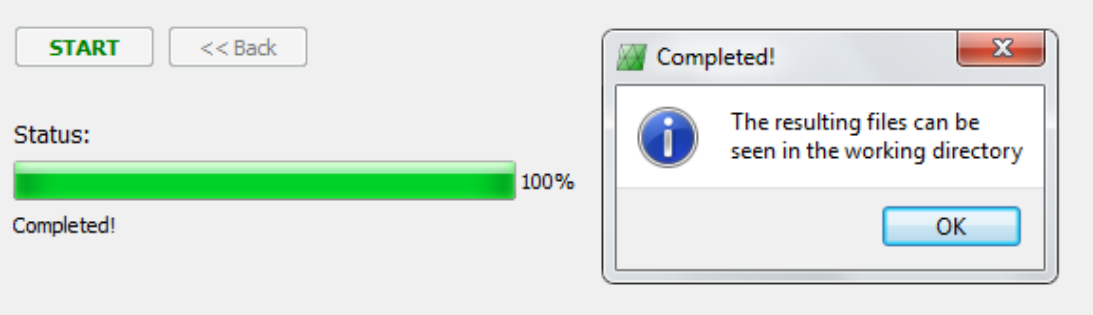


Figure 76 - The user is continuously informed about the progress of the merging process

After the merging is completed, the working directory contains a new folder with the HDF file:

```
\---STEP 0 - Merge clusters              (new folder)
        point_cloud.h5                   (fine DSM)
```

The '<< Back'-button and the home button in the toolbar, are disabled while the merging is performed. When the merging is complete, these buttons are reactivated and can be used to go back to the front page. Back at the front page, a new button will appear which makes it possible to view the raw point cloud in a 3D viewer. The 3D viewer will be explained in section 6.2.6.

Figure 77 below shows an example of a part of the outputted raw point cloud from Step 0. The example is generated by plotting a part of the HDF file in Mayavi using the Python script '*DVD:\python\program\temp2\STEP 0 - Merge clusters\step0_pointcloud_viewer.py*'.

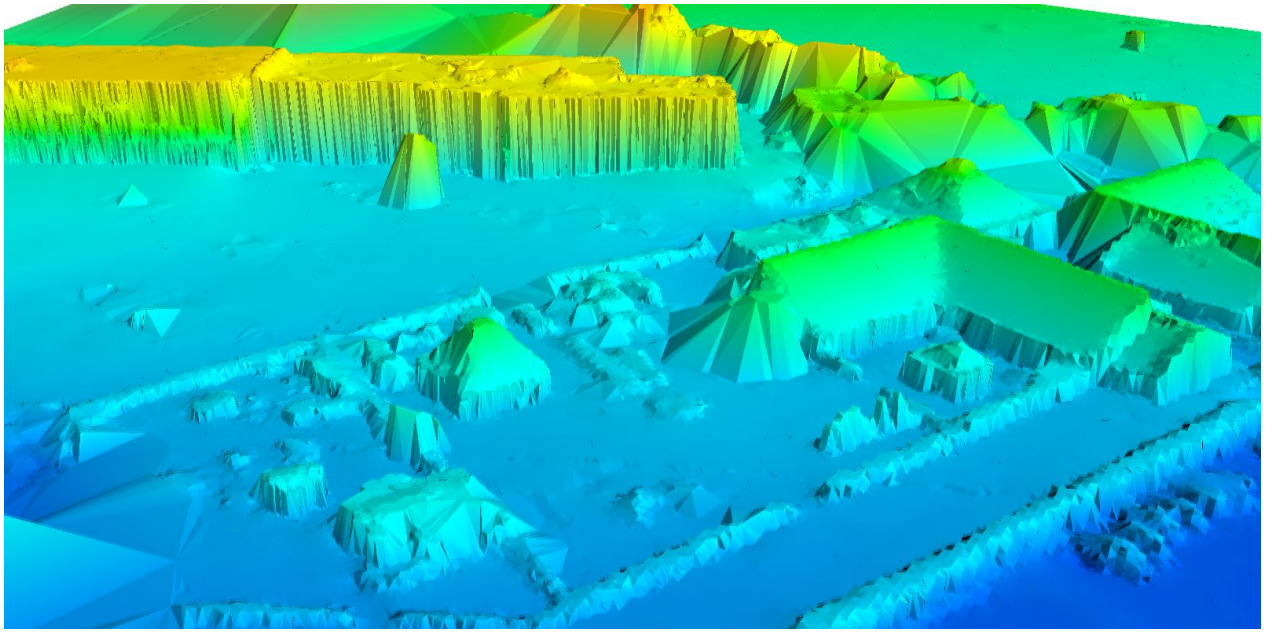


Figure 77 - Example of a part of the outputted raw point cloud from Step 0

6.2.3 Step 1.1 – Block-minimum filtering

The next step in the program can be performed by clicking at the button '*1.1 – Block-minimum*' from the front page. After clicking at this button, the following page is displayed:

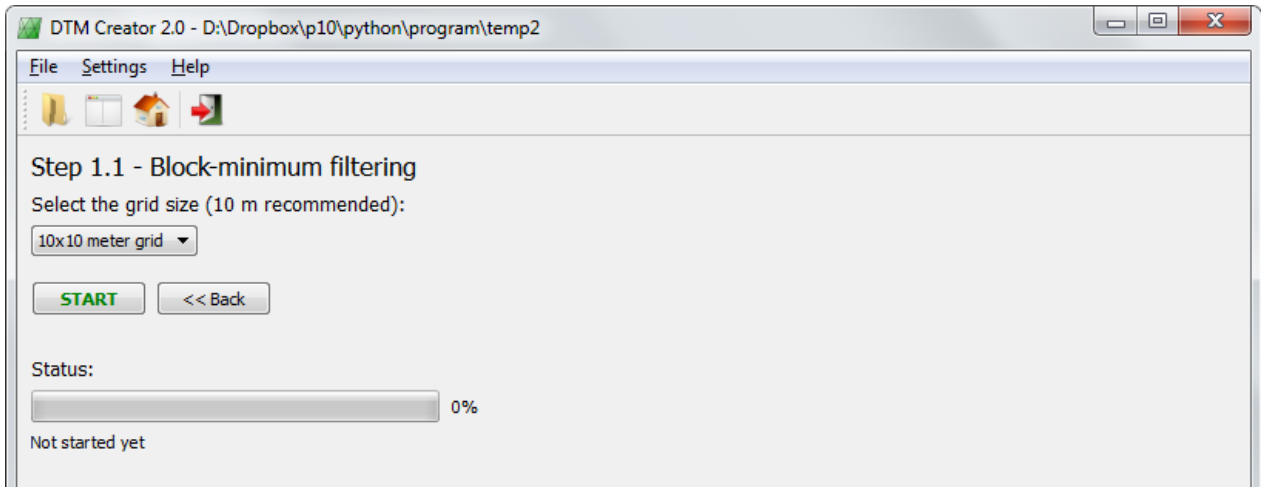


Figure 78 – In Step 1.1 a block-minimum filter is applied to the raw point cloud from Step 0

From the displayed block-minimum page it is possible to choose the grid size which should be used for the block-minimum filter. The recommended grid size of 10 meters is used in this project, but it is possible to select different grid sizes from the interface.

By pressing the 'START'-button, the block-minimum filter is applied to the raw point cloud from Step 0. After the block-minimum filter has completed, an ASCII file with the lowest point from each cell is created. This means that the working directory now contains the following folders and files (new files are described in a parenthesis):

```
+---STEP 0 - Merge clusters
|       point_cloud.h5
|
\---STEP 1.1 - Block-minimum          (new folder)
        block_min.txt                 (coarse DSM)
```

Figure 79 below shows a plot of a part of the output from the block-minimum filter. The plot is comparable with the plot used to visualize the output from Step 0, since the same camera position and orientation is used for both plots. The plot from Step 1.1 is generated by plotting a part of the outputted ASCII file in Mayavi using the Python script '*DVD:\python\program\temp2\STEP 1.1 - Block-minimum\step1_1_pointcloud_viewer.py*'.

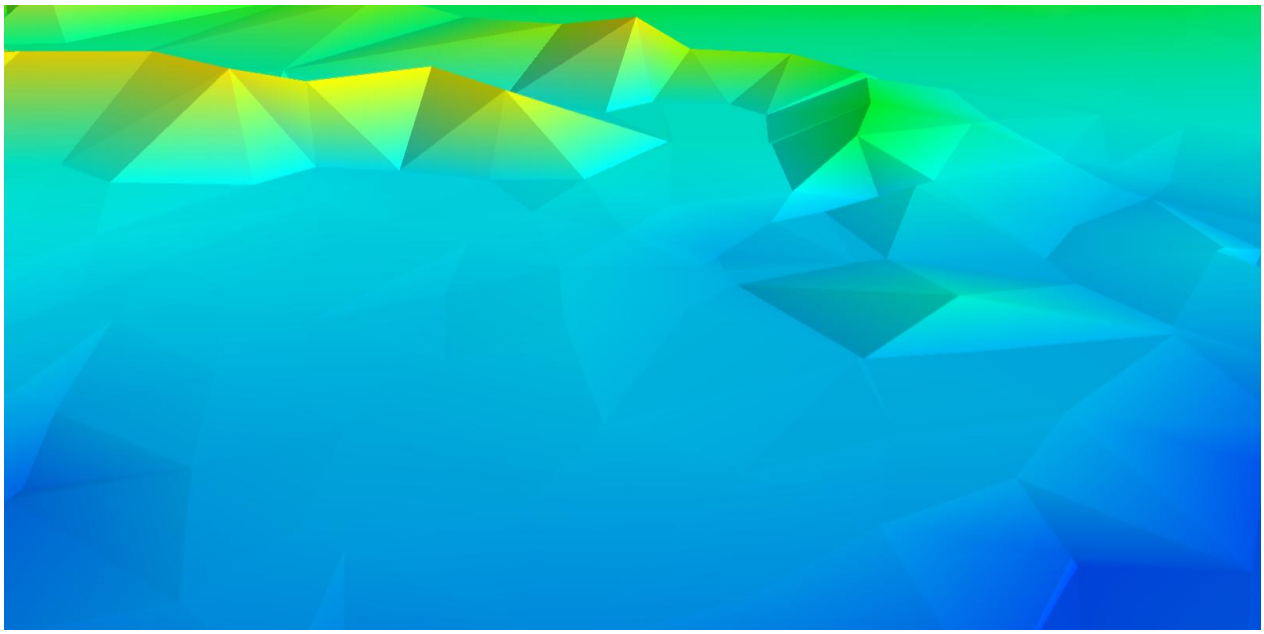


Figure 79 - Example of the block-minimum result from Step 1.1

6.2.4 Step 1.2 – Initial selection of terrain points

The next step in the program can be performed by clicking at the button '*1.2 – Initial selection of terrain points*' from the front page. After clicking at this button, the following page is displayed:

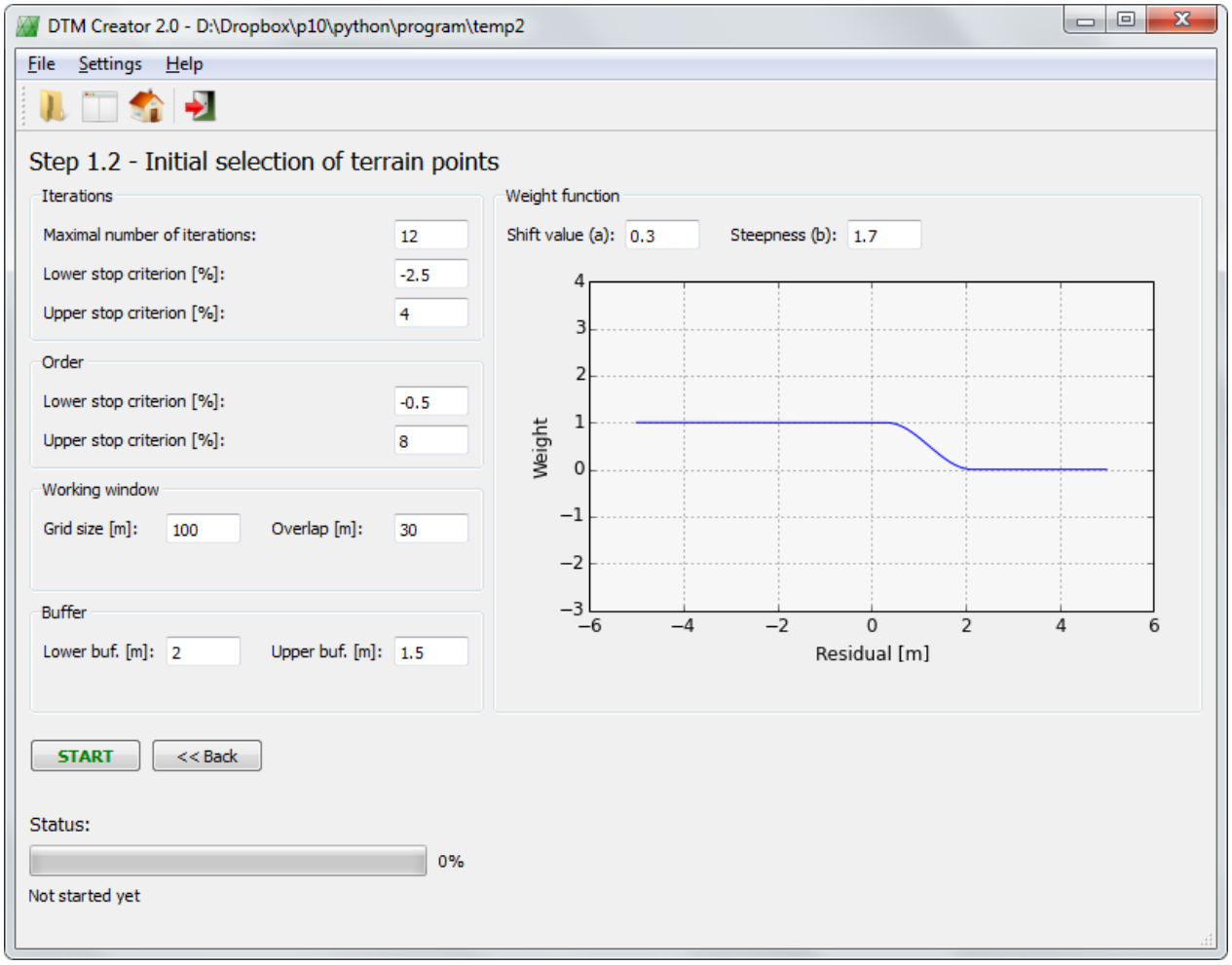


Figure 80 – In Step 1.2 an initial selection of terrain points is performed

From the page, it is possible to select the parameters which should be used for the surface-based filtering algorithm used to remove the big clusters of non-terrain points. If the parameters regarding the used weight function are changed, the plot of the weight function will automatically update itself.

By default, the recommended parameters are used for Step 1.2. In this project, the default parameters will be used, and the user should just press the 'START'-button to start the surface-based filtering algorithm.

After the initial selection of terrain points has completed, the working directory contains the following folders and files (new files are described in a parenthesis):

```
+---STEP 0 - Merge clusters
|       point_cloud.h5
|
+---STEP 1.1 - Block-minimum
|       block_min.txt
|
\---STEP 1.2 - Initial selection                (new folder)
        initial_selection.h5                    (initial selection)
        initial_selection_non_terrain.h5        (non-terrain points)
        report.txt                              (documentation file)
        results.txt                             (file for GIS)
```

In the end of the documentation file for Step 1.2, some overall statistics for the initial selection is calculated:

```
Overall statistics (for used squares only):
  no. of squares:                 176
  skipped squares:                24
  percent skipped sq.:            13.6 %
  mean std. dev. of unit weight: 0.300
  total points before:            2906537
  total points inside buffer:     2190369
  total points removed:           716168
  percent points removed:         24.6 %
  processing time:                0:04:56
```

Figure 81 below shows a plot of a part of the initially selected points. The plot is comparable with the plots used to visualize the output from Step 0 and Step 1.1, since the same camera position and orientation is used for all plots. The plot is generated using the Python script *'DVD:\python\program\temp2\STEP 1.2 - Initial selection\step1_2_pointcloud_viewer.py'*.

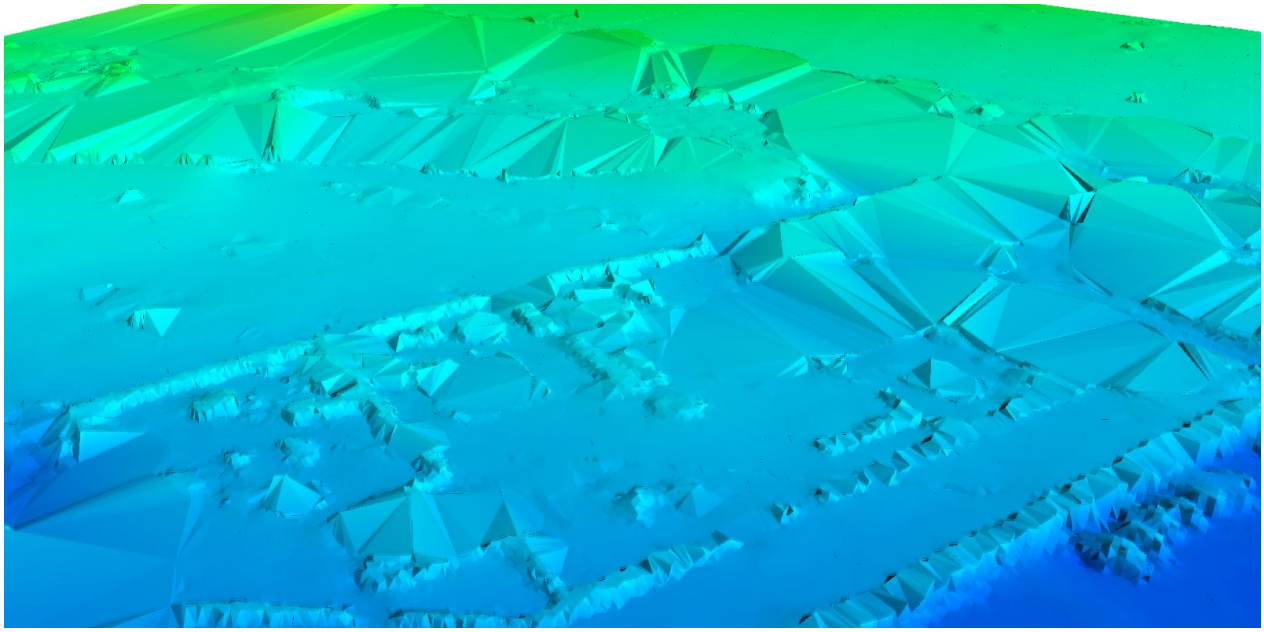


Figure 81 - Example of the initially selected points from Step 1.2

6.2.5 Step 2 – Final selection of terrain points

The next step in the program can be performed by clicking at the button '*1.2 – Final selection of terrain points*' from the front page. After clicking at this button, the following page is displayed:

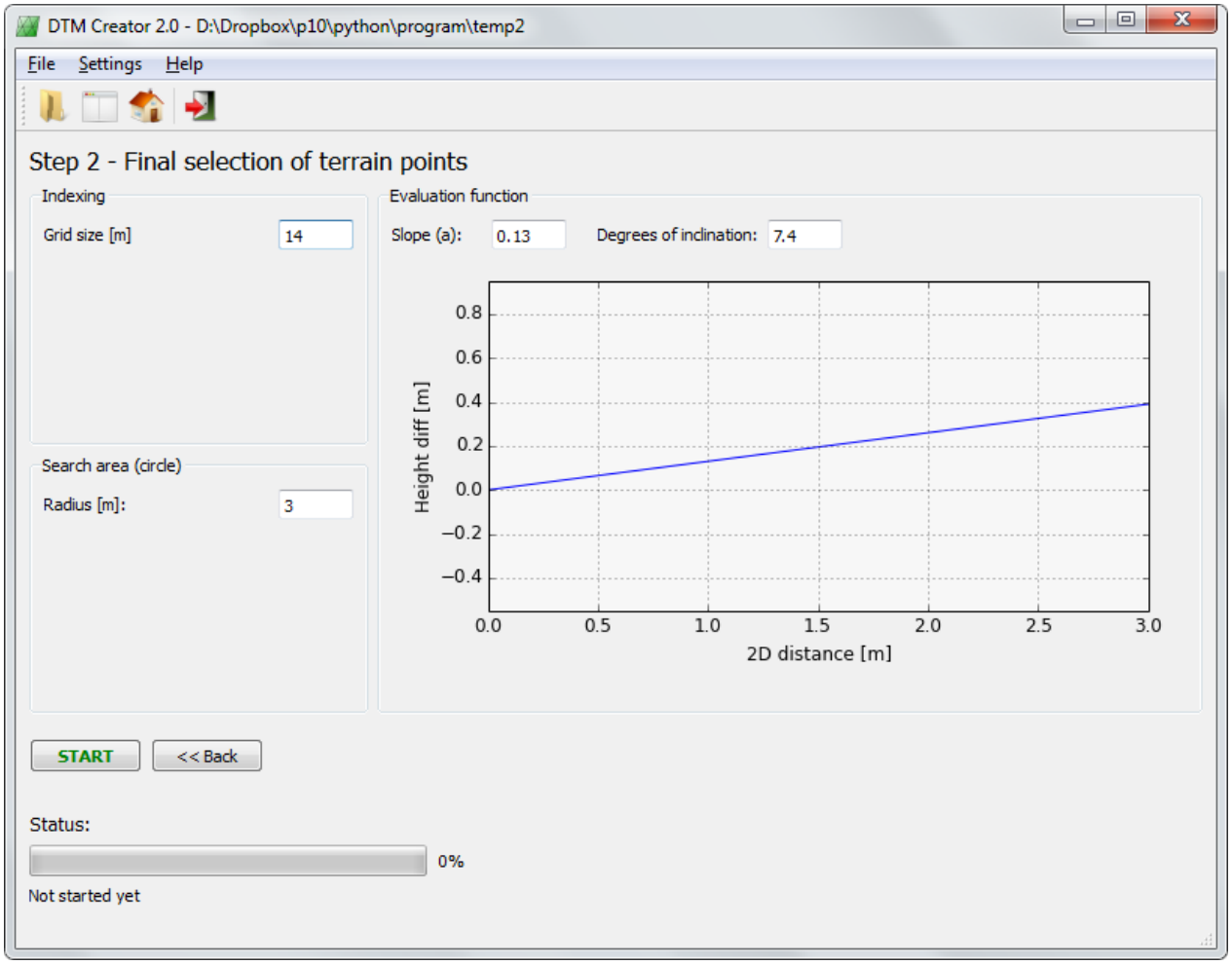


Figure 82 – In Step 2 the final DTM is created using slope-based filtering

The recommended parameters for the slope-based filtering are already set, but they can be changed if desired. The plot of the evaluation function will update itself if a different slope-parameter (a) is inserted. Furthermore, the slope-parameter is automatically converted into the corresponding degrees of inclination for easier understanding of the used slope-value.

In this project, the default values are used, which means that the user should just press the 'START'-button in order to start the final classification of terrain points.

After the final selection of terrain points has completed, the working directory contains the following folders and files:

```
+---STEP 0 - Merge clusters
|       point_cloud.h5
|
+---STEP 1.1 - Block-minimum
|       block_min.txt
|
+---STEP 1.2 - Initial selection
|       initial_selection.h5
|       initial_selection_non_terrain.h5
|       report.txt
|       results.txt
|
\---STEP 2 - Final selection                    (new folder)
        final_selection.h5                      (DTM)
        final_selection_non_terrain.h5          (non-terrain points)
        report_final_selection.txt              (documentation file)
        results_for_GIS.txt                     (file for GIS)
```

In the end of the documentation file for Step 2, some statistics for the final selection is calculated:

```
Slope-based filtering results:
  total points before:                2190369
  total terrain points:               1554927
  total non-terrain points:            635442
  percent non-terrain points removed:  29.0 %
  processing time:                    1:47:14
```

Figure 83 below shows a plot of a part of the generated DTM. The plot is comparable with the plots used to visualize the output from Step 0, Step 1.1 and Step 1.2, since the same camera position and orientation is used for all plots. The plot is generated using the Python script *'DVD:\python\program\temp2\STEP 2 - Final selection\step2_pointcloud_viewer.py'*.

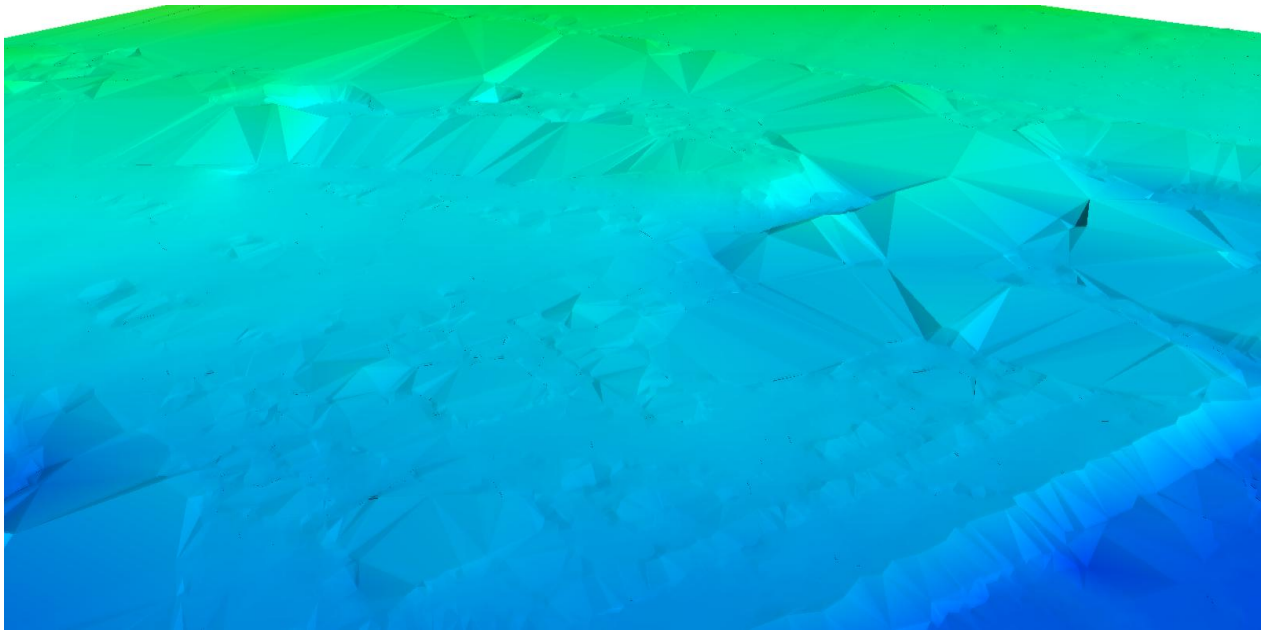


Figure 83 - Example of the DTM points from Step 2

6.2.6 Point cloud viewer

As the different steps are completed, new buttons will gradually appear at the front page of the program, see Figure 84. By clicking at these buttons it is possible to see the generated point clouds from the different steps in a 3D viewer.

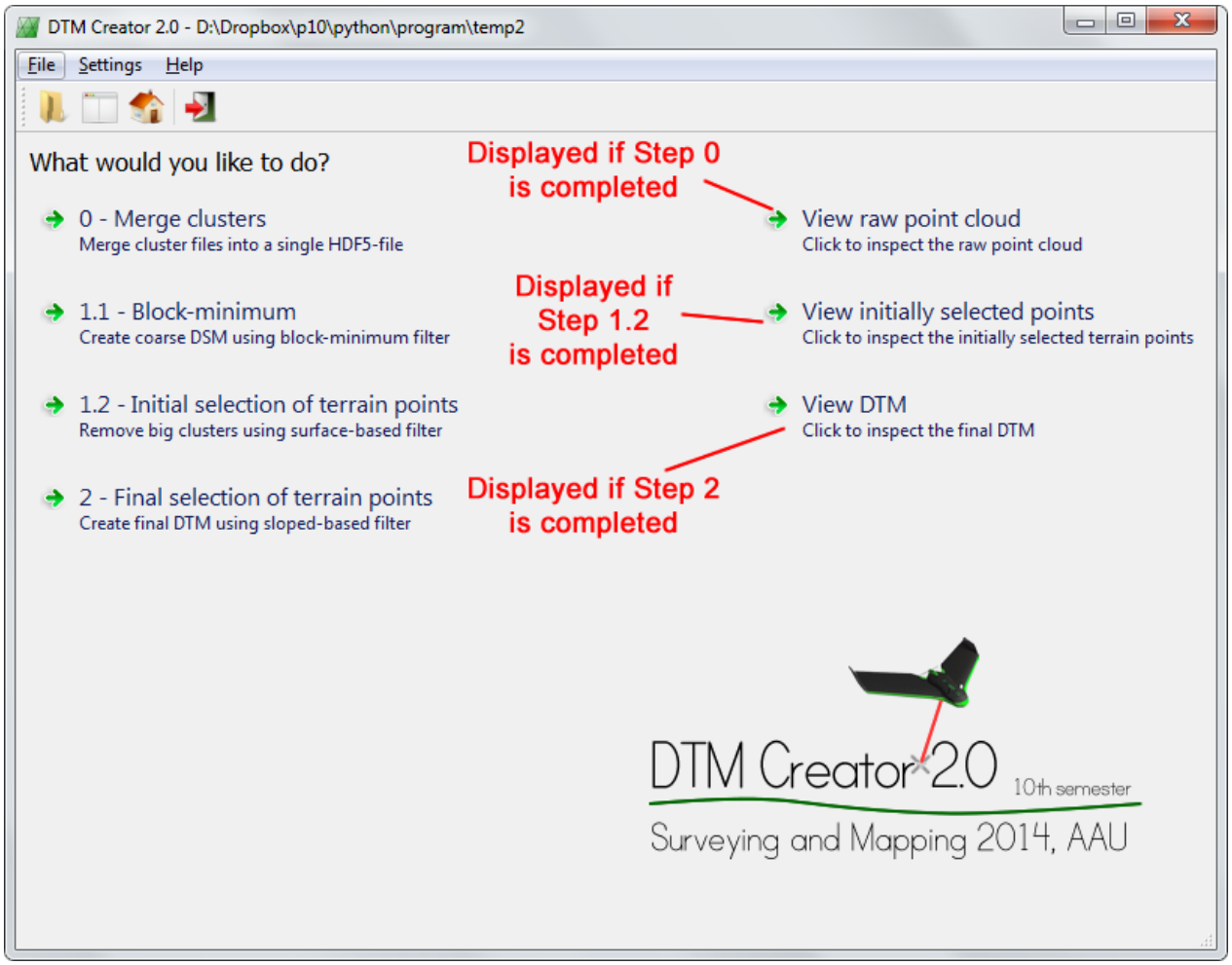


Figure 84 – Links to view the generated point clouds in a 3D viewer gradually appears at the front page

When clicking at one of the three new buttons, a point cloud viewer will be opened. The point cloud viewer will show the raw point cloud, the initially selected points or the final DTM, depending on which of the three buttons the user pressed.

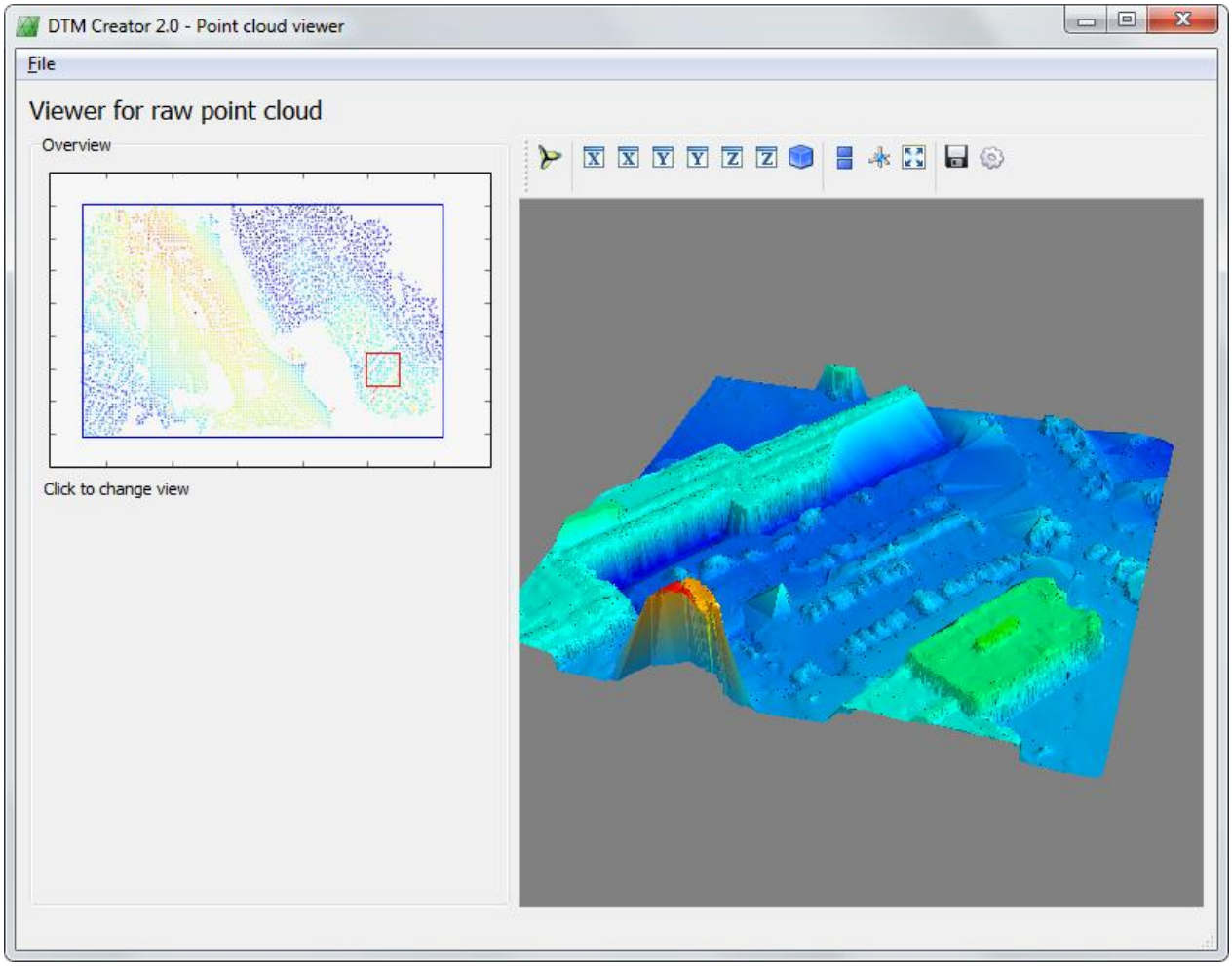


Figure 85 - Point cloud viewer for the raw point cloud

Figure 85 shows the point cloud viewer for the raw point cloud. In the window's right plot, a 100x100 meter square of the point cloud is displayed using the Mayavi extension module. It is possible to freely rotate, zoom and move the plot around. Only a 100x100 meter square is displayed since the Mayavi plot will run out of memory if the entire point cloud is showed on an ordinary laptop with only a limited amount of RAM available.

The left plot in the window is an overview map created using the matplotlib extension module. On the overview map, the bounding box of the point cloud is drawn as a blue rectangle. The current 100x100 m view is marked with a red square. If the block-minimum step is completed, the block-minimum points are plotted as the background of the overview map in order to make the navigation easier.

The current 100x100 m view can be changed by clicking at a new location at the overview map. After selecting a new 100x100 m view, it should be mentioned that the user should have a little patience until a new Mayavi plot has been generated.

6.3 Output – GIS

The final program outputs some result files which can be used to visualize some of the results in a program like ArcMap. The result files are txt-files, but before they can be used they should be converted to shapefiles. The process is shown in Figure 86.

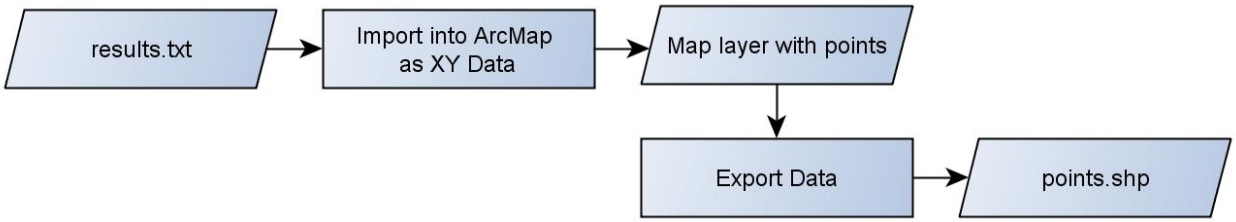


Figure 86 – Flowchart: Text file into point shapefile

Before turning to the results, Figure 87 shows a picture of the project area with some regions and their characteristics.

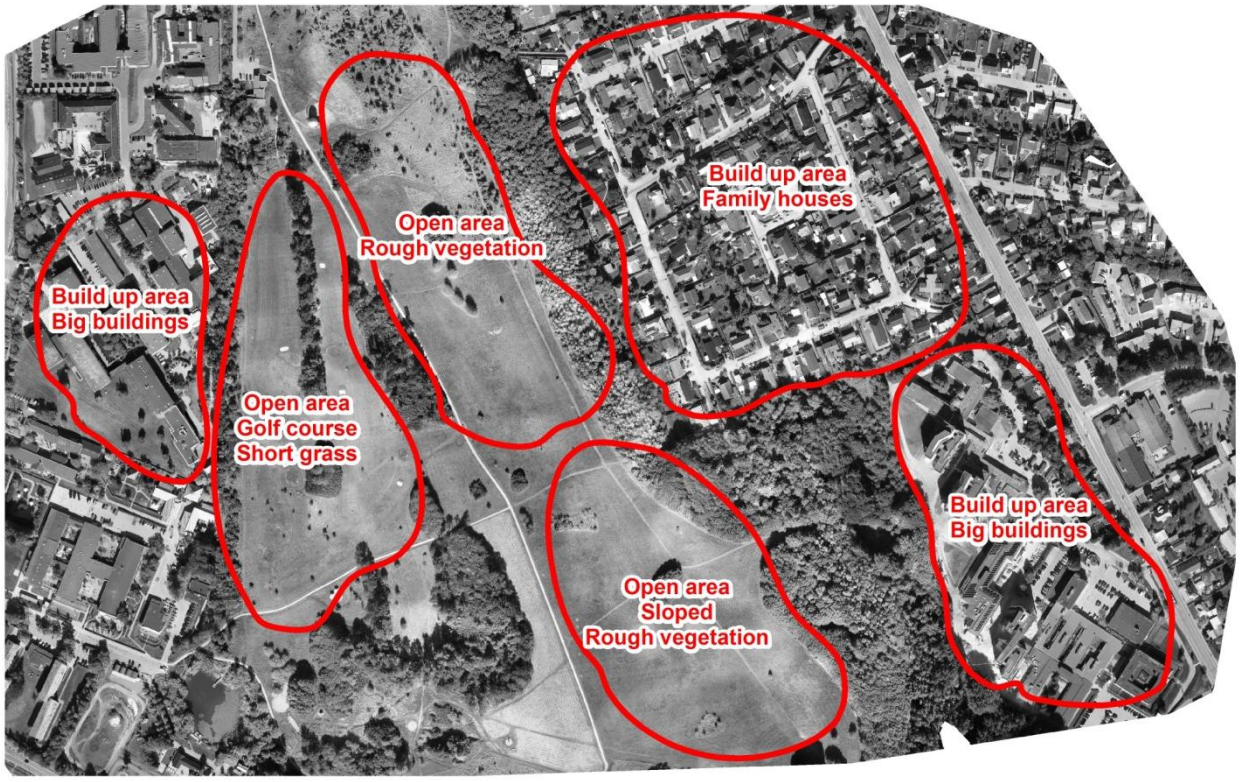


Figure 87 – Regions and their characteristics in the test area

6.3.1 Step 1

In step 1 the polynomial order and the number of iterations are determined automatically. The final program outputs a result file, where information about the surface-based filtering is linked to the center coordinates of the squares. The information is:

- The maximum allowed order
- The actual chosen order
- The number of iterations
- How many points that are removed from the square

Before the data can be used for visualization of the different information, the point shapefile has to be converted into a shapefile containing square polygons. For this purpose a tool is built with the ModelBuilder function in ArcMap, see Figure 88. The tool is called *"point to square"* and can be found at *'DVD:\arcmap_toolbox\point_to_square.tbx'*.

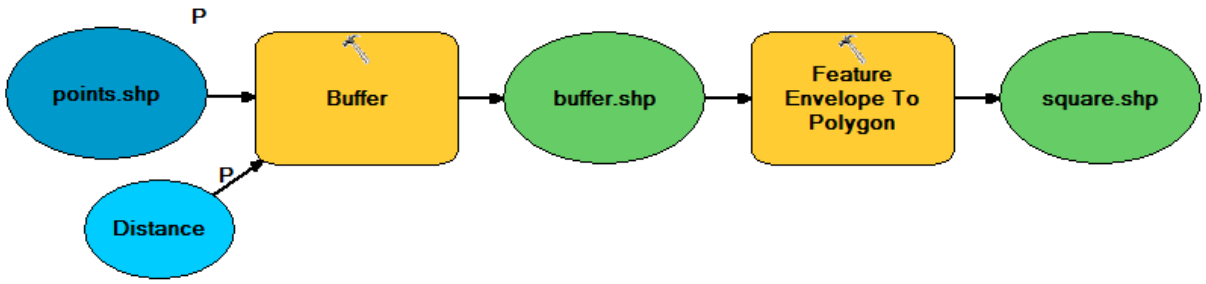


Figure 88 – ModelBuilder for the tool "point to square" *used to convert the point feature into a square polygon feature*

The tool consists of two GIS-analyses. The first is the *"buffer"*, which is a function that creates polygons around the input points according to a specified buffer distance. The next analyze is the *"Feature Envelope To Polygon"*, which takes the input data and makes square polygons around the features. The process in the tool can also be illustrated with Figure 89.

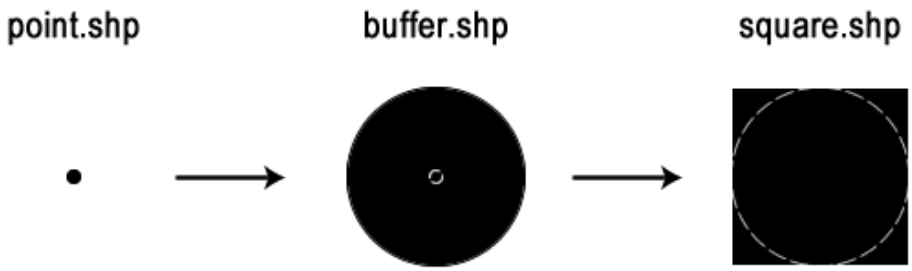


Figure 89 – Process in the tool "point to square". *Black is the feature in each file and the dashed white line shows the feature from the previous file.*

In the following sections some squares do not have any data for order, iterations and number of removed points because the square was skipped in the surface fitting. This is because the number of points in the block minimum point cloud inside the square was too small.

6.3.1.1 Order

The maximal allowed polynomial order depends on the number of points inside each square. The maximal order is shown in Figure 90. I can be seen that in general, build up areas and the golf course where the grass is short are the places where the highest order is allowed. This means that the point cloud has the biggest point density inside these areas. The high point density can be explained from a photogrammetrical perspective, since build up areas and short grass areas apparently have a better texture than areas with more rough vegetation, and therefore contain more points.

Figure 91 shows the actual chosen polynomial order. Here the picture is more mixed, but a little systematic can however be seen. On the golf course a low order is in general used in contrast to the sloped area, where the polynomial order in general is higher, in order to describe the sloped terrain. In the build up areas, the chosen order is more varying, which makes sense since the terrain in these areas is quite complex because it is highly affected by humans (gardening design, levelled roads, earthmoving, etc.).

6.3.1.2 Iterations

The number of iterations is also interesting to visualize on the squares, see Figure 92. It can be seen that in general fewer iterations are used in the open areas in the middle of the project area. This makes sense since there are no or few non-terrain points. It can also be seen that in all but 4 squares the iterations stops after maximal 8 iterations. The 4 last squares continue to the 12th and maximal allowed number of iterations. In all 4 squares there are a quite big area with no points at all, which might be the reason why the polynomial has trouble fitting the points.

6.3.1.3 Removed points

It is interesting to look at the number of removed points in each square. Figure 93 shows how many percent of the original number of points in each square, which is removed and marked as non-terrain. Here there is also a clear systematic. In the open areas none or few points are removed, while in the build-up areas it is between ¼ and ½ of the points which are removed. In some squares it is even more than half of the points which are removed.

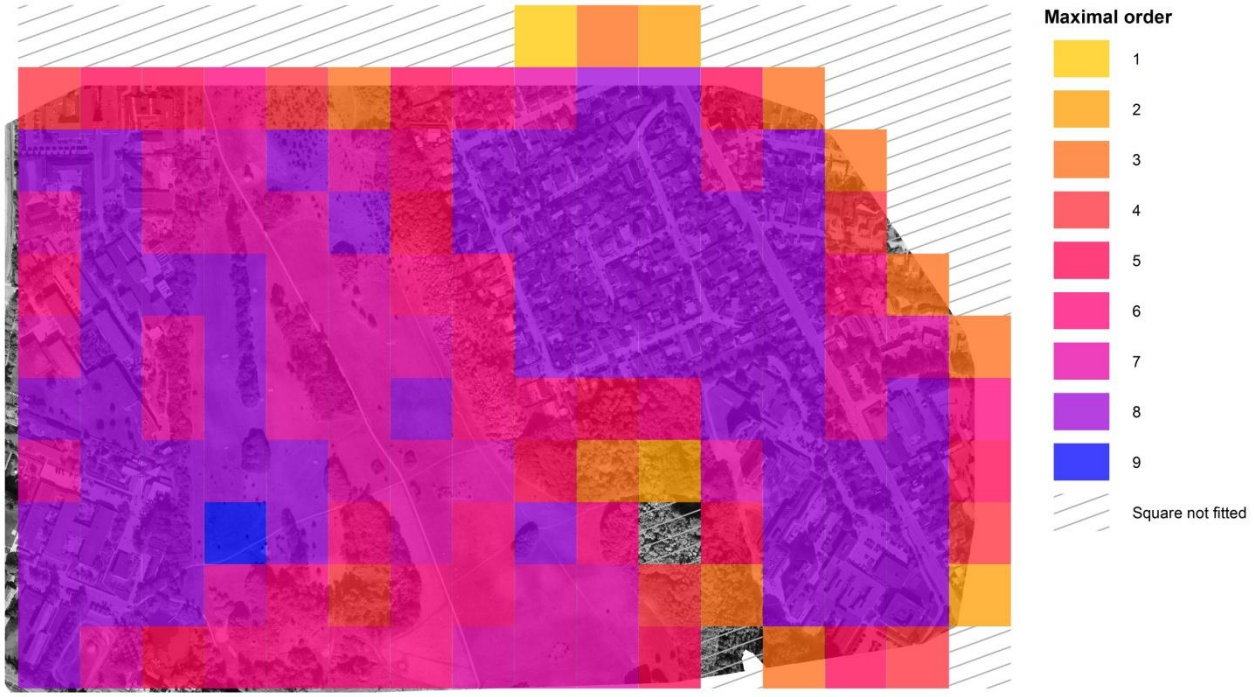


Figure 90 – Maximal allowed order in each square. Orthophoto can be seen behind.

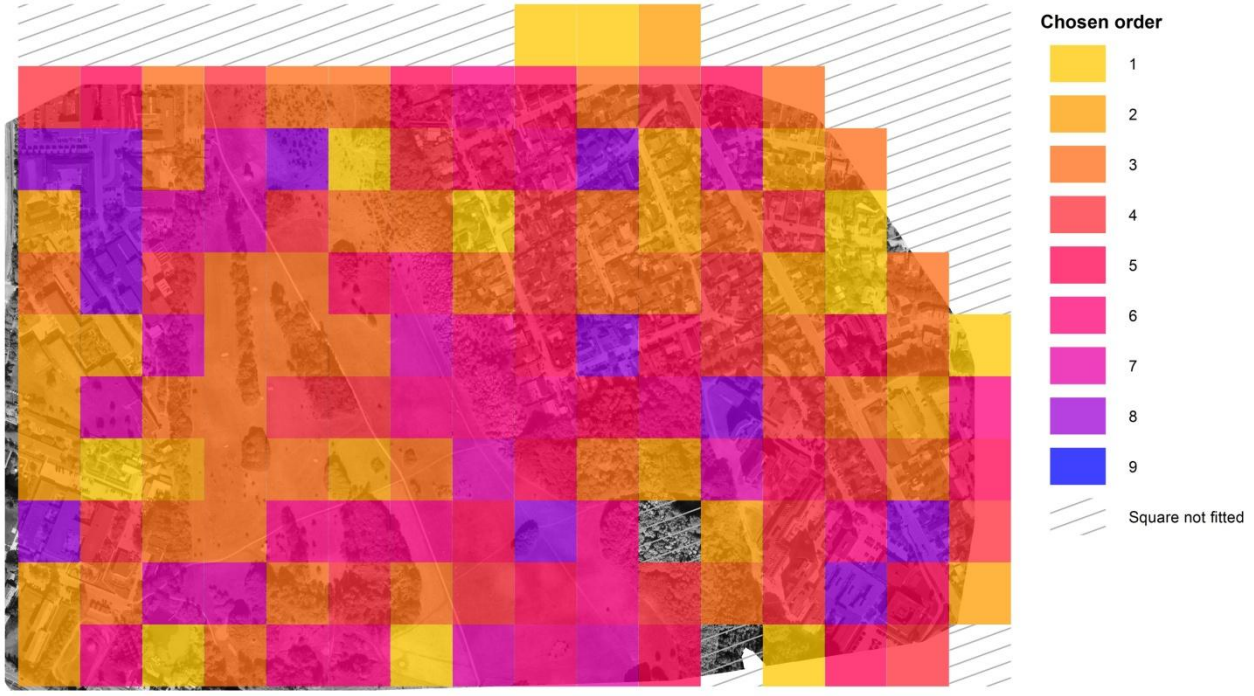


Figure 91 – Chosen order for each square. Orthophoto can be seen behind.

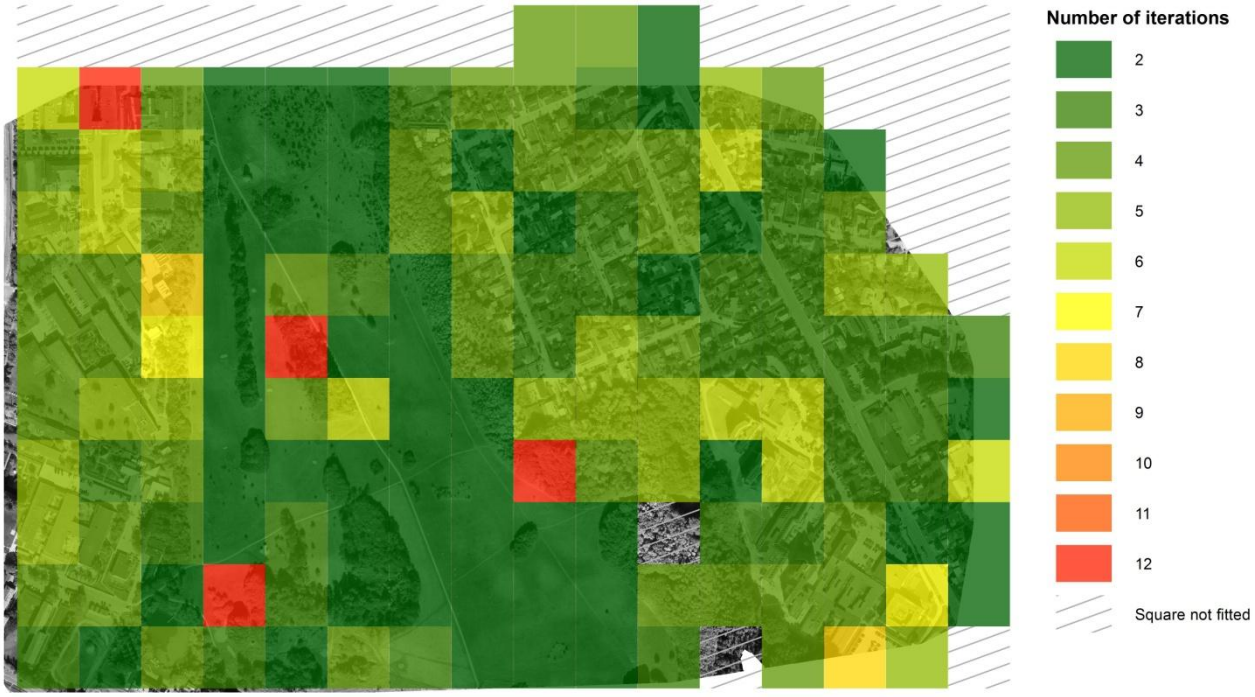


Figure 92 – Number of iterations in each square with the chosen polynomial order. Orthophoto can be seen behind.

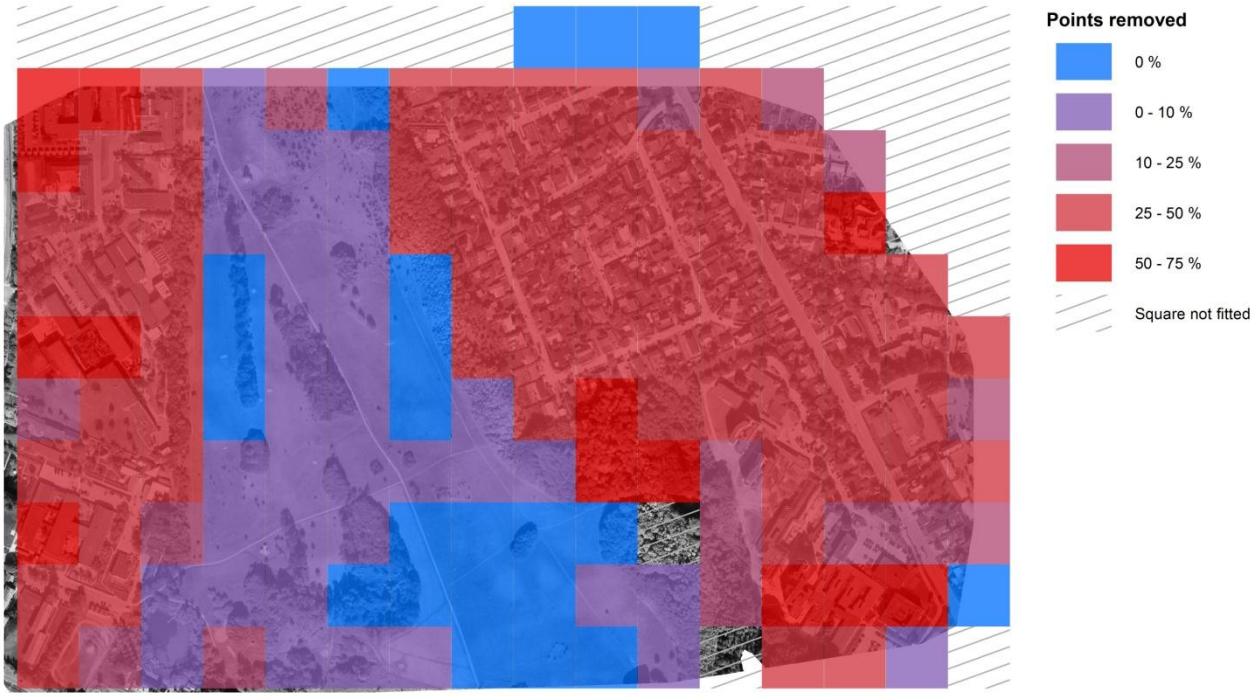


Figure 93 – Removed points as percent of the original number of points in each square. Orthophoto can be seen behind.

6.3.2 Step 2

In step 2 the remains of the non-terrain points are removed with a slope-based filter. The final program makes a result file, where every point is listed together with the following information:

- Terrain point or non-terrain point
- The slope of the fitted plane (slope of terrain)

The result file contains 2.1 million points, which means that the data must be processed in order to visualize the results. This is done by converting the points to a raster dataset, see Figure 94 and Figure 95. The *'cell size'* controls the size of the raster cells and is chosen to be 1 m in order to get a smooth visualization. The *'value field'* describes which data from the shapefile to use when the raster value is calculated. The input in the value field varies according to the information which should be visualized. The *'assignment type'* is the input which decides how the values in the raster should be calculated. All points inside a cell are used to calculate the values of the cell and here the mean value is used.

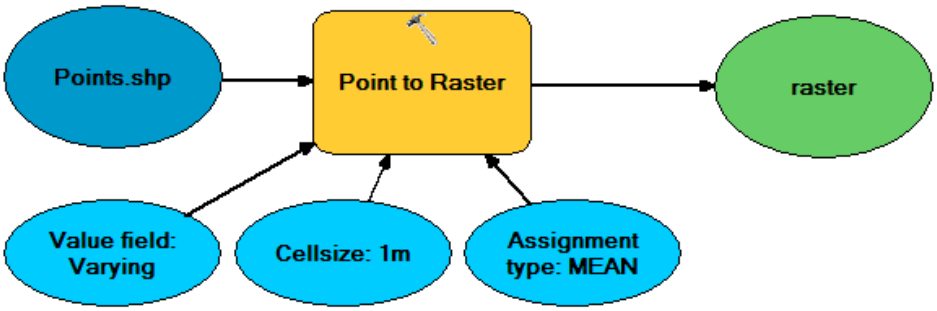


Figure 94 – Points to raster

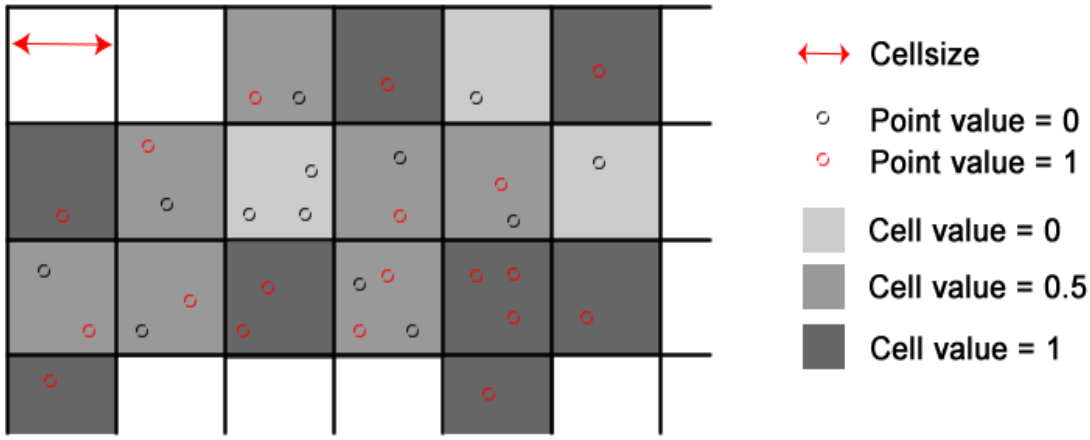


Figure 95 – Point to raster with 'Assignment type' = MEAN

When the points are converted to raster a generalization is needed in order to see patterns. This is done with two following focal analyses, see Figure 96 and Figure 97. The focal analysis is described as: *"The computation of an output raster where the output value at each cell location is a function of the value at that cell location and the values of the cells within a specified neighborhood around the cell."* (Esri n.d.)

The neighborhood is chosen as a circle with a radius of 10 m and the statistics type is again the mean value of all the cells in the neighborhood. It should be mentioned that the generalization means that some of the values cannot be fully trusted. This applies especially for the edges around the areas where there are no points in the dataset.

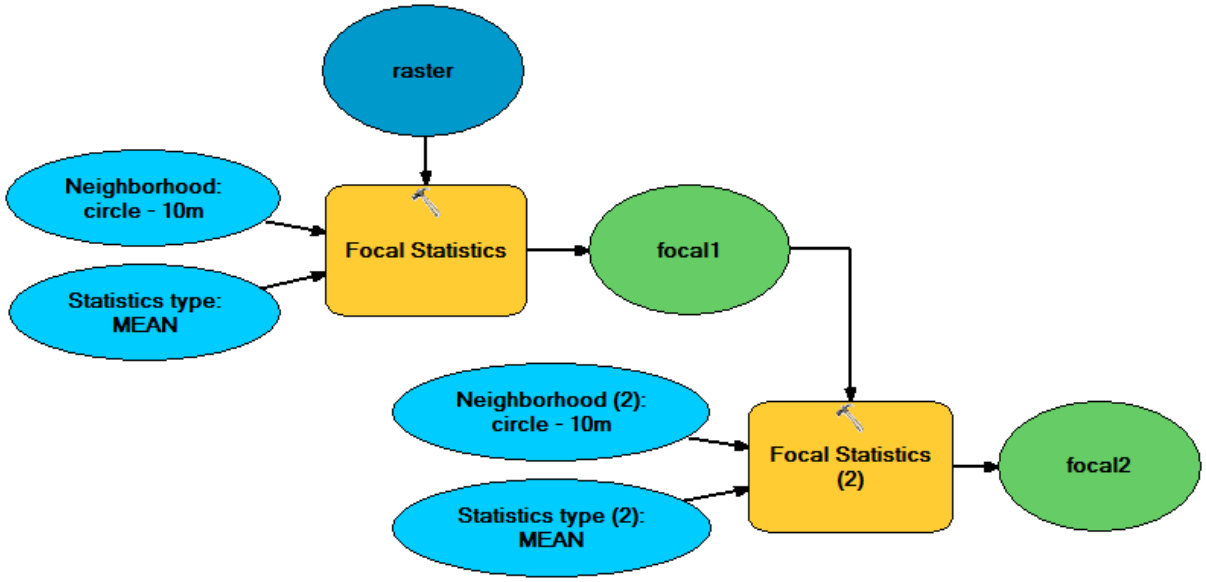


Figure 96 – Focal analysis on the raster data

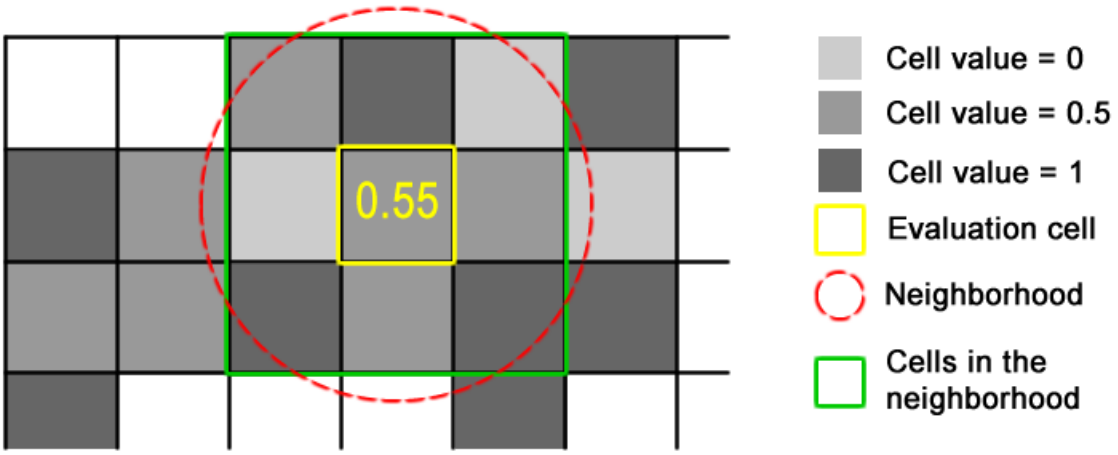


Figure 97 – Principle of a focal analysis for one cell

6.3.2.1 Terrain slope

First the slope of the terrain is showed, see Figure 98. This is done in order to verify if the slope correction worked as expected, see section 0. If the slope correction worked well there should be no places with big slopes.

It can be seen that in general the area is quite flat. In the open area it can be seen that there is a slope in the open area towards the forested area, which match with the reality. In the build-up areas the slope varies more, and especially around some of the houses the slopes are steeper than the surroundings. It was expected that the slope should vary a little more in the build-up areas because of human influence on the terrain.

At the border of the forest, the slope is approaching 46 degrees, but it should be remembered that the data cannot fully be trusted because there is almost no points inside the forest. Furthermore, the problem seems bigger than it is in reality, since insignificant errors at the edges are blown up to be visualized as 20 meters, due to the two focal analyses used.

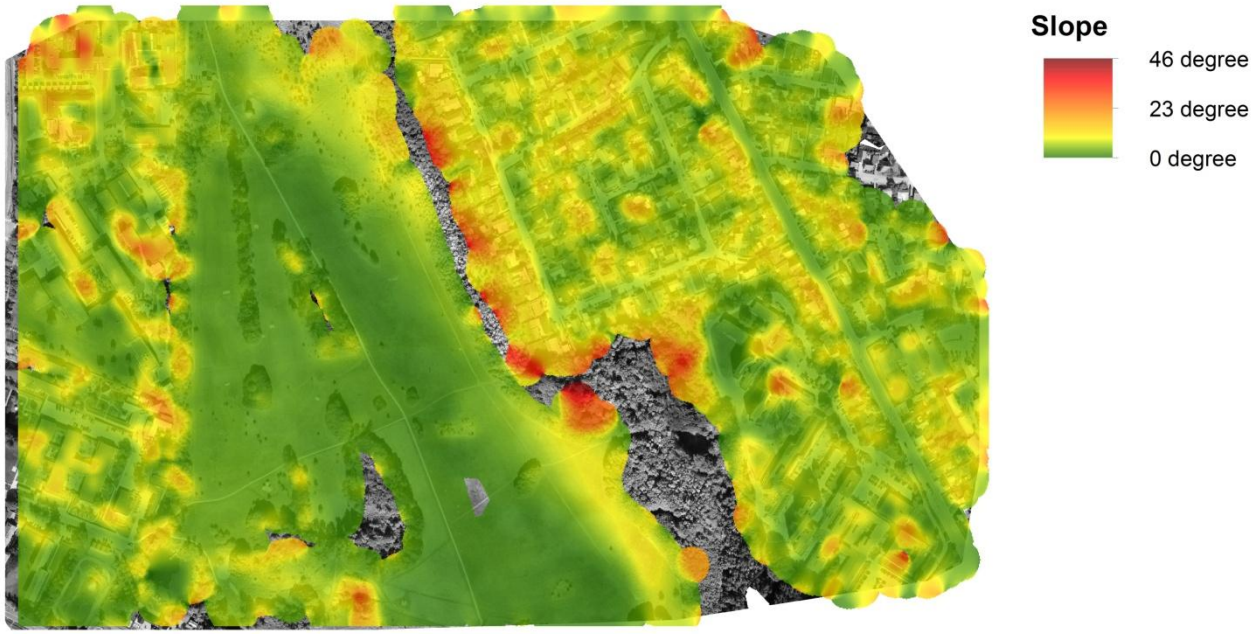


Figure 98 – Terrain slope according to the slope correction in step 2. Orthophoto can be seen behind.

6.3.2.2 Removed points

The number of removed points, or points classified as non-terrain can also be visualized. In the data, a terrain point has the value 1 and a non-terrain point the value 0. After the conversion to raster and the generalization, the raster cells now contains a mean values representing the number of terrain points in the data. This number should be recalculated to represent the number of non-terrain points in percent. This calculation is done with the tool *"raster calculator"* and the following formula:

$$\text{new cell value} = (1 - \text{cell value}) \cdot 100 \qquad (6.1)$$

where *'new cell value' is the percentage representing the number of non-terrain points*
'cell value' is the present cell value

At Figure 99, it can be seen that only a few points are removed in the open areas. In the build-up areas more points are removed, but in general more points are removed around the buildings than on the roads.

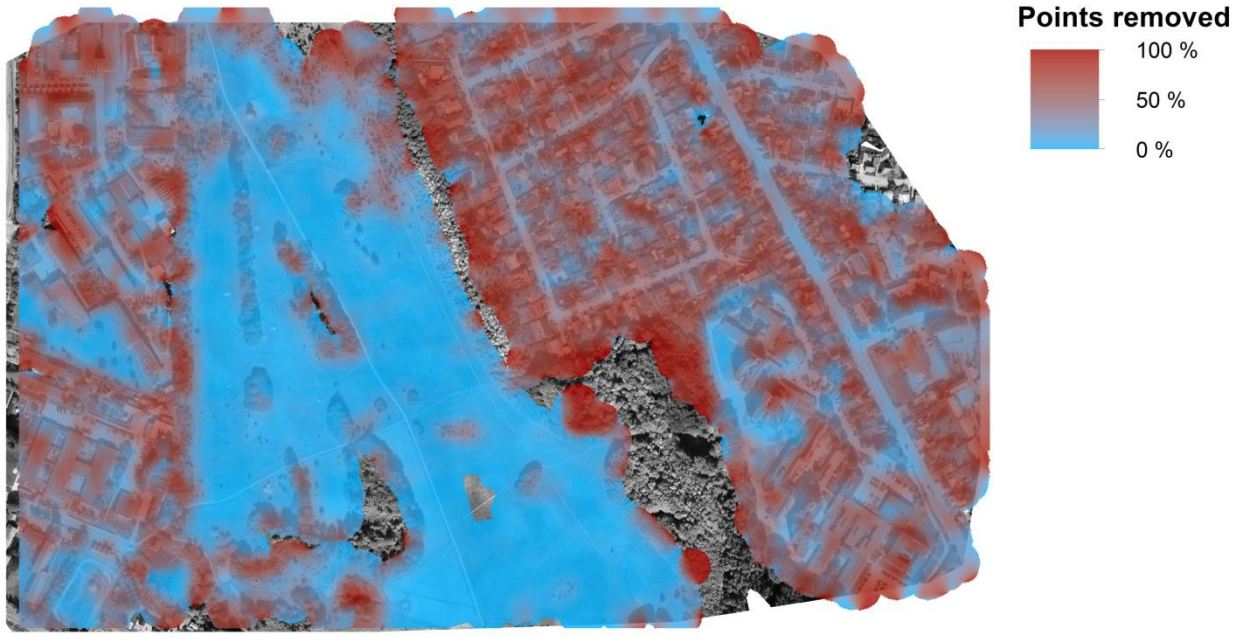


Figure 99 – Removed points in step 2. Orthophoto can be seen behind.

6.4 Conclusion

The purpose of this chapter is to present a GUI program for the created DTM algorithm. At the same time, the project dataset has been processed in the program. Thereby, the chapter answers the 3rd question from the problem statement:

How can step 1 and 2 be implemented in a user friendly program, which makes it possible to run the algorithm on a photogrammetrically produced point cloud?

A multi-threaded GUI program called *DTM Creator 2.0* has been created. The program is written in Python and Cython with the use of various external libraries. Especially, it should be mentioned that the use of the GUI extension module called Qt makes it possible to create a GUI layout which is very familiar to Windows users.

The program is considered to be very user friendly. For instance, the program checks if the previous step has been performed before allowing the next step to be started. Furthermore, when running each step, the program will give feedback to the user, telling the progress of the current task. Another example is that in case of an invalid working directory, the user is forced to select a valid working directory, before the program will start.

With the use of the four steps 0, 1.1, 1.2 and 2, it is possible to process the project dataset in the GUI program. After the project dataset has been processed, some of the outputted documentation files have been processed by GIS software in order to generate maps which

reveal how the used algorithms have performed. In general, it seems like the algorithm has performed as expected.

The result after the project dataset has been processed is that 46.5 % of the points are removed and marked as non-terrain:

```
Total points before:            2906537
Total points after filtering:   1554927
Total points removed:           1351610
Percent points removed:          46.5 %
```

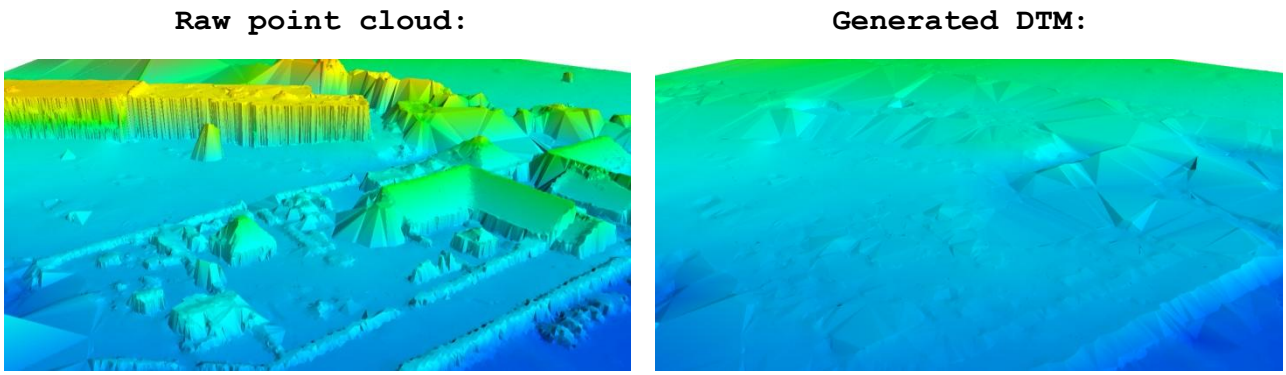


Figure 100 – Points removed with the program DTM Creator 2.0

Based on the final classification of the point cloud (including non-terrain points from both Step 1 and 2), the following overview of removed points can be generated in GIS using a focal analysis similar to the analyses made in section 0:

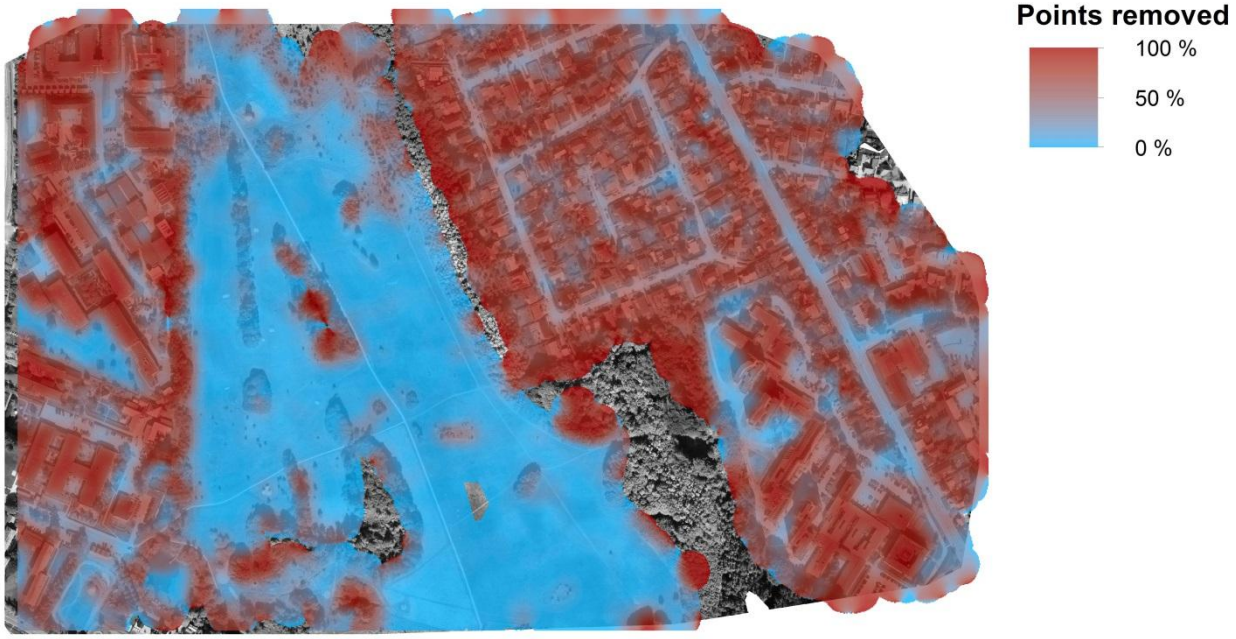


Figure 101 – Focal analysis of removed points

Chapter 7: Quality Assessment

7.1 Introduction

This chapter is about assessing the quality of the developed algorithm, and thereby answering the last question from the problem statement. The quality of the algorithm should in this case be understood as the effectiveness of the removal of non-terrain points and *not* the speed of the algorithm or whether elegant coding practices have been used, etc. In short, the chapter simply deals with the question whether a satisfying DTM has been produced or not.

The quality of the algorithm is initially assessed by doing a visual inspection of the DTM generated based on the drone data from the previous project by Matthesen and Schmidt (2014). If the algorithm works well, the non-terrain points should be removed in the filtered point cloud.

A more theoretic/mathematical approach can also be used by comparing the result of the algorithm with an existing "true" result. In the previous project by Matthesen and Schmidt (2014), the DTM produced from drone data using the developed algorithm was compared to an existing DTM created by the Danish Geodata Agency. Such a comparison can be problematic since the DTMs originate from point clouds collected using different methods. The DTM by the Danish Geodata Agency is produced based on points from aerial laser scanning, whereas the DTM created using the developed algorithm is produced from drone images using photogrammetry.

Laser scanning has an advantage over photogrammetry when it comes to DTM generation. Points will be measured on top of vegetation using photogrammetry, but with laser scanning there is a chance that the last echo of the laser beam has penetrated through the branches and maybe reached the ground. In other words, a big difference between a DTM created from laser scanning and the DTM created from photogrammetric data is not always caused by a bad algorithm. Instead the difference might be caused by an area largely covered by vegetation. Differences could also be caused by areas where no points could be measured photogrametrically due to bad texture. Furthermore the two datasets are not captured at the same time, which could result in differences caused by earthworks or natural landslide. Due to the reasons presented above, it is decided not to directly compare the DTM produced in this project with the existing DTM from the Danish Geodata Agency.

The laser scanned point cloud used to create the existing DTM from the Danish Geodata Agency is freely available from the agency's webpage. Each point is classified as either terrain or non-terrain. In order to get the most accurate assessment of the quality of the developed algorithm, it is decided to use the algorithm developed in this project to re-classify the laser scanned point cloud. By comparing the original classification with the re-classification it is possible to assess the quality of the developed algorithm. It should be mentioned that the comparison of the two classifications only makes sense under the assumption that the original classification is correct.

7.2 Visual inspection

The visual inspection is simply performed by visually comparing the point cloud before and after the DTM algorithm has been applied. This section presents a number of samples of the point cloud before and after the DTM algorithm is applied. The samples are selected to represent different challenges for the DTM algorithm. For each sample, the DTM from the previous project will also be showed.

In the following, the term *new DTM* will be used for the DTM produced in this current project. Similar, the term *previous DTM* will be used to describe the DTM produced in the previous project by Matthesen and Schmidt (2014).

Open area at golf course:

DSM ***Previous DTM*** ***New DTM***

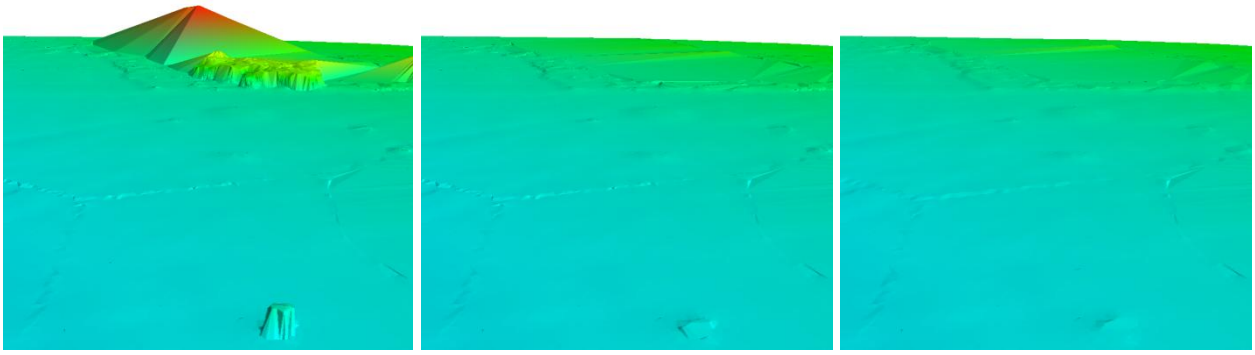


Figure 102 – Sample from golf course with limited number of non-terrain points

It is expected that the algorithm performs well in open areas since they contain only few non-terrain points. Furthermore, the non-terrain points in open areas are in general not very clustered which makes them easier to remove compared to for instance points on a large roof. From the sample at Figure 102 from the old golf course in the project area, it can be seen that the algorithm as expected perform very well in open areas. It can also be seen that there is no significant difference between the previous DTM compared to the new DTM. By rapidly changing between views of the two DTMs at a computer it is however possible to see that the new DTM is smoother than the previous DTM.

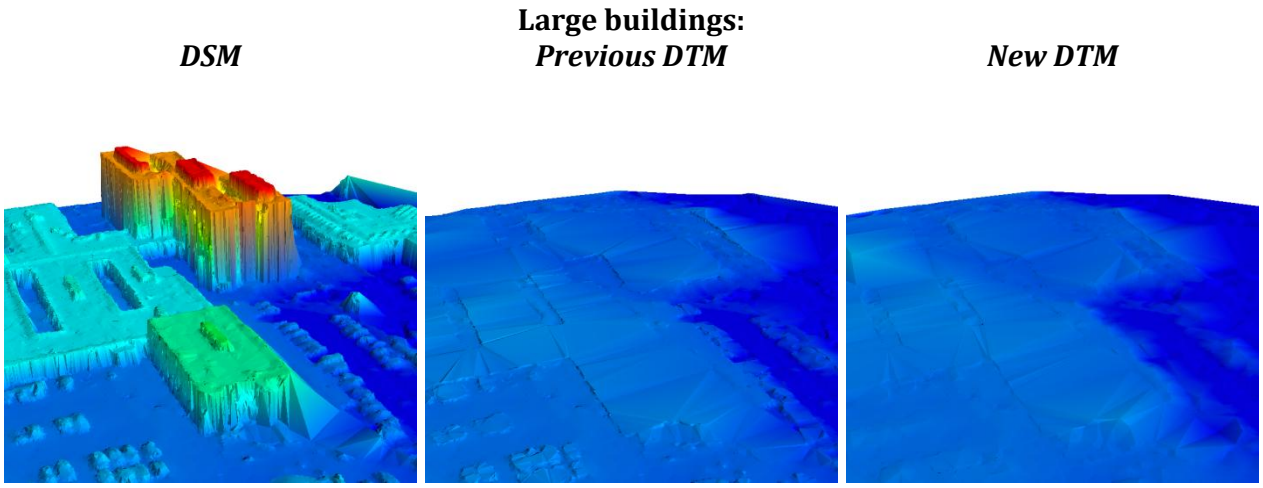


Figure 103 – Sample with large buildings

Figure 103 above shows a sample from a part of the point cloud which contains both tall buildings and low-rise buildings covering a large area. The clustered points on the roof of the tall buildings are expected to be easy to remove due to the big height difference to the surrounding terrain points. On the other hand, the points on the low-rise buildings are much harder to remove, since they can be mistaken as hills. However, at Figure 103, it can be seen that all non-terrain points are removed on both the tall and the low-rise buildings. It is assessed that the open atriums in the low-rise buildings heavily contributes to the successful removal of the points measured on top of the low-rise buildings.

From Figure 103, it can also be seen that the new DTM is much more smooth compared to the previous DTM. In other words, the new algorithm is able to remove the problematic non-terrain points close to the ground, which could not be removed by the earlier algorithm. This is especially easy to see by looking at the points measured at the edges of cars. To see this more easily, a close-up plot of some parked cars has been generated:

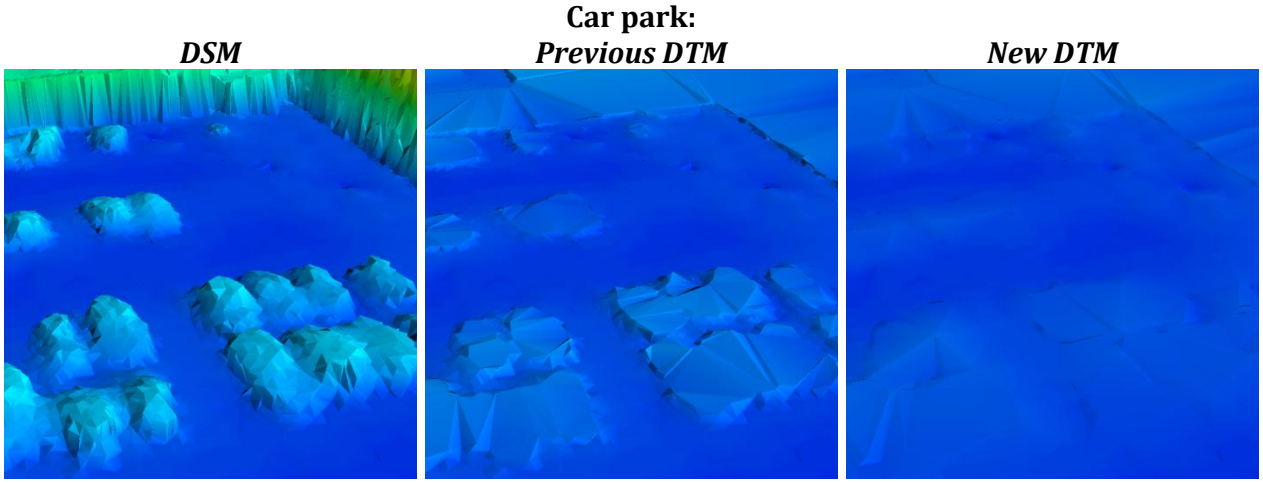


Figure 104 – Sample from a car park

The plot of the parked cars at Figure 104 shows that the slope-based filtering effectively removes the problematic non-terrain points found at the edge of the cars in the previous DTM. The slope-based filtering approach does not only remove low non-terrain points measured on

the side of cars. It is also removing low points measured at the side of buildings and vegetation such as hedges, see Figure 105.

Boundaries near buildings and hedges:

DSM ***Previous DTM*** ***New DTM***

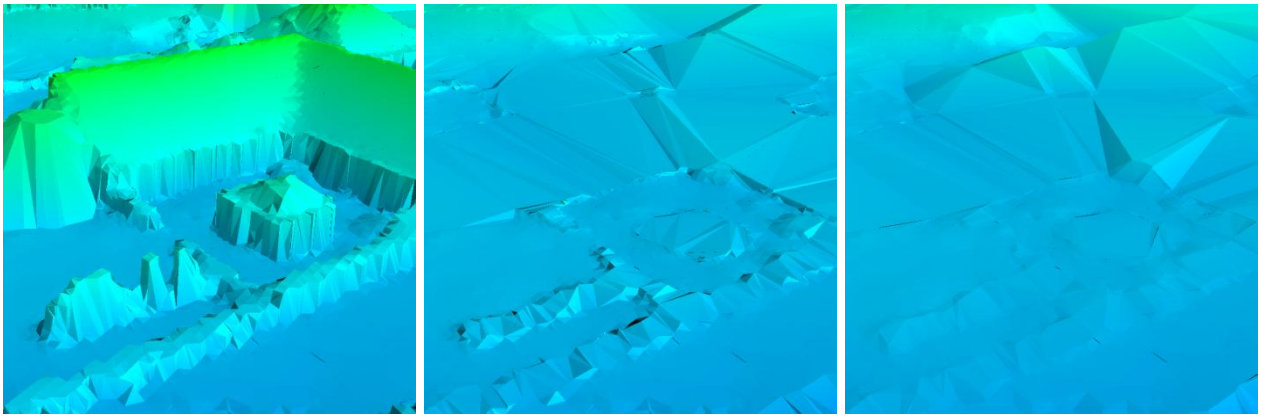


Figure 105 – Sample with hedges

Single-family homes (1):

DSM ***Previous DTM*** ***New DTM***

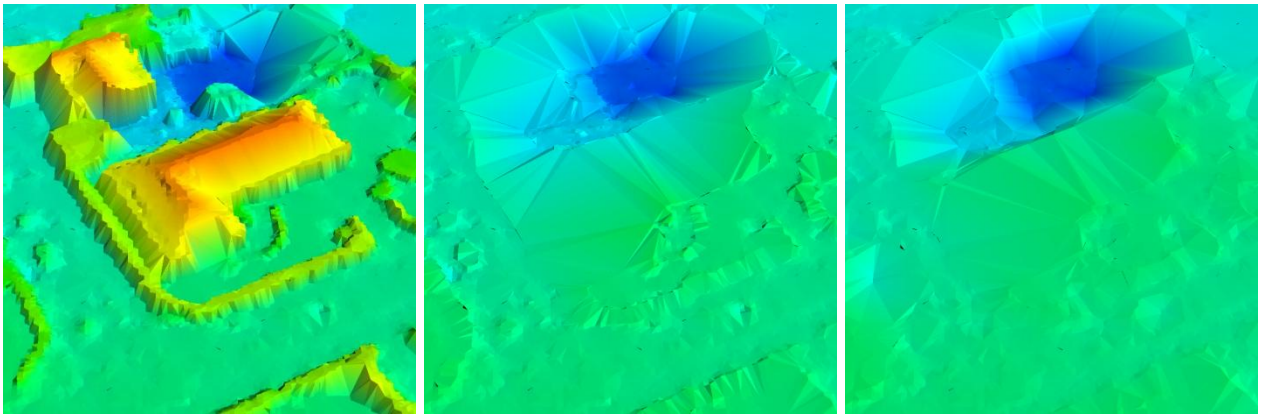


Figure 106 - Sample from complex area with single-family houses

Single-family homes (2):

DSM ***Previous DTM*** ***New DTM***

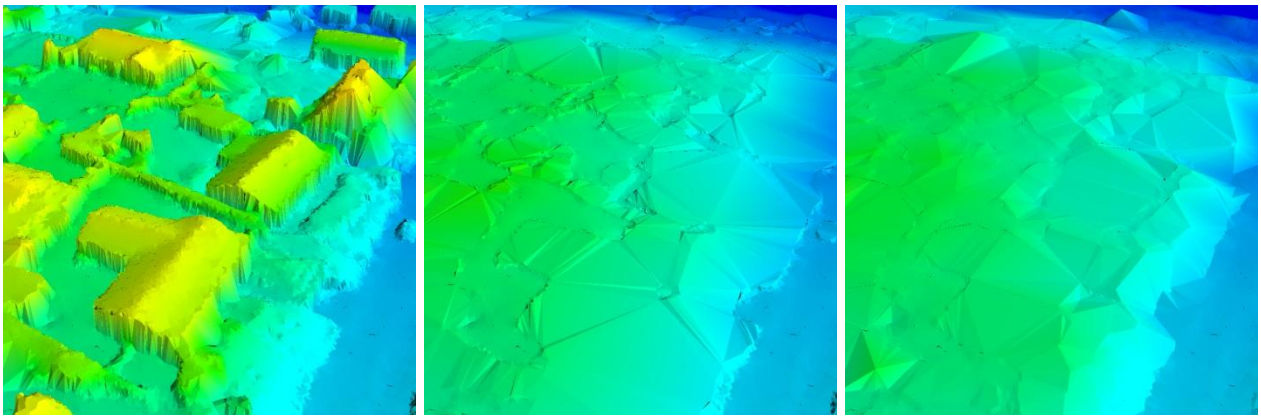


Figure 107 - Sample from complex area with single-family houses

Figure 106 and Figure 107 shows that all clusters of non-terrain points are removed in areas with single-family homes. This is expected since such areas contain a good mixture of terrain and non-terrain points because the houses in general are quite small. It is however hard to create a smooth DTM, because the areas are very complex with lots of low-rise vegetation and edges in the terrain caused by local gardening design.

In the previous DTM created in the single-family areas, there are a lot of non-terrain points near the ground. These low non-terrain points are removed in the new DTM because of the use of the slope-based filtering in Step 2. In general, it is assessed that the new DTM gives the most accurate representation of the terrain in areas with single family houses.

Based on the samples presented so far, it might seem as if the new algorithm results in a flawless result. This is unfortunately not the case. In the previous project, some problematic low-rise buildings were experienced. As it can be seen at Figure 108, the problematic buildings are still not removed in the new DTM.

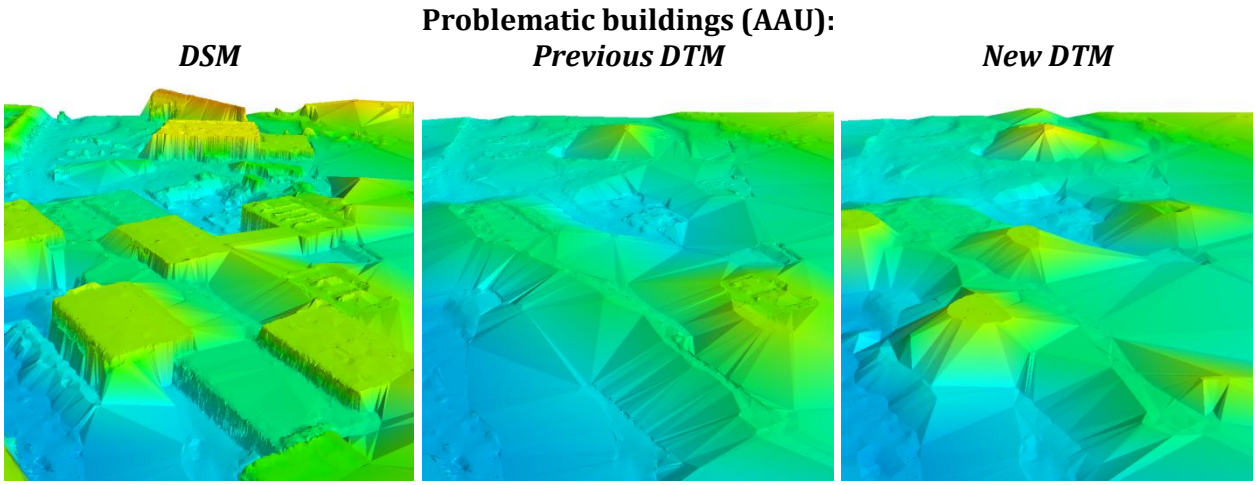


Figure 108 – Sample with problematic low-rise buildings covering a large area

It is assessed the problematic buildings cannot be removed because they simply cover a too large area and therefore are mistaken as hills. A solution to the problem could be the use of existing map data. For instance, the national vector map *Kort10* covering the entire country of Denmark exists(Geodatastyrelsen n.d.). One of the themes in this map is building polygons. If all points within these building polygons are removed, it is assessed that the problem is solved.

7.2.1 Test of proposed solution for problematic buildings

It is decided to test if the use of existing map data can eliminate the problem as expected. At Figure 109 is a map of showing the Kort10 building theme on top of an orthophoto of the problematic buildings. The block-minimum points are also plotted at the map.

Figure 109 – Polygons from Kort10 surrounds the problematic buildings

From the map at Figure 109, it is clear why the buildings are hard to remove. There are no terrain points measured east of the buildings due to vegetation of trees and bushes.

The points inside the Kort10 building polygons can be removed using the *ray-casting algorithm*. In this project, a horizontal ray is used, but in fact the ray could go in any direction. However, vertical or horizontal rays are preferred, since it makes it easier to calculate intersections with polygon edges. From a given point, the number of times the ray intersects with the polygon's edges determines if the given point is inside or outside of the polygon. If the ray intersects an even number of times, the point is outside the polygon. Similar, an odd number of intersections mean that the point is placed outside the polygon(Lawhead 2011). This is illustrated at Figure 110.

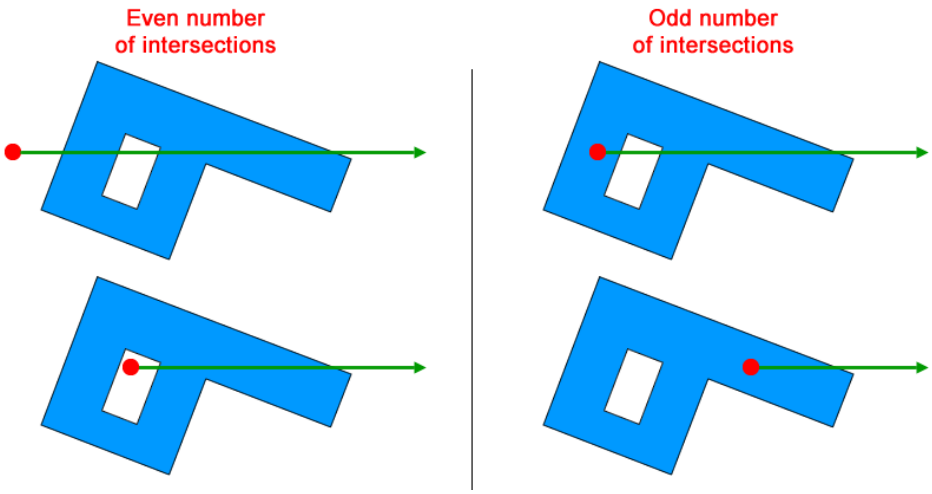


Figure 110 – The number of intersections of the ray (green) determines if the point is inside the polygon or not

At first, the ray-casting algorithm might seem quite hard to implement, but in fact, it is quite simple and mostly consists of if-statements, as it can be seen at the Python example code below: (Lawhead 2011)

```
def point_in_poly(x,y,poly):
    n = len(poly)
    inside = False
    p1x,p1y = poly[0]
    for i in range(1,n+1):
        p2x,p2y = poly[i % n]
        if y > min(p1y,p2y):
            if y <= max(p1y,p2y):
                if x <= max(p1x,p2x):
                    if p1y != p2y:
                        xints = (y-p1y)*(p2x-p1x)/(p2y-p1y)+p1x
                    if p1x == p2x or x <= xints:
                        inside = not inside
        p1x,p1y = p2x,p2y
    return inside
```

It should be mentioned, that the above implementation of the ray-casting algorithm will fail in some rare cases. More advanced implementations exist, which account for these errors. It is however decided to use the simple implementation above, since the ray-casting algorithm only is used for testing a proposed solution and is therefore not really a vital part of the project.

In ArcMap, a buffer of 1.5 meters has been added to the building theme before removing the points inside the buildings. It turned out that a buffer of this size was needed in order to remove all non-terrain points measured on the roof.

The building polygons with buffers are stored as a shapefile. In order to easily read this file, the Python extension module called *pyshp* is used. The extension module can be downloaded at https://code.google.com/p/pyshp/ for free.

Figure 111 –Pyshp is used to read the building theme from Kort10 (Python Shapefile Library n.d.)

A python script has been written which based on the raw point cloud removes the points inside the building polygons. The Python script can be found at *'DVD\python\raycasting\program\temp2\STEP 0 - Merge clusters\kort10_remove_building.py*'. Only a small area near the problematic buildings has been processed with the Python script. At Figure 112 below is an example of the raw point cloud where the points inside the building polygons are removed:

Problematic buildings (AAU):

Original raw point cloud ***Buildings removed with ray-casting***

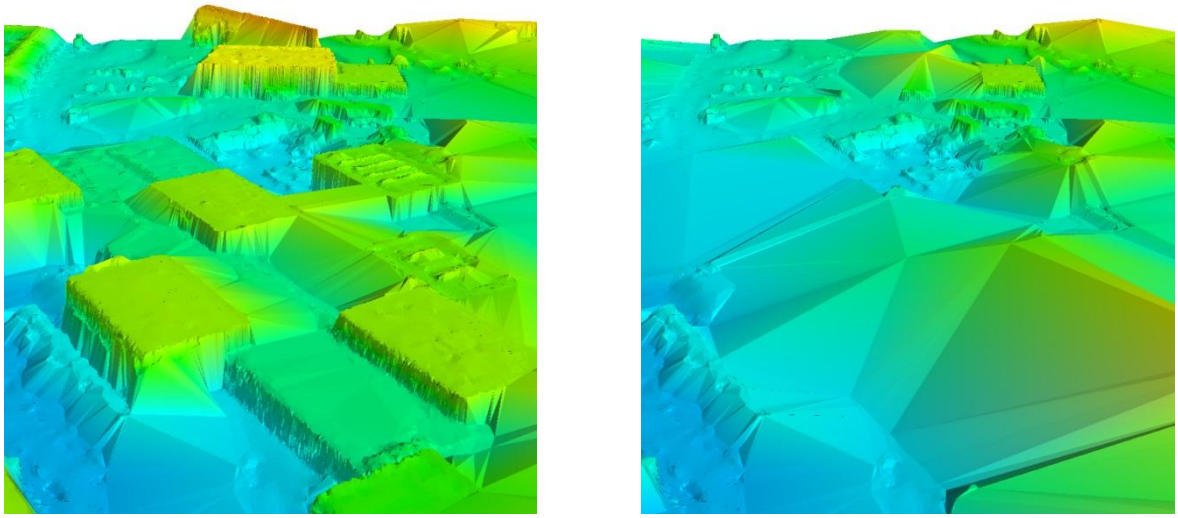


Figure 112 – Plots before and after buildings are removed with the ray-casting algorithm

The raw point cloud where the points inside the building polygons are removed is used to generate a new DTM by using the developed DTM algorithm:

Problematic buildings (AAU):

DTM without ray-casting ***DTM with ray-casting***

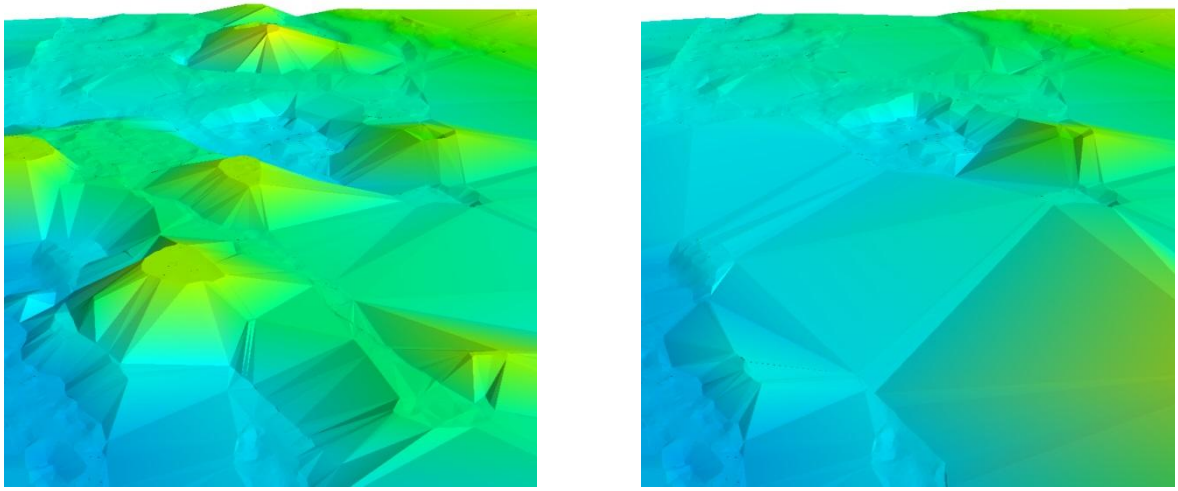


Figure 113 – Plots before and after buildings are removed with the ray-casting algorithm

It can be seen that the use of existing map data effectively removes the non-terrain points on top of low-rise buildings covering a large area. However, the DTM produced with ray-casting, see Figure 113, still contains some non-terrain points measured at the top of vegetation. These

points cannot be removed because they are not surrounded by terrain points. If the point cloud was produced from laser scanning, points would probably be measured on the ground nearby, and the problem wouldn't exist. A solution to the problem could be to add a final step to the DTM algorithm, where isolated clusters of points are found and classified as non-terrain points, since they cannot be trusted.

To sum up, it can be concluded that the use of existing map data can improve the algorithm, especially at places with low-rise buildings covering large areas. Furthermore, it should be considered if isolated clusters should be found and classified as non-terrain points.

7.3 Classification accuracy and errors

This section is about re-classification of a laser scanned point cloud in order to find the quality of the developed algorithm. The point cloud from the Danish Geodata Agency (GST) is used for this purpose since it is freely available and already has a classification of terrain and non-terrain points. It should be mentioned, that the results from this section are calculated under the assumption that the original classification is true.

The process in this section is first to re-classify the point cloud and afterwards compare the new classification with the original "true" classification. This makes it possible to calculate different accuracies and error types that the developed algorithm delivers. In the following text the height model from GST is called the '*original point cloud*', when showed in the original form and with the original classification. When the point cloud has been processed by the program *DTM Creator 2.0* developed in this project, it is called the *'filtered point cloud'*.

The point cloud from the Danish Geodata Agency is delivered as LAS files, each containing a 10x10 km square of the point cloud. In this comparison an area almost equal to the test area is used because it is a familiar area, which has been discussed earlier in the project.

The LAS file format is a public file format used for 3D point clouds (ASPRS online n.d.). The format is a binary format, and cannot just be used as input in the program DTM Creator 2.0. The LAS file must be converted to a HDF5 file in order to be compatible with DTM Creator 2.0. This conversion is done in Python. In order to handle the LAS files, an extension module, called libLAS, is installed. The module is described at http://www.liblas.org, and a simple installer can be downloaded for free at http://www.lfd.uci.edu/~gohlke/pythonlibs. The LAS file is converted with the python script '*DVD:\python\control_GST\las_to_hdf\las_to_hdf.py'.* In the conversion the E, N and H coordinates are saved together with the classification. Only points classified as either non-terrain (code 1) or terrain (code 5) are saved in the HDF5 file (Dalå og Rosenkranz 2014).

Figure 114 – The libLAS logo(libLAS n.d.)

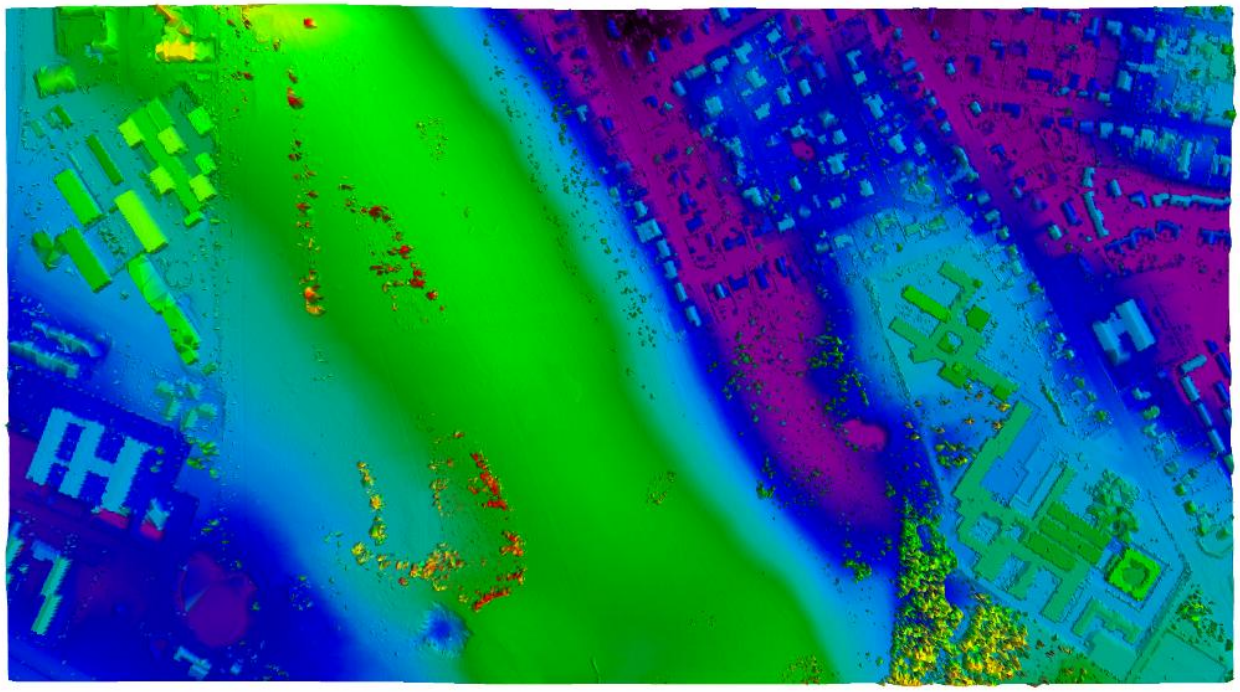
The new HDF5 file is used as input in the program. The parameters in step 1.1, 1.2 and 2 are the same as found under the testing in Chapter 4 and Chapter 5.

7.3.1 Visual comparison

Figure 115 shows the original point cloud and the terrain points from the filtered point cloud. It can be seen that buildings and vegetations has been removed, and in overall the program DTM Creator 2.0 has made a good job generating the DTM. A closer look reveals that the program troubles with pits, as seen in the bottom of the area just to the left of the middle. Here the program has classified a lot of terrain points as non-terrain. The program is not designed to be good at handling discontinuities and abrupt changes in the terrain, even though it should be able to handle terrain with a general slope.

Figure 116 shows terrain points in the original point cloud and terrain points in the filtered point cloud. The difference between the two point clouds is small, but in general the filtered point cloud is smoother. Small slopes and dikes are smoothed, due to the problems the algorithm has with handling discontinuities and abrupt changes.

Original GST point cloud (DSM):



Terrain point in the filtered point cloud (DTM):

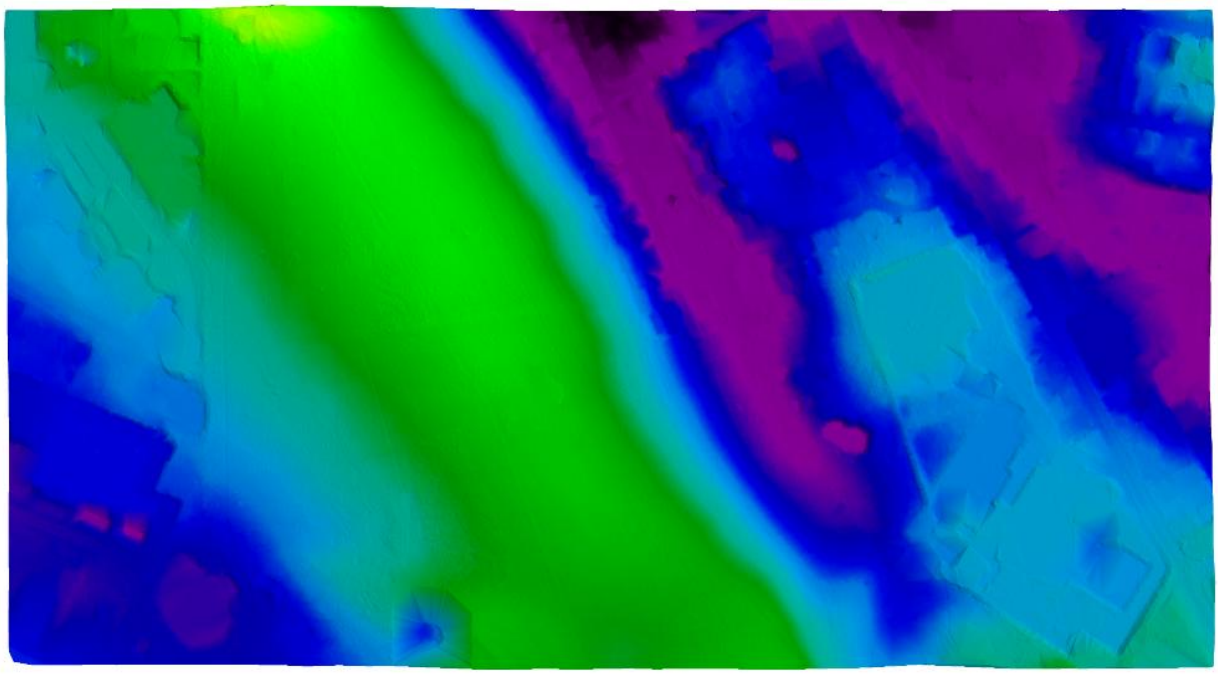
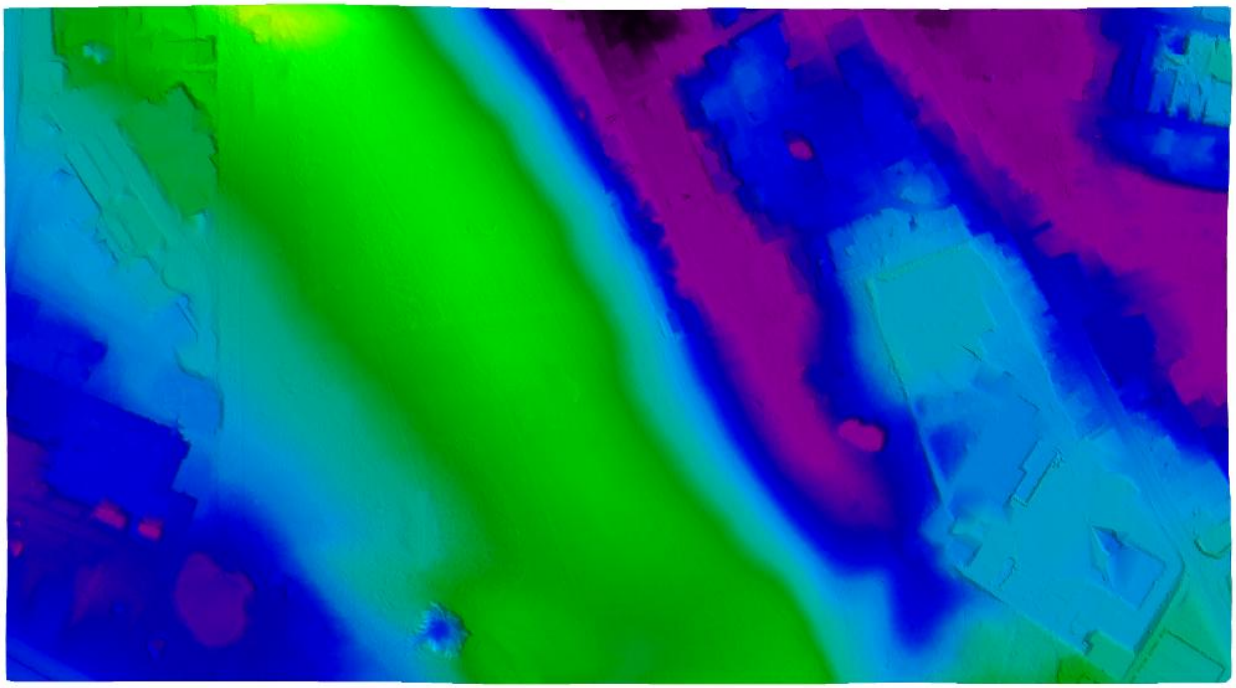


Figure 115 – Top: Point cloud before filtering (DSM). Bottom: Point cloud after filtering (DTM)

Terrain points in the original GST point cloud (DTM):



Terrain points in the filtered point cloud (DTM):

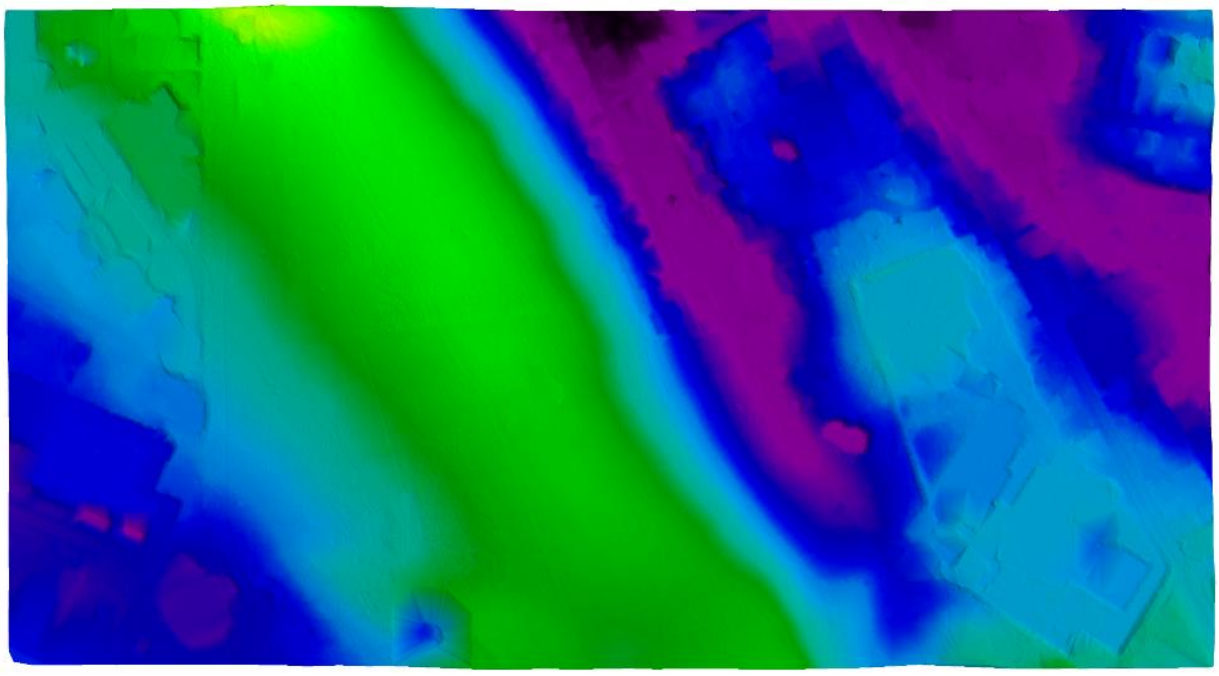


Figure 116 – Top: Terrain points in the original point cloud. Bottom: Terrain point in the filtered point cloud.

7.3.2 Error matrix

In section 2.2.1 two types of errors, which can occur in the filtered point cloud, was explained:

> **Error of omission:** A terrain point is classified as non-terrain
> **Error of commission:** A non-terrain point is classified as terrain

As explained earlier, filtering algorithms in general focus on minimizing commission errors. All non-terrain point should be classified as non-terrain even if it happens at the cost of classifying a few terrain points as non-terrain.

A simple and effective way to present the errors is in an *error matrix*. With the numbers in the matrix it is possible to calculate different accuracies. An error matrix for DTM generation will look like this:

		Filtered point cloud	
		Terrain	Non-terrain
Original point cloud	Terrain	A	B (omission errors)
	Non-terrain	C (commission errors)	D

Where *A is the number of terrain points which are correctly classified as terrain*
B is the number of omission errors
C is the number of commission errors
D is the number of non-terrain points which are correctly classified as non-terrain

The program outputs the final classification in three files. One file from the initial selection, containing points classified as non-terrain and two files from the final selection containing terrain points and non-terrain points respectively. The three files are used to make the error matrix.

The error matrix and the accuracies are calculated with the python script *'DVD:\python\control_GST\compare\error_matrix.py'*. The results are shown in the error matrix below. Furthermore the points are visualized on Figure 117.

		Filtered point cloud	
		Terrain	Non-terrain
Original point cloud	Terrain	230,551	33,819
	Non-terrain	3,744	58,610

Table 20 – Error matrix

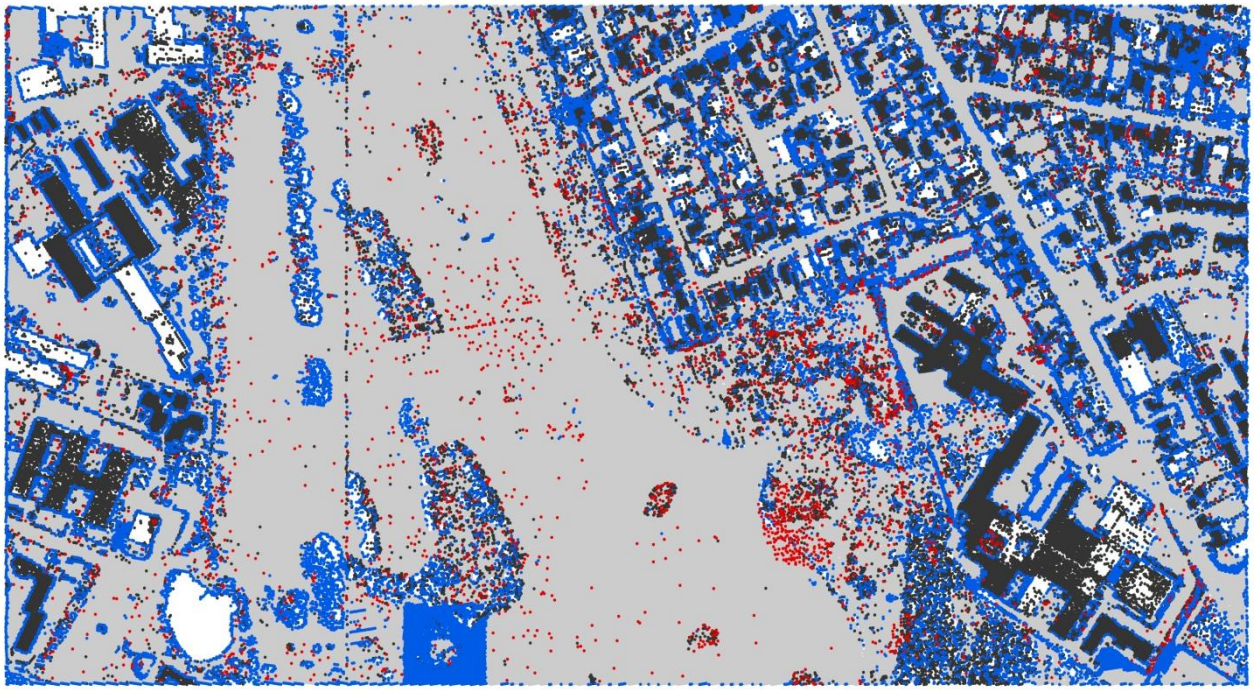


Figure 117 – Visualization of points from error matrix. Colors are equal to the colors used in the error matrix in Table 20. White areas contain no data.

At Figure 117 it can be seen that a lot of the omission errors are around the buildings. The commission errors are more evenly distributed throughout the area without any real systematic.

7.3.2.1 Accuracies

The first accuracy which can be calculated is the *overall accuracy* which is found by dividing the total number of correctly classified points with the total number of points in the error matrix. The overall accuracy therefore describes how often a filtered point is classified correct. (Congalton 1991)

$$\text{Overall accuracy} = \frac{A + D}{A + B + C + D} \cdot 100 = 88.5\ \% \qquad (7.1)$$

The accuracy can also be calculated for each category (terrain and non-terrain), but in contrast to the overall accuracy there are two different ways to calculate it. One is called '*producers accuracy'* and the other is called '*users accuracy*'.

Producers accuracy indicates how often an original point is classified correct. If you have a terrain point in the original point cloud (assumed true terrain point), the producers accuracy describes the probability that the point also is classified as terrain in the filtered point cloud. The producer of the algorithm can use these accuracies to tell something about how good the algorithm is to recognize each category. (Congalton 1991)

The producers accuracy is calculated by dividing the number of correctly classified points in a category, with the total number of points in that category in the original point cloud. (Congalton 1991)

$$\text{Producers accuracy} \begin{cases} \text{terrain} = \frac{A}{A+B} \cdot 100 = 87.2\ \% \\ \text{non} - \text{terrain} = \frac{D}{C+D} \cdot 100 = 94.0\ \% \end{cases} \quad (7.2)$$

Users accuracy can be described as the opposite of the producers accuracy. It indicates how often a point classified as e.g. terrain also is a terrain point in the original point cloud (or in reality). (Congalton 1991)

The reason why it is also interesting to look at the users accuracy can be explained with this little example. If the producers accuracy for terrain is 100 % one might think that the algorithm is really good at classifying terrain points correctly. This can be true, but it can also be because all points in the entire point cloud are classified as terrain. If this is the case, the user cannot trust the classification because a lot of the points classified as terrain in reality are non-terrain. The users accuracy gives the user an idea of how reliable the classification is.

If the algorithm functions well and classify the point cloud correctly, both the producers and the users accuracy should be high for both terrain and non-terrain.

The users accuracy is calculated by dividing the number of correctly classified points with total number of points classified in that given category. (Congalton 1991)

$$\text{Users accuracy} \begin{cases} \text{terrain} = \frac{A}{A+C} \cdot 100 = 98.4\ \% \\ \text{non} - \text{terrain} = \frac{D}{B+D} \cdot 100 = 63.4\ \% \end{cases} \quad (7.3)$$

From the accuracies, it can be concluded that the filtering is succeeded with fine results. Especially the users accuracy for terrain is good. If a point is filtered as terrain, there is a probability of 98.4 % that the point also was a terrain point in the original point cloud. The high users accuracy for terrain comes at a cost which can be seen in the users accuracy for non-terrain points. The algorithm classifies some terrain points as non-terrain, which means that the user will sometimes find terrain where the filtered point cloud says there are non-terrain objects. However, it is assessed that this is acceptable since the purpose of the algorithm is to generate a DTM, and not to model the non-terrain objects.

The producers accuracy for non-terrain points shows that a good deal of the non-terrain points actually are classified as non-terrain. This information together with the users accuracy tells that the developed algorithm works in accordance with the general consensus of which errors is best to make.

7.3.2.2 Errors

In addition to the accuracy, the percentage of omission and commission errors can be calculated. Both error types can also be read from the producers accuracy, because the number of errors is the opposite of the number of correctly classified points.

A total number of errors can also be calculated. This number is the opposite of the overall accuracy, and describes the total number of points which are wrongly classified.

$$\begin{aligned} \text{Omission error} &= \frac{B}{A+B} \cdot 100 = 12.8\,\% \\ \text{Commission error} &= \frac{C}{C+D} \cdot 100 = 6.0\% \\ \text{Total errors} &= \frac{B+C}{A+B+C+D} \cdot 100 = 11.5\,\% \end{aligned} \tag{7.4}$$

From the equations above, it can be seen that the developed algorithm minimizes the number of commission errors compared to the number of omission errors. This was expected, since the algorithm was designed to do so.

The sizes of the errors can be compared to the sizes of the errors in the comparison by Sithole and Vosselman (2003). Through their tests they found that the errors were of these sizes:

$$\begin{aligned} \text{Omission error (Type 1)} &= 0 - 64\,\% \\ \text{Commission error (Type 2)} &= 0 - 19\,\% \\ \text{Total errors} &= 2 - 58\,\% \end{aligned} \tag{7.5}$$

where *Type 1 and Type 2 are the names used in Sithole and Vosselman* (2003)

Compared to those ranges, the errors in the developed algorithm is in the good end of the range. If the numbers of errors should be compared directly, the developed algorithm from this project should be used on the same datasets as used in the report by Sithole and Vosselman (2003).

7.4 Conclusion

The purpose of this chapter is to control the DTM generated by the developed program *"DTM Creator 2.0"*, and thereby answer the 4th question from the problem statement:

How good is the quality of the developed DTM algorithm?

The quality of the generated DTM is investigated in two ways. First a visual inspection where the DTM for the project area is compared to the original DSM and the DTM generated in the previous project by Matthesen and Schmidt (2014). Afterwards the program is used to generate a DTM based on data from the Danish Geodata Agency (GST). The data is already classified as terrain and non-terrain. With the assumption that the GST dataset is a "true" classification, some accuracies can be calculated which tells how good the algorithm is at classifying points as terrain or non-terrain.

The visual inspection showed that in general the program does a good job generating a DTM. However, the algorithm struggles removing one specific low-rise building. This can be explained by the fact that there are almost no terrain points around the building which makes it really hard to remove. A suggested solution is to use existing map data to remove points inside buildings, but this solution does not solve the problem totally. A way to solve the rest of the problem can be to identify small and isolated clusters of points and classify them as non-terrain.

The re-classification of the point cloud from the Danish Geodata Agency is another method to measure how well the algorithm performs. Where the visual inspection gives a qualitative answer the re-classification gives some quantitative answers. Based on the re-classification, an error matrix has been created and various accuracies and errors were calculated. The accuracies describes how well the algorithm has classified the points. From a producers perspective, the producers accuracy should be high and from the users point of view the users accuracy should be high. By looking at the accuracies seen in Table 21 it can be seen that the users accuracy for terrain points is close to 100 %, which means that the points classified as terrain by the algorithm can be trusted to be terrain in reality. On the other hand the algorithm has a tendency to classify some terrain points as non-terrain which can be seen in the lower user accuracy for non-terrain.

	Producers accuracy	**Users accuracy**	**Overall accuracy**
Terrain	87.2 %	98.4 %	
Non-terrain	94.0 %	63.4 %	
All points			88.5 %

Table 21 – Accuracies found when re-classifying data from the Danish Geodata Agency

From the accuracies, it can be concluded that the final program makes more errors of omission (terrain points classified as non-terrain) than errors of commission (non-terrain points classified as terrain).

Chapter 8: Conclusion

8.1 Introduction

The purpose of this concluding section is to answer the questions in the initial and final problem statement. By answering these questions, the main conclusions and results in this report will be presented. The concluding section is ended with a short discussion about difficulties and ideas for further improvements of the developed DTM algorithm.

8.2 Answer to the initial problem statement

The initial studies were basically about investigating possible solutions to the problems experienced with the surface-based DTM filtering algorithm developed in the previous project by Matthesen and Schmidt (2014):

- Problem 1: Remains of non-terrain points because of buffer
- Problem 2: Large low-rise buildings
- Problem 3: Fixed to one terrain type
- Problem 4: Boundaries
- Problem 5: Discontinuities and sharp edges in the terrain
- Problem 6+7: Memory and speed issues

In order to come up with qualified solutions to the problems listed above, various filtering strategies had to be investigated first, as described in the first question from the initial problem statement:

What filtering methods exist, which can handle a photogrammetrically created point cloud?

A lot of different filtering methods exist, but in general they can be grouped into these four groups:

- Morphological and slope-based filtering
- Progressive densification
- Surface-based filtering
- Segment-based filtering

The methods in each group all have different advantages and disadvantages. This means that one of the methods works well in one area, where another method struggles in removing non-terrain points.

Based on knowledge gained from answering the question above, together with experience from the previous project, it is possible to present solutions to the problems from the previous project. These solutions are found by answering the 2nd question from the initial problem statement:

How can the problems experienced in the algorithm by Matthesen and Schmidt (2014) be solved, and how can the presented filter theory help solving some of the problems?

The investigated filtering methods all have different advantages and disadvantages when the problems should be solved. Suggested solutions were found for all the problems. The slope-based filtering is e.g. good at removing smaller non-terrain objects and can be used to solve problem 1. The method can however not be used to solve problem 2 because the method cannot remove big clusters of non-terrain points and at the same time preserve terrain details. Problem 2 can on the other hand be solved with a surface-based approach. Problem 5 could be solved using progressive densification or segment-based filtering, but these methods have troubles when removing e.g. low-rise buildings.

The choice of filtering method therefore has to be a compromise, where the most important problems are solved, at the cost of some of the other problems.

Problem 6 and 7 about speed and memory cannot really be solved with the filtering method. Both problems can be solved by converting the Python code into Cython and only storing the diagonal of the weight matrix in memory.

Based on the suggested possible solutions, an overall structure for a new and optimized DTM algorithm has been proposed, which can be considered as the answer to the 3rd question from the initial problem statement:

At a general level, how can the solutions to the problems be combined into an optimized algorithm for DTM generation?

A new algorithm has been proposed, which should process the point cloud in *two steps*. A two-step strategy is chosen since some filtering methods are good at removing the big clusters of non-terrain points, while others are good at removing non-terrain points on small objects, such as cars and bushes.

The purpose of step 1 is only to remove the big clusters of non-terrain points, e.g. houses and trees. This is done with an intelligent surface-based filter, which automatically detects the order of the surface polynomial, and the number of iterations needed to outweigh the non-terrain points.

Step 2 is about removing the remaining non-terrain points. After step 1, only smaller non-terrain objects exist in the point cloud and those small objects should be removed with a slope-based approach.

8.3 Answer to the final problem statement

From the initial studies, an optimized two-step algorithm for DTM generation has been proposed. The rest of the report deals with development of this two-step algorithm. Step 1 is about surface-based filtering where the parameters automatically adapt according to the given terrain, as described in the 1st question from the problem statement:

How can the surface-based filtering in step 1 be developed to be intelligent and automatically adapt to the terrain?

Step 1 is split into two sub-steps. First a coarse DSM is generated with a block minimum filter. The block minimum filter ensures a more even distribution of points and makes it easier to filter the big clusters of non-terrain points away since fewer points will be on e.g. rooftops. After the block minimum, a surface-based filter is used.

The surface-based filtering use the coarse DSM and works inside smaller slightly overlapping squares. The surface fitting works by using least squares to fit a surface polynomial to the point cloud and iteratively outweigh the non-terrain points using a weight function.

The surface-based filtering is made intelligent, meaning that the algorithm automatically chooses a polynomial order which fits the terrain. This is done by iterating through different polynomial orders starting from order 0 and continuing until a certain stop criterion has been reached. The stop criterion is based on the standard deviation of unit weight.

For every order, a number of iterations are done where the non-terrain points are gradually outweighed. When a new iteration does not lead to any significant change of the fitted polynomial, the iterations are stopped. An insignificant change is determined by a given stop criterion. Again, the stop criterion is based on the standard deviation of unit weight.

After the best polynomial is iteratively fitted to the terrain, the last part of step 1 is simply to select points from the fine DSM which are inside a buffer around the fitted polynomial.

After the big clusters of non-terrain are removed, the remaining non-terrain points should be removed in step 2. The 2nd question from the problem statement is about how to develop the slope-based algorithm used in step 2:

How can the slope-based approach in step 2 be implemented in a way that removes the non-terrain points and preserve the terrain details?

A slope-based filtering algorithm will run extremely slow if the point cloud is not indexed. Therefore a grid-based spatial indexing is performed before the slope-based filtering is applied to the point cloud.

The slope-based filtering works on one point at a time, the evaluation point. In order to optimally use the slope-based filter at sloped terrain, a solution where plane is fitted to the neighboring points is used. When the plane is fitted, a transformation is performed where the neighboring points are rotated with use of the coefficients of the plane. The 2D distance to the neighboring points are calculated and the height differences are compared to an evaluation function, which defines the maximum allowed height difference as a function of the 2D distance.

For simplicity and speed, it is decided to use a relatively simple evaluation function, which is simply a straight line going though (0, 0). The optimal steepness of the line has been found though a series of tests, and is a compromise between removing the non-terrain points and still preserving the details of the terrain.

With all the details and parameters for step 1 and 2 in place, the next step is to implement the DTM algorithm in a user friendly program, as described in the 3rd question from the problem statement:

How can step 1 and 2 be implemented in a user friendly program, which makes it possible to run the algorithm on a photogrammetrically produced point cloud?

A multi-threaded GUI program called *DTM Creator 2.0* has been created. With the use of the created program, it is possible to generate a DTM based on the project dataset. After the project dataset has been processed, some of the outputted documentation files have been processed by GIS software in order to generate maps which reveal how the used algorithms have performed. In general, it seems like the algorithm has performed as expected.

The result after the project dataset has been processed is that 46.5 % of the points are removed and marked as non-terrain:

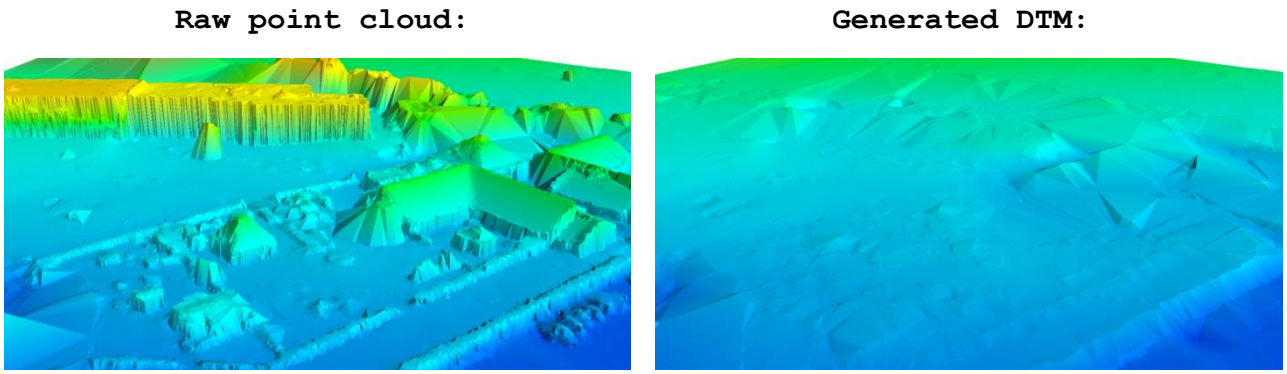


Figure 118 – Points removed with the program DTM Creator 2.0

The two-step DTM algorithm has now been implemented in a program, and a DTM has been created. If the quality of the DTM algorithm is good, the DTM should be almost without errors. This is investigated further, as described in the 4th question from the problem statement:

How good is the quality of the developed DTM algorithm?

The quality of the developed DTM algorithm is investigated in two ways; a visual inspection and a re-classification of a laser scanned point cloud.

The visual inspection showed that in general the program does a good job generating a DTM. However, the algorithm struggles removing one specific low-rise building. This can be explained by the fact that there are almost no terrain points around the building which makes it really hard to remove.

The re-classification of the laser scanned point cloud is another method to measure how well the algorithm performs. The data is already classified as terrain and non-terrain. With the assumption that the classification is “true”, the overall accuracy, the producers accuracy and the users accuracy can be calculated, see Table 22.

From a producers view the producers accuracy should be high and from the users point of view the users accuracy should be high. It can be seen that the users accuracy for terrain points are

close to 100%, which means that the points classified as terrain by the algorithm can be trusted to be terrain in reality. On the other hand, the algorithm has a tendency to classify some terrain points as non-terrain which can be seen in the lower user accuracy for non-terrain.

	Producers accuracy	**Users accuracy**	**Overall accuracy**
Terrain	87.2 %	98.4 %	
Non-terrain	94.0 %	63.4 %	
All points			88.5 %

Table 22 – Accuracies found when re-classifying data from the Danish Geodata Agency

From the accuracies, it can be concluded that the final program makes more errors of omission (terrain points classified as non-terrain) than errors of commission (non-terrain points classified as terrain).

8.4 Discussion

DTM generation is a topic which can be discussed forever, because it is hard or maybe impossible to create a perfect automatic algorithm which can create a DTM based on a DSM. The reason is that the algorithms work in blindness and even though they can adapt to the current terrain type there will be places where the algorithm fails because it is not possible to predict the shape of the terrain.

Even though DTM generation might be a problem which will never be totally solved, this project has led to several topics and questions which can be investigated further. The topics both deals with improvements of the developed algorithm and with DTM generation in general.

The following is ideas regarding the surface-based filtering in step 1:

- **The stop criterions:** The way the stop criterions are used could be reconsidered. In the developed algorithm the standard deviation for an order or an iteration is compared to the previous order or iteration. It could be considered if the algorithm would work better if the standard deviation is compared with 2 or 3 previous values.
- **Stress testing:** The values of the stop criterions are found quite precise in this project. However, maybe the found parameters are only valid in the current project. In order to find out, a stress test could be run on the algorithm to test what happens with the results if the stop criterions are changed. Alternatively, the developed algorithm could be tested on multiple datasets from areas with different characteristics.

These next topics are about improvements in step 2, where the slope-based filtering was used:

- **Indexing:** The indexing used in this project for the search of nearby points could be improved. In the slope-based filtering a radius of 3 m is used. If an indexing grid of 3 m is used for the indexing, the function used to search for nearby points will always have to search in all neighborhood cells, which means 9 cells in all (12x12m). The search

implemented in this project will search in 1-4 cells. One cell is 14 m, which means that, when the search should go through 4 cells the size of the search area is 28x28m.

- **Plane fitting with constraint:** A constraint could be added, saying that the fitted plane should be levelled with a certain weight. This will help avoiding that the plane is wrongly fitted near large non-terrain objects.
- **Slope correction:** In this project the slope correction in section 0 works by using a transformation where all nearby point are rotated into a local coordinate system according to the fitted plane. Another solution could be, for each point, to simply extract the height which can be found using the plane equation found by fitting the plane. This however happens at the cost of preserving the distances measured along the terrain. The advantage is that this method is faster than the one used in the current project.

In general for the developed algorithm these topics could be interesting to investigate further:

- **Handling of low outliers:** The algorithm does not handle low outliers well. If there is a low outlier in the data, the algorithm will remove valid terrain points around the low outlier. The used drone point cloud does not have such low outliers since noise was removed by the photogrammetry software. If the algorithm should be used on a raw laser scanned point cloud, the ability to handle low outliers should be improved.
- **Discontinuities:** The developed algorithm is not good at handling discontinuities since it was neglected in favor of other abilities. This is especially true when the algorithm was used on the laser scanned point cloud. Here it was clear that the algorithm smoothened the terrain and removed discontinuities.
- **Large low-rise buildings:** In the drone point cloud the algorithm has problems removing a certain large building because there were almost no terrain points near the building. This can to some extent be solved with use of some existing map data, where points inside building polygons are removed. It could however be interesting to investigate if the problem can be solved in other ways.
- **Isolated clusters:** It should be considered if small isolated clusters should be classified as non-terrain points. From experience it is found that isolated clusters of points often are measured on top of vegetation in photogrammetrically produced point clouds. Such clusters are hard to classify correct with the developed algorithm due to the lack of neighboring points on the terrain.

In general when generating DTMs, it should be considered if the goal is to develop a perfect automatic algorithm or if the time is better spend with an algorithm which work fine in simple terrain and can point out difficult areas which should be manually sorted or controlled.

As a final note, it should be mentioned that the developed DTM algorithm works not only with photogrammetrically produced point clouds, but also with laser scanned point clouds. For DTM generation, a laser scanned point cloud is preferred. Points will be measured on top of vegetation using photogrammetry, but with laser scanning there is a chance that the last echo of the laser beam has penetrated through the branches and maybe reached the ground.

In this project, the focus was to create a DTM based on a point cloud produced using a small camera drone. However, the Danish Geodata Agency (GST) has already produced a freely available detailed DTM covering the entire country. Since the DTM from GST is produced using laser scanning, it will in general be of much higher quality compared to a DTM produced using a camera drone. It is a limiting factor for the deployment of camera drones that such a high quality DTM already exists. However, the big advantage of the small camera drones are the actuality of the data they produce.

List of Literature

AIM. »AIC B 08/14. Erhvervsmæssig mv. brug af ubemandede luftfartøjer (UAS/RPAS) i Danmark.« *Aeronautical Information Management.* 20. March 2014. http://www.trafikstyrelsen.dk/~/media/Dokumenter/09%20Nyheder/Luftfart/2014/AIC%20B%2008%202014.ashx (senest hentet eller vist den 30. March 2014).

ASPRS online. *LASer (LAS) File Format Exchange Activities.* http://www.asprs.org/Committee-General/LASer-LAS-File-Format-Exchange-Activities.html (accessed May 20, 2014).

Axelsson, Peter. »DEM Generation from Laser Scanner Data using Adaptive TIN Models.« 2000. http://www.isprs.org/proceedings/XXXIII/congress/part4/111_XXXIII-part4.pdf.

Briese, Ch., N. Pfeifer, and P. Dorninger. "Applications of the Robust Interpolation for DTM Determination." 2002. http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.222.2690.

Cambridge University Press. *Cambridge Dictionaries Online.* 2014. http://dictionary.cambridge.org/ (senest hentet eller vist den 1. April 2014).

Cederholm [1], Peter. »Udjævning, 2. udgave.« 2000.

Cederholm [2], Peter. "Matlab script: surfpoly.m." 8 June 2004. http://people.plan.aau.dk/~pce/matlab/surfpoly.txt.

Cederholm, Peter. "Robust adjustment 1 + 2." *Slide from Large Scale Mapping.* 2013.

Chen, Qi, Peng Gong, Dennis Baldocchi, og Gengxin Xie. »Filtering Airborne Laser Scanning Data with Morphological Methods.« 2007. http://www.researchgate.net/publication/216775276_Filtering_airborne_laser_scanning_data_with_morphological_methods/file/9fcfd513dfb0a833ab.pdf.

Chen, Ziyue, Bernard Devereux, Bingbo Gao, og Gabriel Amable. »Upward-fusion urban DTM generating method using airborne Lidar data.« 9. August 2012. http://ac.els-cdn.com/S0924271612001244/1-s2.0-S0924271612001244-main.pdf?_tid=4333c5aa-ed78-11e3-a6bb-00000aacb35f&acdnat=1402058833_b6fe934db2e62e189faf09438e99a321.

Congalton, Russell G. *A Review of Assessing the Accuracy of Classifications of Remotely Sensed Data.* 1991. http://ac.els-cdn.com/003442579190048B/1-s2.0-003442579190048B-main.pdf?_tid=9074d3ae-ed8c-11e3-94b3-00000aacb361&acdnat=1402067552_2b85444c0a4f8977f34a4a0b64303859.

Cython 0.20 documentation [1]. *Getting Started » Cython - an overview.* http://docs.cython.org/src/quickstart/overview.html (accessed februar 23, 2014).

Cython 0.20 documentation [2]. *Tutorials » Basic Tutorial.* http://docs.cython.org/src/tutorial/cython_tutorial.html (accessed februar 24, 2014).

Dalå, Nynne Sole, og Gitte Rosenkranz. *Geoforum: Danmarks Højdemodel bliver bedre og mere nøjagtig.* 16. January 2014.

http://www.geoforum.dk/Admin/Public/DWSDownload.aspx?File=%2fFiles%2fFiler%2fPrsentationer%2fDanmarks+h%26oslash%3bjdemodel+bliver+bedre+og+mere+n%26oslash%3bjagtig%2f20140116_Geoforum.pdf (senest hentet eller vist den 20. May 2014).

Danish Transport Authority. »Nye retningslinjer om erhvervsmæssig brug af droner.« *Trafikstyrelsen.* 20. March 2014. http://www.trafikstyrelsen.dk/DA/Presse/Nyhedsarkiv/Civil-luftfart/2014/03/nye-regler-om-droner.aspx (senest hentet eller vist den 30. March 2014).

Duraiswami, Ramani. *Representing Data on the Computer.* 2009. http://www.umiacs.umd.edu/~ramani/cmsc662/662_Lec4-1.pdf (senest hentet eller vist den 23. April 2014).

El-Sheimy, Naser, Caterina Valeo, and Ayman Habib. *Digital Terrain Modeling: Acquisition, Manipulation, and Applications.* Artech House, Inc., 2005.

Esri. *GIS dictionary.* http://support.esri.com/en/knowledgebase/GISDictionary/search (accessed May 14, 2014).

Geodatastyrelsen. *Kort10.* http://www.gst.dk/produkter-og-ydelser/produktkatalog/topografiske-data/vektordata/kort10/ (accessed June 2, 2014).

Jensen, Karsten. "Integrerede kortlægningssystemer baseret på positionering og attitudeestemmelse ved GPS-måling." June 2005.

Killian, Johannes, Norbert Haala, og Marcus Englich. »Capture and evaluation of airborne laser scanner data.« 1996. http://www.isprs.org/proceedings/xxxi/congress/part3/383_XXXI-part3.pdf.

Kraus, K., and N. Pfeifer. “Determination of terrain models in wooded areas with airborne laser.” 1998. http://ac.els-cdn.com/S0924271698000094/1-s2.0-S0924271698000094-main.pdf?_tid=e548c3f2-7485-11e3-9fee-00000aacb35e&acdnat=1388760597_95e405c40cb01067a1b47cef6bf88907.

Lawhead, Joel. *GeospatialPython.com: Point in polygon.* 19. January 2011. http://geospatialpython.com/2011/01/point-in-polygon.html (senest hentet eller vist den 22. May 2014).

Li, Yong, Huayi Wu, Hanwei Xu, Ru An, Jia Xu, og Qisheng He. »A gradient-constrained morphological filtering algorithm for airborne LiDAR.« 2. July 2013. http://ac.els-cdn.com/S0030399213002235/1-s2.0-S0030399213002235-main.pdf?_tid=d75b2ad4-ed78-11e3-b4b8-00000aacb35e&acdnat=1402059081_181bd63e9ee0ebdfd3aa86e86bbfb171.

libLAS. *libLAS.* http://www.liblas.org/ (accessed June 1, 2014).

Lin, Xiangguo, og Jixian Zhang. »Segmentation-Based Filtering of Airborne LiDAR Point Clouds by Progressive Densification of Terrain Segments.« February 2014. http://www.mdpi.com/2072-4292/6/2/1294/pdf.

Liu, Xiaoye. »Airborne LiDAR for DEM generation: some critical issues.« 2008. http://ppg.sagepub.com/content/32/1/31.full.pdf+html.

Maguya, Almasi S., Virpi Junttila, og Tuomo Kauranne. »Adaptive algorithm for large scale dtm interpolation from lidar data for forestry applications in steep forested terrain.« 21. September 2013. http://ac.els-cdn.com/S092427161300186X/1-s2.0-S092427161300186X-main.pdf?_tid=721380e6-ed77-11e3-acdd-00000aab0f02&acdnat=1402058482_18a.

Matthesen, Anders Westh, og Kathrine Schmidt. »Terrain Modelling – DTM Generation using UAVs.« 2014.

Mongus, Domen, og Borut Zalik. »Parameter-free ground filtering of LiDAR data for automatic DTM generation.« 4. November 2011. http://ac.els-cdn.com/S0924271611001122/1-s2.0-S0924271611001122-main.pdf?_tid=d525dc6e-ed7d-11e3-9101-00000aacb362&acdnat=1402061225_6e4775b1b3417654ded7efb4a7b04680.

Perera, Gamage Sanka Nirodha. »Segment Based FIltering of LASER Scanner Data.« March 2007. http://www.itc.nl/library/papers_2007/msc/gfm/perera.pdf.

Python Shapefile Library. *Python Shapefile Library.* https://code.google.com/p/pyshp/ (accessed May 22, 2014).

SAE International. *Sign convention for vehicle crash testing.* 1994. https://law.resource.org/pub/us/cfr/ibr/005/sae.j1733.1994.html (senest hentet eller vist den 29. April 2014).

senseFly and Pix4D [3]. “Help function in Postflight Terra 3D-EB 2.2.6.” *Manual -> Processing -> Local processing options.*

Shekhar, Shashi, og Sanjay Chawla. *Spatial Databases: A Tour.* 1. edition. Prentice Hall, 2003.

Sithole [1], George. »Filtering of laser altimetry data using a slope adaptive filter.« October 2001. http://www.isprs.org/proceedings/XXXIV/3-W4/pdf/Sithole.pdf.

Sithole [2], George, og George Vosselman. »Comparison of filtering algorithms.« 2003. http://www.lr.tudelft.nl/fileadmin/Faculteit/LR/Organisatie/Afdelingen_en_Leerstoelen/Afdeling_RS/Optical_and_Laser_Remote_Sensing/Publications/Papers/016-2003/doc/Comparison_of_filtering_GeSi.pdf.

Sithole [3], George. »Segmentation and Classification of Airborne Laser Scanner data.« 2005. http://repository.tudelft.nl/assets/uuid:64c82a6e-6db0-4457-9c4f-2446653e5b9d/ae_sithole_20050524.pdf.

Sithole, George, and George Vosselman. *Report: ISPRS Comparison of Filters.* http://www.itc.nl/isprswgIII-3/filtertest/Report05082003.pdf, Delft University of Technology: Department of Geodesy, Faculty of Civil Engineering and Geosciences, 2003.

The Math Forum. *Names of Polynomials.* 1997. http://mathforum.org/library/drmath/view/56413.html.

Tóvári, D., og N. Pfeifer. »Segmentation based robust interpolation – a new approach to laser data filtering.« 2005. http://www.isprs.org/proceedings/XXXVI/3-W19/papers/079.pdf.

Vosselman [1], George, and Hans-Gerd Mass. *Airborne and Terrestrial Laser Scanning.* Edited by George Vosselman and Hans-Gerd Mass. Whittles Publishing, 2010.

Vosselman, George. »Slope Based Filtering of LaserAltimetry.« 2000. http://www.isprs.org/proceedings/XXXIII/congress/part3/935_XXXIII-part3.pdf.

Weisstein, Eric W. *Normal Vector from MathWorld - A Wolfram Web Resource.* http://mathworld.wolfram.com/NormalVector.html (accessed April 28, 2014).

Zakšek, Klemen, og Norbert Pfeifer. »An improved morphological filter for selecting relief points from a LIDAR point cloud in steep areas with dense vegetation.« June 2004. http://iaps.zrc-sazu.si/sites/default/files/Zaksek_Pfeifer_ImprMF.pdf.

Zhang, Keqi, og Dean Whitman. »Comparison of Three Algorithms for Filtering Airborne Lidar Data.« March 2005. http://www.researchgate.net/publication/242705839_Comparison_of_Three_Algorithms_for_Filtering_Airborne_Lidar_Data.

Zhang, Keqi, Shu-Ching Chen, Dean Whitman, Mei-Ling Shyu, Jianhua Yan, og Chengcui Zhang. »A Progressive Morphological Filter for Removing Nonground Measurements From Airborne LIDAR Data.« April 2003. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1202973.