TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technology
Department of Software Science

Eerik Potter 146017IVSM

TOOLSET FOR DATA-BASED EVALUATION AND VISUALISATION OF ROAD SURFACE CONDITIONS

Master's thesis

Supervisor: Martin Rebane
MSc, lecturer

Tallinn 2017

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Eerik Potter 146017IVSM

TEEKATTE TINGIMUSTE ANDMEPÕHINE HINDAMINE NING VISUALISEERIMINE

Magistritöö

Juhendaja: Martin Rebane
MSc, lektor

Tallinn 2017

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Eerik Potter

07.05.2017

Abstract

The goal of this thesis is to analyse and build a toolset for data-based evaluation and visualisation of road surface conditions. The idea grew out of Teede Tehnokeskus's need to better utilize the newly acquired laser mapping system. While the potential of forming a digital model of Estonian roads is promising, it creates new technical problems to be solved, some of them which are addressed in the underlying study.

This study concentrates on multiple aspects. Firstly, there's the visualization of data to get a sense of road characteristics. For this a median-based robust base line is calculated, which allows for relative heights to be formed. Secondly, one of the biggest potentials of having accurate data is the ability to calculate approximations of filling and milling for road repairs. For this, the same base line is used, in addition with the geo-coordinates and trapezoidal approximation.

Going forward, as there are too many data points for later processing, there's a need to compress the laser outputted data. This is a two-sided problem. For one, there's a need to smooth the data to improve compression rate. A combination of different error measurements was used to pick out Savitzky-Golay smoothing algorithm. Following, for compression, the best result was achieved by first pre-compressing with Radial distance algorithm before applying Douglas-Peucker algorithm. To make it more flexible, a custom threshold was implemented.

This thesis is written in English and is 64 pages long, including 5 chapters (with 30 sub-sections), 33 figures, 4 tables and 12 code examples.

Annotatsioon

TEEKATTE TINGIMUSTE ANDMEPÕHINE HINDAMINE NING VISUALISEERIMINE

Käesoleva töö eesmärk on analüüsida ja implementeerida tarkvarakomplekt teekatte tingimuste hindamiseks ning visualiseerimiseks. Esialgne vajadus kasvas välja Teede Tehnokeskuse eelneva aasta (2016) investeeringust soetada endale laseril põhinev kaardistussüsteem. Arvestades, et tänapäeval tehakse jätkuvalt nii teede mõõtmised kui ka projekteerimine käsitsi, on sellise ekspertiisi kulud üsna kõrged. Potentsiaal, mida pakub Eesti teede digitaliseerimine on paljulubav – lisaks kiirele visualiseerimisele ja esmapildile on võimalik automaatselt arvutada ka teeremondi orienteeruvat maksumust. Selleks on vaja lahendada tehnilisi probleeme, nendest olulisematele keskendub käesolev uurimus.

Struktuurselt koosneb töö järgnevatest osadest: sissejuhatus, teoreetiline analüüs, implementatsioon, tulemuste hindamine ning kokkuvõte. Sisult keskendutakse mitmele aspektile: visualiseerimine, freesimis -ning täitmismahtude arvutamine, andmete silumine ning õgvendamine.

Esmalt analüüsitakse teelõigu visualiseerimist, milleks on vaja konverteerida absoluutkõrgused suhteliseks, et tee omadused silmale nähtavad oleksid. Selleks leitakse iga tee profiililõike kohta telg tema otspunktide mediaani vahel. See võimaldab taandada välja tee kalde ning tuua esile rööpad ning muhud. Freesimis- ning täitmismahtude arvutamiseks kasutatakse sama eelpool mainitud telge, geokoordinaate ning trapetsvalemit integraali ligikaudseks kalkuleerimiseks.

Digitaalne mudel erineb käsitsi mõõdetud mudelist eelkõige detailirohkuse poolest. Kui automaatsete kalkulatsiooni puhul on infoküllus pigem hea, siis hilisema 3D töötlemise jõudlus võib kannatada miljonite andmepunktide tõttu. Vähendamaks andmete liiasust oleks vaja andmeid õgvendada, hoides info kao minimaalsena. Uurimuse käigus ilmnes, et andmetes peituv müra vähendab oluliselt õgvendamise tulemust. Sellest tulenevalt jagunes probleem kaheks - vaja on andmeid kõigepealt siluda ning siis õgvendada.

Andmeid saab siluda kahel viisil: töödeldes 1-dimensioonilist signaali (kasutades tee profiililõikeid) või 2-dimensioonilist pilti (kasutades kogu tee maatriksit). Läbi teoreetilise analüüsi sai selgeks, et antud probleemi korral on õigem tee 1-dimensiooniline lähenemine, pakkudes rohkem kontrolli tulemuse üle, ning kaotades vähem detaili. Kombineerides erinevaid andmehulga võrdlemistehnikaid (Hausdorff'i distants, Pearson'i korrelatsiooni koefitsient ning summeeritud ruutviga), jõudis autor Savitzky-Golay filtrini, mis pakkus nii head silumistulemust kui ka vähest viga üle kolme kasutatud võrdlemisalgoritmi. Parimaks konfiguratsiooniks osutus 50 perioodi aken koos 11. astme polünoomiga.

Õgvendamise lahendamisel tuli kõigepealt leida viis maksimaalse andmekao defineerimiseks. Selleks implementeeriti meetod, mis kasutas jällegi eelpool mainitud telgjoont, võrreldes profiililõike pindalade muutusi üleval ning allpool telgjoont. Seades limiidi aksepteeritud pindalade muutusele saame kontrollida andmekadu. Algoritmide valimisel ning implementeerimisel osutus parimaks kombinatsioon radiaalkauguse ning Douglas-Peuckeri algoritmidest, mille tulemusel saavutati 10-15 kordne õgvendus maksimaalselt 1.5% andmekaoga.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 64 leheküljel, 5 peatükki (sealhulgas 30 alampeatükki), 33 joonist, 4 tabelit ning 12 koodinäidet.

Table of contents

List of figures

List of code snippets

List of tables

1 Introduction

Teede Tehnokeskus is a company that provides engineering technical consultation services. Among many areas, they invest heavily in analysing road conditions. Currently, a lot of manual work has to be done by road draftsmen, in order to predict the volume of road milling and filling. As the prediction is directly needed to estimate the cost of a new road or repair, it is crucial that it is as accurate as possible. To achieve this, experience and long manual labour is needed. This approach has several disadvantages. Firstly, it requires lengthy manual measurements. Those measurements are suitable for creating an understanding of the big picture, but they don't map the road in detail. Even in case of best scenario the resulting prediction is quite vague. Secondly, experience is needed, which is expensive and often irreplaceable. Finally, presuming manual model can predict the overall cost in some acceptable accuracy, it doesn't leave behind a detailed model of the road that could be used for visualization and later analysis. Manual measurements lack the detail that more sophisticated techniques could offer.

1.1 Research goal

This study focuses on building a toolset that draftsmen could use to reduce time-costly manual measurements and concentrate on applying intuition and experience on pre-processed and accurate data models gathered by automatic systems. In recent decade, laser systems have become increasingly popular in modelling our surroundings, by offering extremely accurate and detailed mapping of the environment. The people in Teede Tehnokeskus have also noticed that and have started to experiment with those systems. The resulting digital map gathered from lasers holds everything one needs to build automatic and smart decision-making on top of it. That said, while it's a good problem to have, the volume of such data is often grand enough that simple visualization and calculation can get inefficiently slow.

To help boost the efficiency of road draftsmen work and manage the huge volume of data gathered by laser system, this study concentrates on two main areas: toolset for visualisation, including calculations for milling/filling quantities; and data compression

for further manual analysis by 3rd party tools. All of this will be implemented using Matlab 2017a.

Visualising the scanned roads help speed up the analysis and gives a good overview of the road characteristics. The problem with the input data is that the heights are in absolute values, meaning drawing them out raw would not paint a useful picture. There are however many different ways to make sense of the input data. One of the key decisions is to pick the most suitable technique that describes the road in an accurate and flexible way.

Final decisions are ultimately done in a software owned by road draftsmen. For this to work, a reasonably detailed base model has to be imported into the software. As the initial output of the lasers is highly detailed, it would be beneficial if the data were to be compressed without losing too much precision. Significant consideration has to be put into choosing the best algorithm and using it rationally – compression rate is important but so is not losing too much valuable precision. Additionally, for the compression to work, input must not be too noisy. It is likely some type of smoothing operation might be needed before dealing with compression.

Giving an automatic estimation of both milling and filling quantities would perhaps give the best and fastest overview of the road. This means a good understanding of the desired outcome must be achieved in collaboration with Teede Tehnokeskus. From that, an algorithm must be designed to estimate the quantities.

Both visualisation and data compression rely heavily on the quality of validation. Without the means of validating the result, we have no way to grade the quality of the toolset. For validating the visualisation, it is possible to compare the results with a third-party tool that co-exists with the laser system. For data compression however, there is a need to develop a method to evaluate the loss of precision and only accept the result if that loss is smaller than what Teede Tehnokeskus finds acceptable.

1.2 Overview

This work is divided into 3 parts:

- Background research and analysis
- Building the toolset

- Examining and validating the results

Background research dives first into the world of mobile mapping systems to get a better sense of the underlying domain. After that, the analysis of the specific problem can begin to unfold. First, visualization of the road is under scope, followed by comparison of real-time or batch processing. Then, data structure itself is analysed which results in decisions of how to handle the input in different stages of the workflow. Relative heights, needed for the visualization, are covered next, followed by a section for smoothing. When, if any, should the smoothing take place, how to compare different algorithms and what would be the best smoothing algorithm? What follows is an analysis for compression – again, how to differentiate between algorithms and how to use a threshold for controlling the precision for compression. A section about calculating the filling and milling volumes is covered after that. Finally, the last section of theoretical analysis summarizes everything together and forms a basis for building the toolset.

Building the toolset describes the decisions and implementations done on top of theoretical analysis. Code examples referenced from this section can be seen from Appendix 1 - code examples. For all of the problems solved, there are theoretical reasoning, code samples, comparisons and visualizations to aid the reader understand why some of the decisions were made.

The last part of this study, examining and validating the results, combines the work from theoretical analysis and implemented code and takes a step back to evaluate the results. Both the quality and performance are evaluated, leading to the final section about dissecting future work.

2 Background research and analysis

The goal of this paragraph is to first give insight to the domain of this topic and then follow that with background analysis on processing the output produced by the laser system acquired by Teede Tehnokeskus. Mobile mapping and more specifically laser scanning are explained in section 2.1 and 2.1.1. Majority of theoretical analysis is produced in section 2.2, containing six sub-sections: 2.2.1 describes the visualisation of the data, 2.2.2 compares real-time vs batch data processing, 2.2.3 explains the data structure and decisions on how to process data on each stage, 2.2.4 section is about relative height calculation and why is it needed, 2.2.5 is about answering the question whether to smooth the data and if yes, how to pick and validate the algorithms, 2.2.6 section compares different data reduction algorithms as well as validating the quality of the result, 2.2.7 goes over the algorithm derived from communicating with Teede Tehnokeskus to give approximate quantities for milling and filling the road.

Finally, section 12.3 combines all the learned theoretical analysis, describes the workflow to be followed and states a goal for each part of the workflow. Exact solutions at that point might not be clear yet, but theoretical information should be sufficient to continue based on that in paragraph 3, where the most suitable algorithms are chosen and implemented.

2.1 Mobile mapping system

To get a sense of where the data to be processed actually comes from, it's good to first understand the technology of laser systems. Since the development of the first operational land based mobile mapping system in 1991 [1], many systems have been implemented with the aim of increasing both reliability and performances [2], [3]. "A Mobile Mapping System (MMS) allows to determine the coordinates of different points from a georeferenced platform", with the typical components for road surveying and mapping being: a mobile platform, navigation sensors and mapping sensors [4, p. 31].

One of the most common utilizations of MMS is concerned with road mapping, needed for many GIS applications. In some cases a high precision and 3D mapping is required, but often, like for road condition management and maintenance, a 2D mapping may be sufficient [4, p. 31].

"MMS integrates navigation sensors and algorithms together with sensors that can be used to determine the positions of points remotely" [5]. All the sensors are rigidly mounted together on a platform; the former sensors determine the position and orientation of the platform, and the latter sensors determine the position of points external to the platform. The sensors that are used for the remote position determination are predominantly photographic sensors and thus they are typically referred to as imaging sensors [5].

The strength and the reason of growing popularity of MMS lays in their ability to directly georeference their mapping sensors. "A mapping sensor is georeferenced when its position and orientation relative to a mapping coordinate frame is known. Once georeferenced, the mapping sensor can be used to determine the positions of points external to the platform in the same mapping coordinate frame" [6]. In the direct georeferencing done by MMS the navigation sensors on the platform are used to determine its position and orientation [6]. Having a digital model of the surroundings with accurate georeferences gives the ability to automate a wide range of functionalities, often completely eliminating the need for manual intervention, as the detail is far more than a human can process.

2.1.1 Mobile laser scanning

Accurate and intelligent up-to-date roadside information is needed not just for road and street planning and engineering but also for increasing number of other applications, "such as car and pedestrian navigation, noise modelling, road safety, and other planning purposes" [7]. Vehicle-based mobile laser scanning (MLS) is an extension of the previously described mobile mapping system. "The navigation sensors typically include Global Navigation Satellite System (GNSS) receivers and an Inertial Measurement Unit (IMU), while the data acquisition sensors include typically terrestrial laser scanners and digital cameras" [7].

Data provided by MLS systems can be characterized with the following technical parameters [8]:

a) point density in the range of 100-1000 pulses per m^2 at 10 m distance
b) distance measurement accuracy of 2-5 cm
c) operational scanning range from 1 to 100 m.

Based on this information we can conclude that the amount of data produced by such systems is huge (at the rate of 0.25-1 million pts/s), and manual processing of such enormous data is too time-consuming, which prompts a need for automated methods that decrease the amount of manual work required to produce accurate 3D models.

The industry of mobile laser scanners is a growing one and there are already many competitors on the market. In 2013 EuroSDR project "Mobile Mapping - Road Environment Mapping using Mobile Laser Scanning" [7] focused on benchmarking various popular mobile laser scanning systems. The study revealed that high-quality point clouds were generated by all observed systems under good GNSS conditions. "With all professional systems properly calibrated, the elevation accuracy was better than 3.5 cm until 35 m. The best system had 2.5 cm planimetric accuracy even with the range of 45 cm" [7]. The study concluded that overall, the accuracy of all the systems was high. Not coincidentally, laser systems is a globally growing strong business, with a revenue of the laser industry exceeding 10.5 billion dollars last year, despite the slowing global economy [9].

2.2 Processing laser-scanned road surface data

ViaPPS, the system Teede Tehnokeskus uses, is a Pavement Profile System designed to examine the condition of any pavement. "It uses LiDAR technology, a 360° laser scanner, to create a high-resolution 3D point cloud" [10]. The output this study has available for use consists of 4 fields: longitude, latitude, height and light. Of those four, the most interesting work can be done with the height measurements.

The laser system also comes with a separate software called ViaPPS Desktop Analyzer. Although the software itself is quite flexible, offering multiple ways to smoothen the data and make different reports, it's more useful for getting a sense of the data, rather than doing any serious work. The road is visualised in 3D and in short segments, shown on the left side of Figure 1. It also displays profile cuts on the right side, which are good for giving the user a sense of the road but not for any complex work as it displays the segments and is quite slow in moving from one segment to the next.

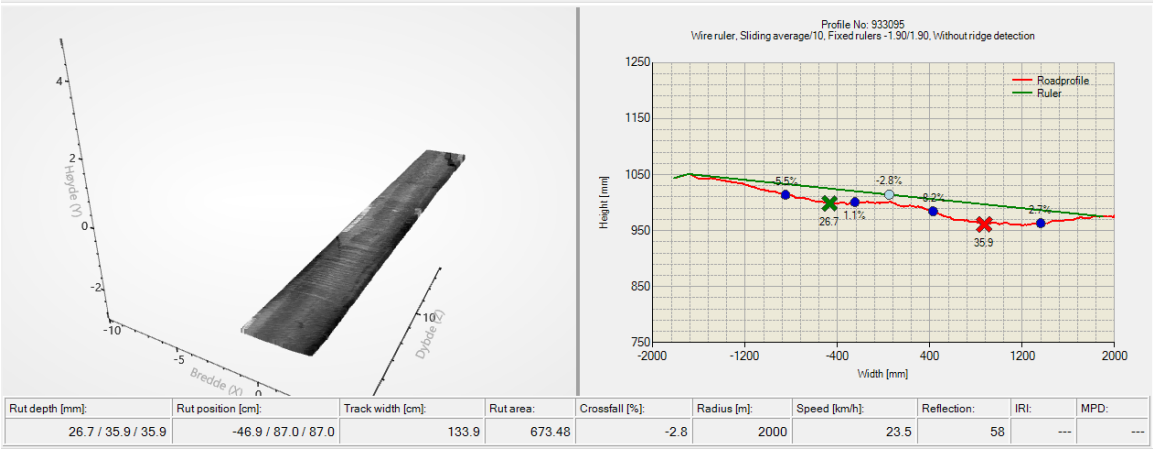

Figure 1 ViaPPS Desktop Analyzer

The initial raw data coming from ViaPPS is hidden in binary files. There are two ways to extract useful data in a better form, that could be used for external processing. One is using a functionality to output data from ViaPPS Desktop Analyzer to clipboard. This has an unfortunate limit of approximately 350m at one go, as over that crashes the application. That's around where Windows default clipboard also reaches its peak, which could be changed by increasing the virtual memory on the system [11]. However, as the software itself is not able to copy more than 350m of data (which is approximately 3.6-4 million rows) then modifying the virtual memory does not help. Other way to extract data is to use a 3rd party tool that combines those binary files together and outputs a readable format. This is costly, but has an added benefit of calculated geo-references in the data as well, instead of relative X, Y, Z coordinates like shown in Table 1. In either case, the output could be used later by custom software to apply compression and display the data in a more convenient 2D form.

X-Value	Y-Value	Height	Light intensity
-3.862617728	-1823.864149	1041.813668	148
-3.862617728	-1811.555809	1044.667022	153
-3.862617728	-1799.752759	1044.727402	154
-3.862617728	-1791.451996	1044.32111	157
-3.862617728	-1782.787008	1043.457226	156

Table 1 Output segment from ViaPPS Desktop

2.2.1 Visualising scanned road

3D reconstruction of road surfaces and surroundings enables to capture all the detail digitally for further processing with minimal work. While it's extremely useful in some areas, like street scene mapping and autonomous vehicles, it's also quite resource consuming as suggested by a paper on real-time 3D road modelling [12]. With high detail (mm precision) even a relatively short road segment produces a lot of data. For a 200m road segment the ViaPPS system produces approximately two million rows of data. For a user-friendly 3D model, that many data points puts a lot of strain on the GPU [13]. Additionally, as the height does not change much, the Z-coordinates in a 3D form might not be visually easy to comprehend.

2D scatter plots for road pavement visualisation capture the same amount of detail with fewer dimensions, reducing both the rendering speed and the memory consumed. The height difference between points can be visualised with colour. Also, the need for rendering the graph multiple times is eliminated, something that is necessary when rotating the 3D graph. Taking this into account, the 2-dimensional visualisation is the right path to go to get a sense of the data and visually validate processing algorithms.

2.2.2 Real-time streaming or batch processing

This study concentrates on how to help make sense of the data produced by lasers, while also cleaning and compressing it for further analysis. 2D visualisation is relatively cheap compared to 3D, which uses more memory [14]. In fact, "GPUs generally approach 2D image processing as a restricted form of 3D" [15]. Taking this into account eliminates the necessity of building any real-time processing capability. Meaning, that in a sensible realm, algorithmic complexity for us is not a major concern, letting us comfortably choose quality over speed.

2.2.3 Data structure and handling

One of the initial outputs from the laser was briefly shown in Table 1 Output segment from ViaPPS Desktop. To more conveniently process the data, it is better to reshape it to a 2D matrix, with the Y-axis representing data for a single profile (containing the geo-coordinates, height and light) and the X-axis contains all the profiles. This gives us a more logical data structure and also allows us to use optimized matrix operations.

Having the data in a matrix form allows us to choose between applying algorithms on 1-dimensional data, so called profile cuts, or 2-dimensional data. This decision should be made in each stage of data processing separately.

a) For the creation of relative heights there is a clear need to process every profile separately, as different base lines are required for every profile cut. Some, if not all, operations could be done on the whole dataset using matrix operations for a more optimized solution.

b) For data smoothing it is not clear in regard of the research aim of this thesis which way is better. From one end, applying smoothing on 1-dimensional data, also called "line smoothing", allows more fine-grained control over the input signal. Then again, handling the data as an image allows us to process it more efficiently, which will probably not be the main concern. Both ways will be explored, if and when the smoothing seems necessary at all.

c) For data reduction, the goal is to reduce individual data points. As every profile is different there is also a need to handle them differently. Some profiles might have deep rails and much detail contained in the data that cannot be removed, others might have smooth surface and could be reduced significantly. Also, as the verification of the algorithm will highly likely use 1-dimensional algorithms, then the whole compression should be treated in a same manner.

2.2.4 Relative height calculation

As briefly discussed in 2.2, there are two different types of output coming from the laser scanning system. One uses paid 3rd party tool to apply additional calculations and produces absolute values: longitude, latitude and height from the sea level. The other is simpler and more relative: both x and y coordinates are relative to the road and the height is relative from the base, which is approximately 10m below surface. First one uses the full potential of the mobile scanning system, the second one has the information without the georeferences. In both cases, however, we might as well consider the height being an absolute value. This creates a problem regarding further processing: road height can easily change a couple of meters even in the span of a short road segment, creating more unique values and making the visualisation of the road problematic.

The main gain of the digital scan is to accurately measure and visualise the size of the rails in the road, in order to estimate the quality and the wear down. Those rails are measured in cm. This creates a need to convert the heights to a new base which takes into account road characteristics and would allow us to visualize the road even when the absolute height changes considerably.

To achieve that there is a need to work on individual profile cuts. There are multiple ways of calculating relative heights based on information from Teede Tehnokeskus.

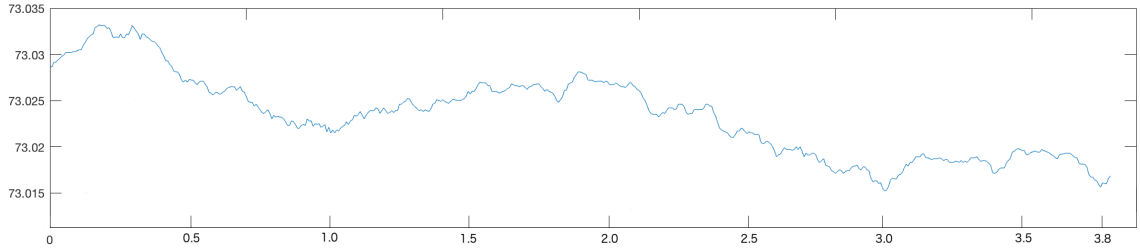

Figure 2 Sample profile cut

Let's first look at a sample, pictured in Figure 2 (profile after applying a moving average for better visual experience), profile cut of highway Jõhvi-Tartu-Valga with coordinates 58.0106261, 26.1628515. The road is 3.8 meters wide and the absolute height ranges between 73.015 and 73.035 meters. It's also intersecting with a road Võru-Kuigatsi-Tõrva, which is not scanned but does affect the characteristics of the road. Following are some of the ways of eliminating the need for absolute height values.

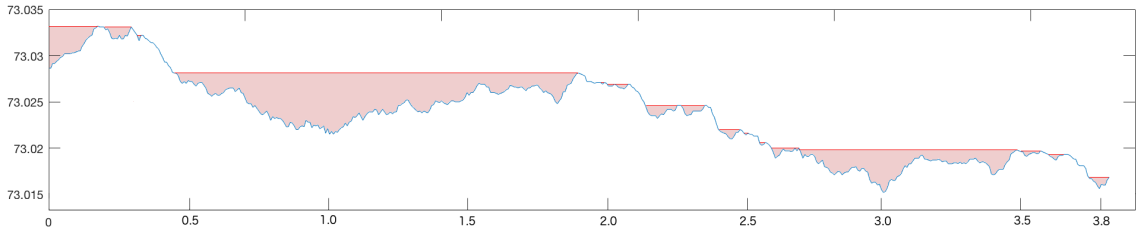

Figure 3 Relative height calculation - water method

Figure 3 shows the water method, which is the nature's way of finding all the holes in the road. While good at that, it lacks a way to detect bumps on the road. It's also more tedious to implement as it consists of many lines instead of one. Relative heights that form from drawing the holes are also inconsistent as the area between holes are unused.

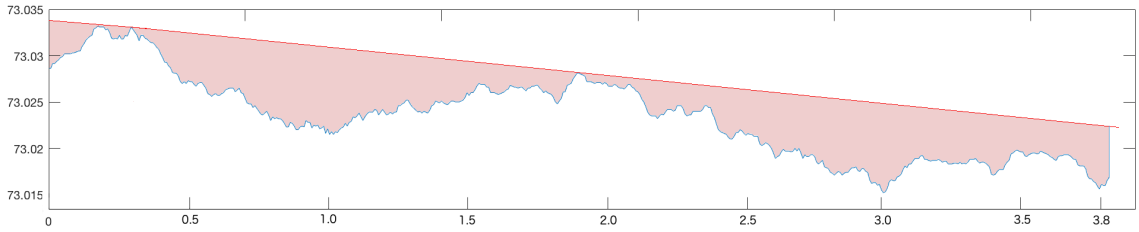

Figure 4 Relative height calculation - ruler method

Ruler method, shown in Figure 4, is the standard way of measuring the road manually. A long ruler is placed on the road and everything that is between the ruler and the road is considered the relative height. Having been in use manually, this serves as a good technique to either use, or to compare against.

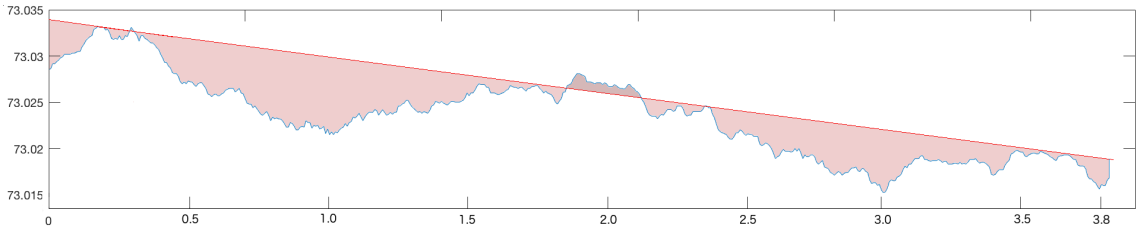

Figure 5 Relative height calculation – string method

String method takes the peaks from both sides of the road and connects them with a line, shown in Figure 5. The advantage of this method over the ruler method is that bumps in the middle of the road, while still being considered, don't have a big impact on the line itself. As the connection points are always at the sides of the road instead of dynamically changing depending on the bumps, this produces more stable relative heights. As the relative heights are calculated per profile there is a need to find a stable reference point so that all the profiles would be similarly adjusted.

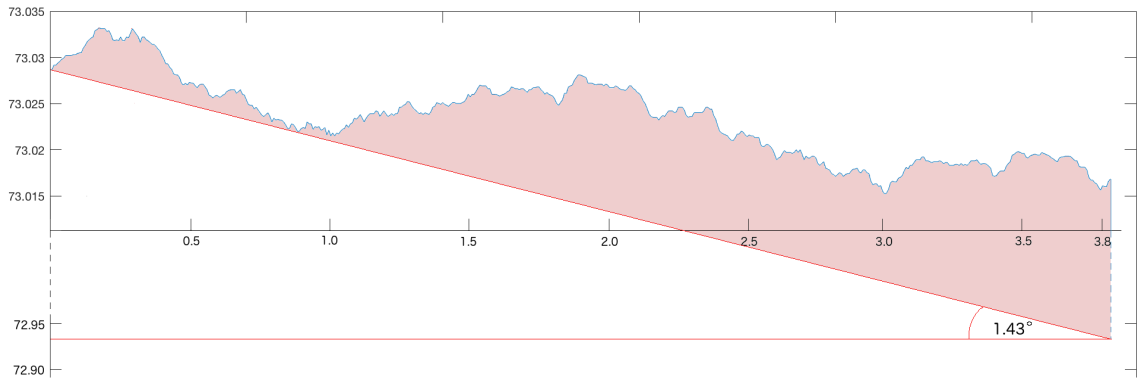

Figure 6 Relative height calculation - angle method

Finally, Figure 6 shows the angle method. The ideal angle of a straight road is 2.5%, which is 1.43°. This could be taken as a base for calculating relative heights. This would work very well if the angle would be constant, however road conditions do change. During corners the angle might increase several degrees, also during intersections the angle is generally smaller. This can be seen from the figure above which shows a road segment with a low angle due to intersection.

2.2.5 Data smoothing

The question whether to smooth a data or not proves to be trickier than initially seems. The fact that a laser produces very detailed measurements and the roads are uneven means that the resulting heights are noisy, at least viewed by a human. Question arises whether we should smooth the data and if yes then what should this procedure accomplish and when and how should it be done.

Should we smooth data? There are generally three reasons to choose smoothing:

a) "for cosmetic reasons, to prepare a nicer-looking or more dramatic graphic of a signal for visual inspection" [16]

b) "if the signal will be subsequently analysed by a method that would be degraded by the presence of too much high-frequency noise in the signal" [16], for example when a data holds key points needed for further processing (like relative heights calculation or approximating the size of areas). Optimization of the amount and type of smoothing is important in these cases as eliminating the noise has the ability to leave the essential information behind

c) there's also the fact that by smoothing the data, we need less unique points to represent the values, which allows to compress the initial input often multiple times. "As ways to capture data increases by the day in today's digital world the problem regarding data storage grows as well. Data compression using smoothing filters is a technique to store large amount of data in a limited storage device" [17]

Generally, if a computer is available to make quantitative measurements, it's better to perform them on the unsmoothed data, rather than graphical estimates on smoothed data [16]. For the visualization of the road, it's reasonable to assume that by smoothing we lose some of detail, thus reducing the quality of the digital model of the road. As Figure 7 shows, there is a slightly noticeable difference in the visualization of raw versus the smoothed road (using the simplest smoothing filter, unweighted moving average).

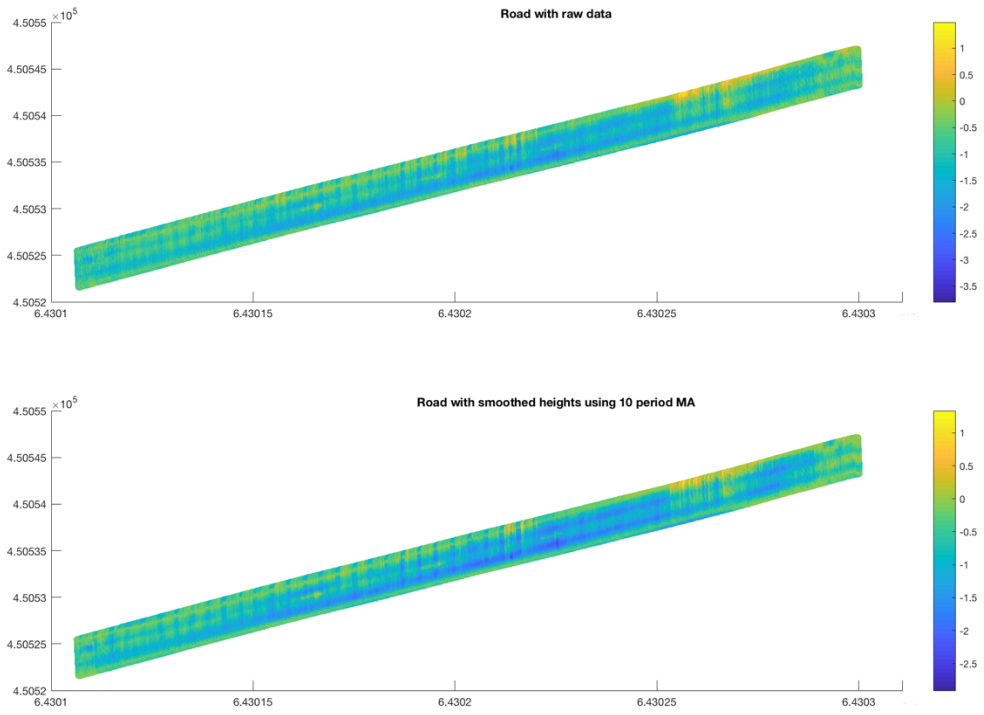

Figure 7 Road with raw input vs smoothed heights using 10MA

While some minor loss of quality might be acceptable, it does not add any value, thus there is no need to smooth any data for visualization purpose.

Smoothing to aid compression depends on the algorithm and the need remains to be seen at the current moment. Intuition dictates that a smoother line would be a better

candidate for compression, as noise could be used incorrectly and thus resulting decreased quality in the compression. At this point the compression part hasn't been analysed yet, but there is a need to try out a sample compression to get a sense how it is affected by the smoothing factor. Figure 8 pictures Gaussian filter (a two-dimensional smoothing filter) and Douglas-Peucker (line compression algorithm): as can be seen, smoother the graph, better the compression rate. Logic supports that - noisy data has rapid changes between neighbouring elements, thus it's more difficult to spot a trend and eliminate useless elements. This comes with a cost, as we smooth the data we lose some of the detail that we can't get back.

Therefore, it's reasonable to assume even from this sample alone, that smoothing for compression is valuable and there is a need to dive more into this topic.

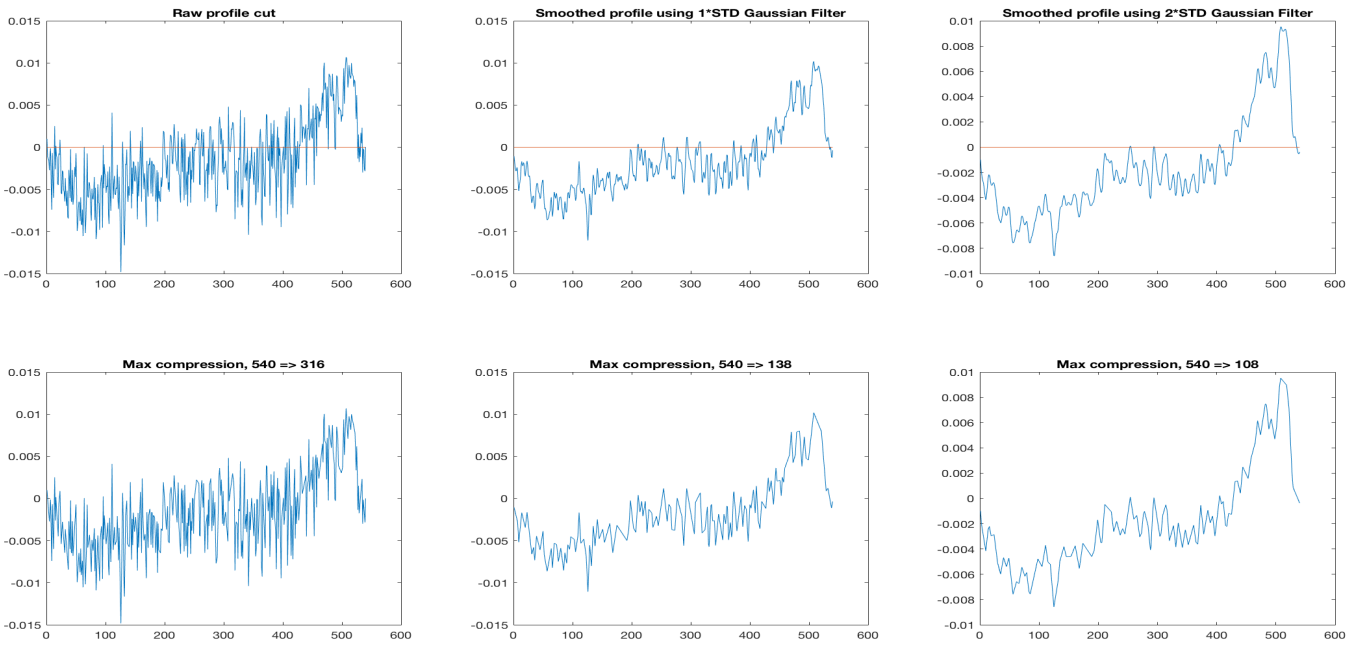

Figure 8 Compression (using Douglas-Peucker line simplification algorithm) for raw data vs smoothed

What should the smoothing procedure accomplish?

There are three key points to consider:

a) Eliminating outliers is necessary for any further algorithms, including smoothing: outliers hurt both smoothing and data reduction algorithms

b) Smoothing is necessary for the compression to work well, otherwise neighbouring elements are too far from each other and compression algorithms will suffer from it
c) Smoothing should not lose road characteristics too much

These points indicate that our goal is to remove outliers and to determine a good estimate of a smoothing algorithm for the compression to perform well.

When should we smooth the data?

Leaving the outlier removal aside for a moment, there are two places we could smooth the data: before calculating relative heights and before reducing data points. To help make the decision let's plot the relative heights of a sample profile cut using the string method.

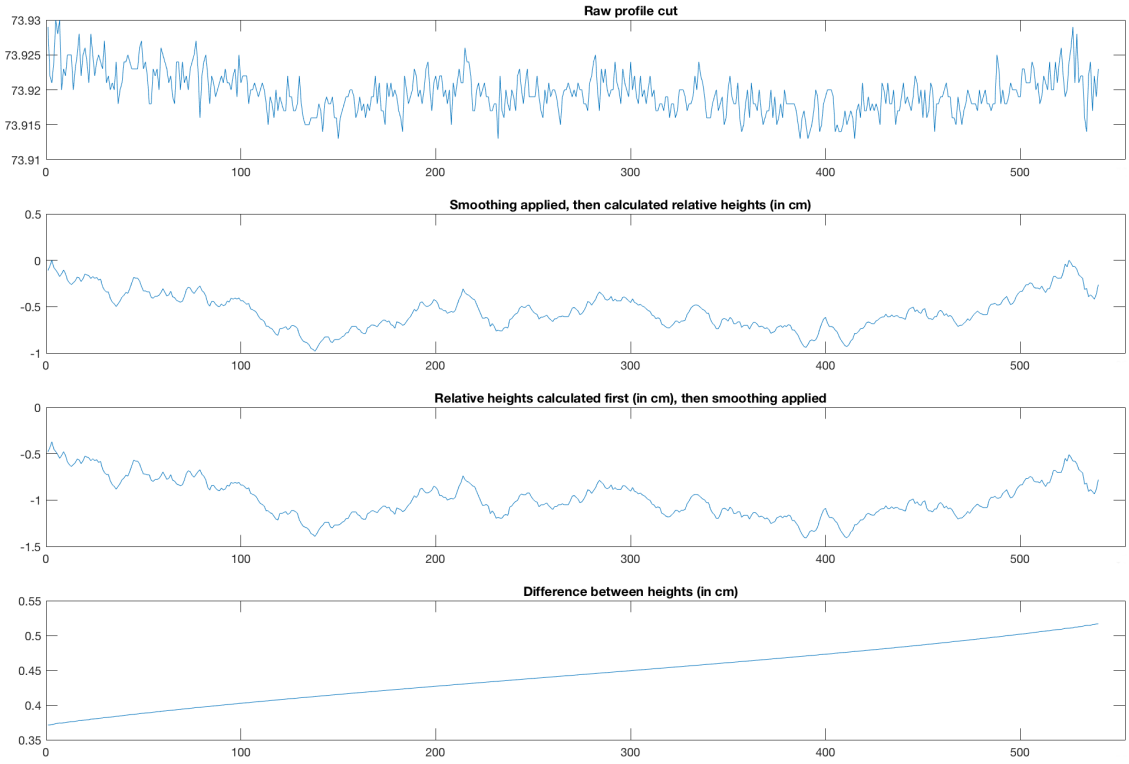

Figure 9 Smoothing before vs after relative height (string method) calculation

As can be seen from Figure 9, the overall characteristics of the road remain the same judging by the eye. However, as the base line for relative heights differ before and after applying the smoothing techniques, the resulting heights are different as well. The importance of relative height calculation in both visualization and data reducing is

major. In addition, smoothing definitely loses some detail. While it is crucial that relative heights calculation is not affected by noise in the system, smoothing is not the best way to make it robust as it reduces the richness of the data. Therefore, we shouldn't apply smoothing before calculating relative heights. Resulting educated guess would be to postpone the smoothing until compression stage and calculate relative heights on the original data.

If can't be done with the smoothing itself, we additionally need to remove outliers before smoothing process. This leaves with the following work flow: calculate relative heights, remove outliers and smooth, compress.

How to smooth the data – 1D vs 2D?

As described in 2.2.3, our data structure enables us to apply algorithms in two ways: using vector (profile) or matrix. At this point it remains unclear which is better and more suitable, so both ways are explored. Smoothing 1-dimensional data is called "line smoothing", while 2-dimensional data allows us to view it as an image and apply image smoothing algorithms.

2.2.5.1 Line smoothing

"Approximation theory is an established field in mathematics that among many things deals with solving a problem of smoothing a line. The idea of smoothing is to create an approximation function(s) that attempts to capture important patterns in the data" [18]. The immediate results of smoothing process are the produced values, not so much the functional form which may or may not be used later. Compared with a similar concept of curve fitting, the aim is to approximate the general changes in data, rather than achieve as close of a match with the data as possible [19]. Those ideologies however are very closely linked in both literature and practice and are often used intertwinely. There are a lot of different ways how to achieve smoothing and to analyse them all would fall out of the scope of this study. After surveying the literature, intuition is used in the selection of some key algorithms from a wide range with the goal being to try something from every angle.

A lot of smoothing algorithms are based on the same "shift and multiply" concept, where a group of adjacent points in the original data are multiplied point-by-point by a

set of coefficients that defines the smoothed shape. Those products are added up and divided by the sum of the coefficients, while becoming one point of the smoothed data. Then the coefficients are shifted by one point and the process is repeated [20, Ch. 15]. This suits us well, as the group is adjustable by us and gives control in filtering out the outliers.

Moving average filter is one of the most commonly used and easiest filters. "The idea behind using moving averages for smoothing is that observations which are nearby in time are also likely to be close in value" [21]. The average eliminates some of the randomness in the data, leaving a smooth trend component. There are two ways to use a moving average: two-sided moving averages are used to smooth a time series in order to estimate the underlying trend; one-sided moving averages are used as simple forecasting methods for time series [21], often used in aiding financial predictions. Dealing with a road conditions domain, I care more about smoothing than estimating.

In equation form, a moving average is written in the following form:

$$\mathrm{f(t)} = \frac{1}{2k+1}\sum_{j=-k}^{k} y_{t+1}\,, \quad t = k+1, k+2, \dots, n-k$$

Equation 1 Moving average filter, where f(t) is the smooth function, y is the input signal and k is the number of points used in the moving average

"In spite of its simplicity, the moving average filter is optimal for a common task: reducing random noise while retaining a sharp step response". This makes it the premier filter for time domain encoded signals [20, Ch. 15]. Downside with the moving average is the sensitivity to outliers, meaning there is a need to remove outliers when this option is chosen. The so called smoothing average "can also be interpreted as a local linear regression with a rectangular kernel" [22], which assigns equal weight to each point in its window. Local is meant in a sense, that a fixed size of a moving window is decided beforehand and the function is applied only to this window not all of the data. We'll use this concept many times throughout this work. We could however presume that adding weights to be used inside the kernel would help. The road conditions can change rapidly but it's only natural to think that for every measurement it would be closer to the previous one than for a distant previous one. This brings us to the weighted moving average.

Weighted moving average with k-point window can be written as

$$f(t) = \sum_{j=-k}^{k} a_j y_{t+j}$$

The advantage and reason to prefer weighted averages is that the resulting trend estimate is much smoother. Instead of observations entering and leaving the average abruptly, they can be slowly down-weighted. “There are many schemes for selecting appropriate weights - those values are not chosen arbitrarily, but because the combination of moving averages can be shown to have desirable mathematical properties” [21]. While choosing our own weights for the moving average algorithm adds flexibility and freedom, it also takes time and careful calibration. Road conditions are constantly changing, which creates a danger to overfit the data. If a more dynamic algorithm compared to the simple moving average is needed, this should be achieved by a more generic way, without specifying tuned constants fit to sample data.

A better and more dynamic filter for smoothing a signal is called **Savitzky-Golay filter**, which grew out of a need to smooth noisy data obtained from chemical spectrum analysers [23]. This method could be thought of as a generalized moving average, with the “filter coefficients derived by performing an unweighted linear least squares fit using a polynomial of a given degree” [23]. Compared to the sliding-average smooths, the Savitzky-Golay smooth is less effective at reducing noise, but more effective at retaining the shape of the original signal. The filter uses a span of neighbouring elements x and fit a polynomial of order k to these points. Then the point at the centre of the x point is replaced by the value of the polynomial of this point. Therefore, the smoothing is stronger for higher x and smaller k [24], making it easily configurable. In general, higher degree polynomials can more accurately capture the heights of narrow peaks, but could do poorly at smoothing wider peaks. “Savitzky and Golay's paper is one of the most widely cited papers in the journal Analytical Chemistry” [25].

Previously described weighted moving average could also be denoted as a locally weighted zero-degree polynomial regression [26]. By using a weighted linear least squares regression over the span of the values (window), we get a **Lowess** regression method, derived from the term “locally weighted scatterplot smoothing”. Using a locally weighted quadratic least squares regression will produce **Loess** regression. At each point in the range of the data set a low-degree (usually 1 or 2) polynomial is fitted

to a subset of the data. “Least squares denotes that the overall solution minimizes the sum of the squares of the “error” made in the results of every single equation” [25]. Both lowess and loess are non-parametric flexible regression methods that do not require a function to fit a model to all the data [26].

If data contains outliers, the smoothed values can become distorted and not reflect the behaviour of the bulk of the neighbouring data points. To overcome this problem, it is possible to smooth the data using robust procedure that is not influenced by a small fraction of outliers. Both loess and lowess have a robust version, which include additional calculations to determine robust weights. This robust smoothing procedure follows four steps [22]:

a) Calculate residuals from the smoothing procedure
b) Compute robust weights
c) Smooth the data again using robust weights – the final smoothed value is calculated using both local regression and robust weights.
d) Repeat the previous two steps for a total of five iterations

This is suitable for us, but not having a robust version of the algorithm should not be a problem either. There is also a possibility to remove outliers separately from the smoothing process described in 2.2.5.3, eliminating the requirement for robustness in choosing smoothing algorithm.

2.2.5.2 Image smoothing

It is worth mentioning again that the initial data of heights is not only detailed but also, depending on the road, can have a wide range of heights during a single measurement. For the line smoothing this is not important as the range of data during a profile is minimal. For the 2-dimensional algorithms this makes the smoothing on raw data complicated, as an outlier in one profile, can be an average height 100m later. The holistic view of the range of unique values is too wide. After calculating relative heights however, we reduce the range of heights by approximately 2 orders of magnitude, making the data more suitable for image smoothing.

Generally, “image smoothing is synonymous with low pass filters, having the same ideology with single dimension processing” [27]. Image processing applications are

different from filter processing applications because most of them are tuned to the eye. Additionally, smoothing can be applied both horizontally and vertically.

The world of image smoothing algorithms is wide and complex. Just to name a few algorithms out of many: L0 gradient minimization, local Laplacian filters, edge-preserving decompositions, bilateral filter, nonlinear total variation based noise removal, weighted least squares, tree filtering, edge-avoiding wavelets, diffusion maps, gradient weighting filtering, image convolution, etc. [28], [29].

Before diving into any of them, it is necessary to understand why and when would one smooth an image at all. This gives intuition whether to choose 2D smoothing algorithms over the ones described in the previous section 2.2.5.1. The need to smooth and image can derive from a couple of different reason. It can be used to blur an image, remove detail and noise.

For the problem and the toolset this study concentrates on, we have no interest in blurring the produced image. The only reason we need smoothing is to aid compression efficiency, blurring is an unfortunate side effect. We also don't want to remove any specific detail from the data. "Noise in images can be anything that does not interest the user of the image, like light fluctuations, finite precision, sensor noise, white noise" [30]. This differs from noise in our data, which isn't really noise, it's the data itself. Road conditions are naturally uneven, which the laser captures in a detailed matter.

The added effect of 2-dimensional processing would serve yet another bad side effect for us, as it introduces a second dimension for data points. This means a data point, which needn't be smoothed from a profile perspective, could be smoothed because of neighbouring data points in close proximity. As the compression algorithm processes every profile separately we will lose detail where it's not needed. Taking all this into account, I have decided not to continue with image smoothing and concentrate on line smoothing instead.

2.2.5.3 Removing outliers

The term outlier seems to be used rather informally in the literature. Barnett and Lewis define an outlier as "an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data…" [31], which is quite a subjective

statement. What is objective, is the fact that smoothing algorithms in simplest form are sensitive to them, therefore it is recommended, if possible, to eliminate them beforehand. Applying relative height calculation results in a somewhat fixed range of heights, meaning the unique values in the whole dataset is relatively small for the calculated heights (at least compared to the original input). Relative heights should approximately range from -10 to 10 cm.

The simplest way to remove an outlier is to use a filter to check whether there are any values that are some user defined standard deviations away from the mean. This would normally work, but for current case it's not sophisticated enough as our input signal can vary quite significantly in the span of one profile cut.

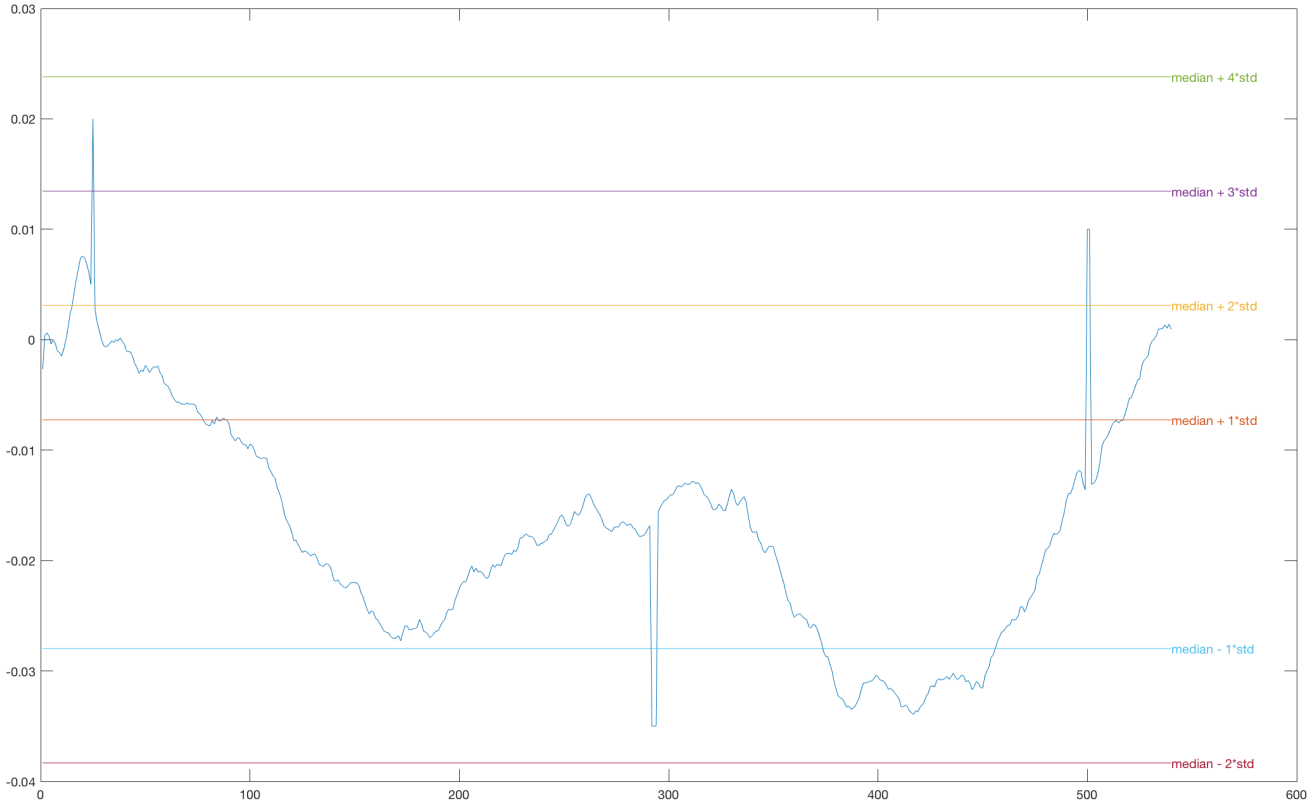

Figure 10 Profile with artificial outliers with standard deviation levels

Figure 10 shows that there does not exist a good global level, even for a one-dimensional profile, that would allow us to remove outliers beyond it. Eliminating everything further from median than one standard deviation would eliminate a lot of data. Choosing two standard deviations or anything above would leave outliers in, anything in between might work for current profile but suffers from overfitting. There is

a need to process data more locally. Luckily, there's an algorithm that exactly follows that intuition, called Hampel identifier.

"Hampel suggested the median and the median absolute deviation as robust estimates of the location and the spread" [32]. For each sample of data X, the function computes the median of a window composed of the sample itself and its neighbouring elements. It also estimates the standard deviation of each sample about its window median using the median absolute deviation. If a sample differs from the median by more than some user defined standard deviations, it is replaced with the median [32], [33]. This works for us as both the window and differing from the median is configurable, which is all we need.

2.2.5.4 Comparing and validating smoothing algorithms

There are two characteristics of a line we can measure in order to evaluate a smoothing algorithm: one is the smoothness of the line, the other is the difference between raw and smoothed line.

Easiest way to get a sense of how smooth a line is, is to calculate the differences between adjacent elements along the profile. By taking an absolute value of those differences gives us a change for every element for our profile data. To get a smooth factor of a line we could take an average or a median of those differences. It's better to use median, as our data might be characterized by skewed distributions. When comparing the result across multiple smoothing algorithms, then a smaller smoothing factor indicates a smoother line. Thus, we get the following formula:

$$S_i = median(abs(diff(f(Y_i))))$$

Equation 2 Smooth factor formula, where S_i is the calculated smooth factor for a line, f is the smoothing algorithm, *diff* calculates differences for adjacent elements, Y is the vector of heights and i is the index of the profile

From one end, we would like to smooth the data as much as possible, as it aids the compression process. From another, we don't want our data to lose a lot of detail and separate from the raw input. For this, there is a need to compare the smoothed data to the original, or to compare the similarity between them. This is a studied topic in mathematics called trajectory or curve similarity. Similarity, or dissimilarity is a wide notion in itself and depending on the goal, different techniques can be used. The goal for evaluating smoothing is to verify that the overall trend of the signal didn't change

too much. Similarity score is not difficult to measure *per se*, but there are a lot of different choices to choose from.

There are some criteria we can define to make the choice of similarity algorithm easier. First of all, we have one dimensional time-series data. Data itself resembles a signal, which is a polyline that does not self-cross. We don't expect our smoothing to result in any time or space shifting. In addition, without any algorithms, we can already know that our smoothed line is definitely "similar" to the original, as it's just smoothed, the range of values are just slightly smaller. Therefore, the goal is to find a distance between our original profile and the smoothed line. This is done via distance functions.

Sum of absolute difference (SAD) also known as Manhattan-norm is the sum of the absolute differences of the data pairs. Using the absolute function makes this metric a bit complicated to deal with analytically, as our smoothing algorithm will smooth in both ways. As this simple algorithm does not use any quadratic formulas, it is quite robust in dealing with the outliers [34].

Sum of squared difference (SSD) is equivalent to the squared Euclidean distance. This is the fundamental metric in least squares problems and linear algebra. The absence of the abs function makes this metric convenient to deal with analytically, but the squares cause it to be very sensitive to large outliers [34].

There are many algorithms that are built on top of those two, often some sort of normalization is applied. For example, mean-absolute error (MAE) is the normalized version of SAD, mean-squared error (MSE) is the normalized version of SSD, normalized mean square error (NMSE) takes MSE and normalizes it by the original data. Euclidean distance is the square root of SSD. As our lines are in the same range, we don't need to normalize our data in any way. Therefore, those modifications will not be taken into account.

The Pearson distance is a correlation distance based on Pearson's product-momentum correlation coefficient of the two sample vectors [35]. "A correlation coefficient is a number that quantifies a type of correlation and dependence, meaning statistical relationships between two or more values in fundamental statistics" [36]. Pearson correlation coefficient is a widely-used way of measuring the strength of linear association between two variables. It is calculated by dividing the covariance of the

variables by the product of their standard deviations [35]. The algorithm is not robust, so it's sensitive to outliers, but we expect to remove those outliers before smoothing anyway.

Hausdorff distance works differently as it doesn't average or summarize anything, rather it measures how far two subsets of a metric space are from each other. "Informally, two sets are close in the Hausdorff distance if every point of either set is close to some point of the other set" [37]. It measures the longest distance forced to travel by choosing a point in one of the two sets, from where you then must travel to the other set. "In other words, it is the greatest of all the distances from a point in one set to the closest point in the other set" [37].

There are many more ways to calculate the difference between curves. Some don't suit the current problem well (Fréchet distance), some do too much (Procrustes distance), some are unnecessary for our data (dynamic time warping). I think to achieve the requirement for comparing smoothing algorithms, the previously described ways to find the difference between our profile lines are sufficient. Which one to use will be decided on section 3.3, where the eventual smoothing algorithm is chosen.

2.2.6 Reducing data points

Data reduction is the process of minimizing the amount of data that needs to be stored in a data storage environment. Data reduction can increase storage efficiency and reduce costs. "Data compression reduces the size of a file by removing redundant information from files so that less disk space is required" [38]. This differs a little for the toolset at hand. The goal of compression Teede Tehnokeskus is interested in isn't necessarily to reduce the size of file, by reducing the unique values. Smoothing helps in that regard. What is needed is to remove as much data points as possible without losing much detail to aid post-processing efficiency. As explained in paragraph 2.2.3, this should be accomplished by processing profile cuts, instead of the whole data set, due to the conditions of the road.

2.2.6.1 Compression with curve simplification

According to an experimental study with GPS trajectory data [39] that compares the major compression approaches, "no approach outperforms others under all scenarios as

they have their pros and cons". With the GPS trajectory observed in that study, there is a dimension that is absent in road conditions data – time. Our laser produced profile is a line, not historically traced points. Word-for-word, the produced points are of course historical traces, but as the time is constant between every point, we can eliminate that dimension. As a consequence, "the use of perpendicular distance, shortest distance between a point and a line, as condition for compression is optimal" [40], allowing to reduce the scope of the compression problem.

Reducing the scope from general compression to a one-dimensional time-series line simplification allows us to concentrate on a few algorithms that are most suited for reducing excess detail from road condition data.

Radial distance is a simple brute force linear complexity algorithm for polyline simplification. "It reduces successive vertices that are clustered too closely to a single vertex, called a key. The resulting keys form the simplified polyline" [41]. The first and last vertices are always part of the simplification, and are thus marked as keys. Starting at the first key the algorithm walks along the polyline. Succeeding vertices, which fall within a user defined distance tolerance from that vertex are removed. The next vertex which lies beyond the tolerance is kept and marked as a next key. This is then repeated until it reaches the last vertex.

Douglas-Peucker algorithm is often considered as the most precise line simplification algorithm, that uses a recursive divide-and-conquer approach. The algorithm is widely used in robotics to perform simplification and de-noising of range data acquired by a rotating range scanner [42]. "It was originally proposed for line simplification, and tries to preserve directional trends in the approximation line using a pre-defined distance threshold", which may be varied according to the amount of simplification required [40].

Detailed study of mathematical similarity and discrepancy by McMaster [43] ranks the DP algorithm as "mathematically superior". White [44] performed a study on simplification algorithms on critical points of curve similarity and showed that Douglas-Peucker method was best at choosing splitting points and called the obtained results as "overwhelming". Another study by McMaster [45] calls it as one of the most geometrically efficient algorithms in processing strings of x-y coordinate pairs.

However, [46] describes the method as highly time consuming, having a worst case complexity of $O(n^2)$. However, due to simplicity of the algorithm, different methods have been proposed to implement it. One of such proposals, which succeeded to reduce the complexity of the method from $O(n^2)$ to $(n\log_2 n)$ was Hershberger [47].

"Douglas-Peucker method starts by drawing a straight line from the first point to the last point and calculates the perpendicular distance for all the points between those points" [47]. Then it finds the point with the biggest perpendicular distance - if this distance is smaller than a certain threshold, then all the points between the initial two are discarded since they have a negligible impact on the overall shape. If the distance is bigger it splits the polyline into two halves and the process recursively starts over for both those sublines. In the end, all sublines are combined into one.

Another popular algorithm, called the **Visvalingam-Whyatt algorithm,** works from the inside-out. "It starts by computing the area of the triangle formed by each consecutive three points along the polyline. Then the midpoint of the triangle with the least area is thrown out since those three points are the closest to collinear and the area of triangles on either side are recomputed" [48]. The process continues until all remaining triangles are above a certain threshold. While dropping intermediate points if they fall within a tolerance like Douglas-Peucker algorithm does is reasonable, the "assumption that the furthest point from the anchor-floater line is a critical point is questionable" [48], says Visvalingam. Visvalingam-Whyatt algorithm uses elimination via an effective area, rather than selection.

Visvalingam and Whyatt argue, that contrary to popular claims, Douglas-Peucker algorithm ceases to be global and holistic after the selection of the first point. Their process requires a holistic view of the line whilst eliminating detail. In practice, the performance of Visvalingam-Whyatt and Douglas-Peucker is the same, as the worst-case for Douglas-Peucker requires specific conditions.

A study by New York Institute of Technology compared 6 different line compression algorithms. They used both algorithmic and manual visual grading for comparison. The results show that Douglas-Peucker algorithm was selected superior over the second

place by over 2 times, while Visvalingam-Whyatt was placed 3rd [49]. This mirrors the opinion in academic world: while Douglas-Peucker can be slow, it's quality is superior.

2.2.6.2 Evaluating compression

Un-compressed and compressed data sets differ in size, making the comparison between them more difficult. Some similarity algorithms could work, for example Hausdorff algorithm described in 2.2.5.4 does not require the datasets to be of equal size. One way to continue would be to find more similarity algorithms that do not require the size of datasets to be the same. However, there is a more intuitive and correct way.

Base line calculated for relative heights serves as a good reference point. It also marks the angle of the road. There are two areas, above and below the base line, which can be used to compare the data before and after compression, shown in Figure 11. In both datasets, areas above base line should be close in value, same applies to areas below baseline. As compression doesn't smooth the trend, we can be sure, that as long as the areas didn't change too much we didn't lose valuable information. We can then use a maximum allowed difference between those areas as our evaluation – meaning we will compress until it exceeds the given maximum allowed difference.

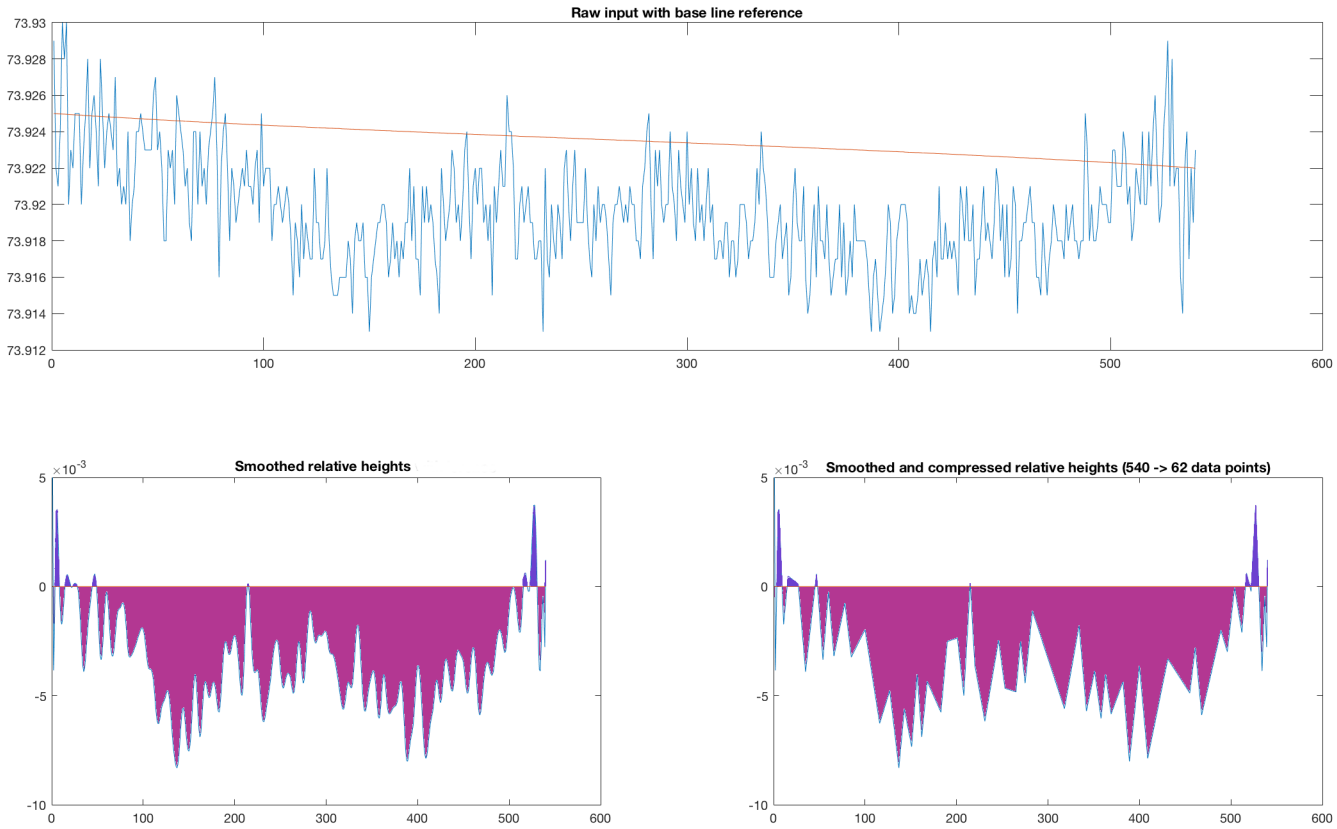

Figure 11 Top most figure shows a raw profile; bottom left shows relative heights smoothed by Savitzky-Golay filter using 11th degree polynomial and window size of 40; bottom right is the compressed version of smoothed profile.

2.2.7 Calculating quantities for milling and filling

Automatic calculation of milling (removing parts of road) and filling (adding a layer of new asphalt) is something that even Teede Tehnokeskus does not have a good idea how to do. The problem is the dependencies of various variables: angle of the road (current, historic, desired), quality of data (is their extreme noise at the edges of road data), conditions below road surface (will have an effect on the thickness of road) – which are data not known to our toolset. For the proof of concept however, we could presume that the angle of the road is ideal, meaning we don't have to correct the angle. This is a suitable suggestion for Teede Tehnokeskus. Having an ideal thickness of a new road as a user input (which could be determined by road draftsmen knowing background knowledge), shown in Figure 12, lets us minimise the filling quantity and approximate both milling and filling volume.

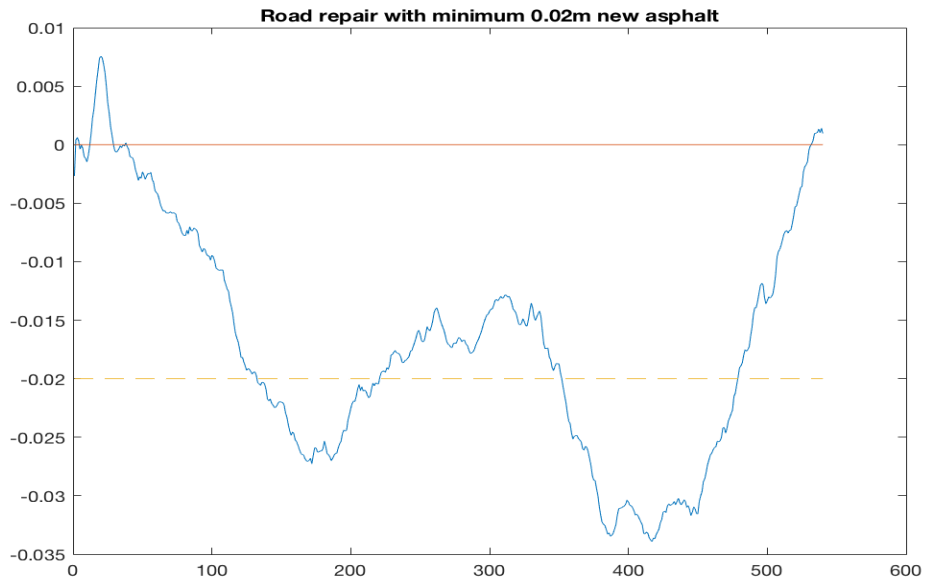

Figure 12 Road repair calculation

2.3 Evaluating theoretical analysis

What first seemed like separate problems to be solved grew more intertwined after further analysis. The initial intuition to first smooth the data before doing any other operations seems to have been wrong as necessary detail would be lost. The reason for smoothing itself moved from aiding visualisation to necessity for compression (section 2.2.5). In addition, a simple moving average filter which at first seemed reasonable might not be enough, at least not for the removal of outliers (2.2.5.1, 2.2.5.3).

This section describes what was learned from theoretical analysis in paragraph 2 and what should be the next steps going forward.

Workflow

The order of processing the data, shown in Figure 13 should be the following: extract and separate the initial data, reshape it to 2D form, calculate relative heights, remove outliers and smooth the data, compress the data. After calculating the relative heights it's also possible to find approximate filling and milling quantities, for this there is no need for smoothing nor compression.

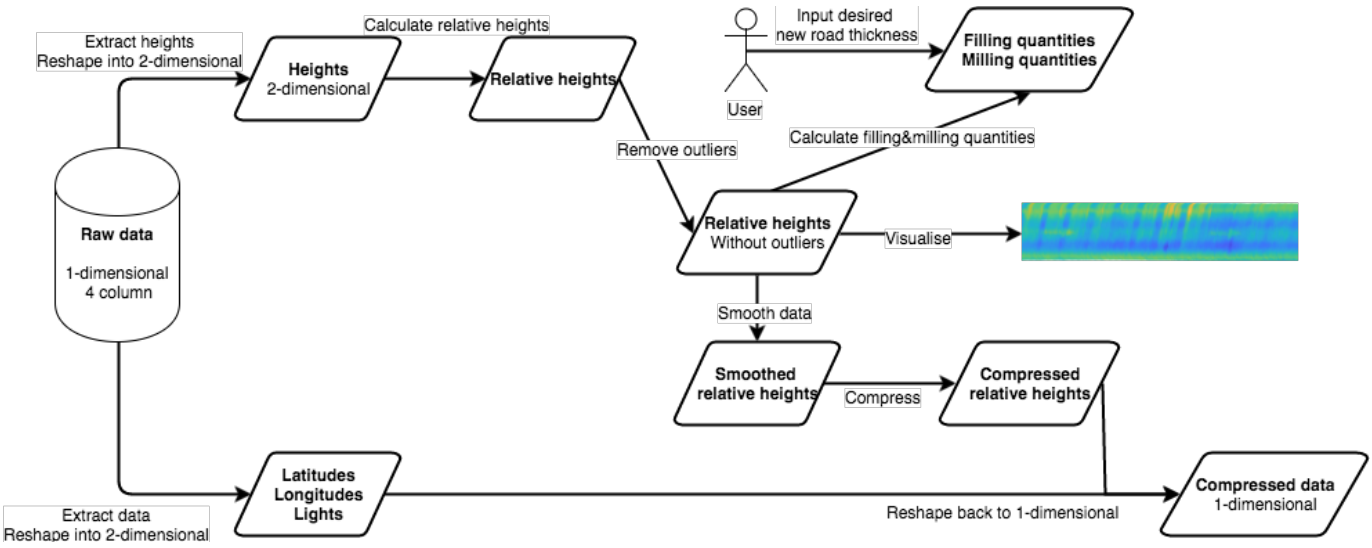

Figure 13 Expected workflow

Relative height calculation:

Road height changes often, which makes the absolute heights useless for further analysis. This results in a need to calculate relative heights, which are consistent enough to reveal the road characteristics. Both algorithmic and visual validation depend on those relative heights, making it clear that those heights should be calculated on raw data. Raw data comes initially in a one-dimensional form, holding 4 columns: latitude, longitude, heights, and lights. Before any processing, these values should be extracted and each of them reshaped to a two-dimensional matrix. Produced matrix of raw heights might contain outliers at the edges of the road, either in the form of rocks or other objects, so it is necessary for the relative height calculation to consider applying some counter measure for that without smoothing the data itself (section 2.2.4).

Calculating approximate filling quantities

Achieving an approximation of any quantities considering road conditions is difficult. The main theme seems to be that everything depends on something. This makes any automatic decision more difficult. In the realm of this scope, having a system that considers all different parameters would be too complex. However, communicating with Teede Tehnokeskus gave birth to a simple goal: having a function, that takes in a desired thickness of new asphalt, measure the needed volumes for milling and filling to achieve a smooth road (section 2.2.7). This solution would leave the current angle as the

ideal one. Additional improvement would be to add an optional angle parameter, which could be used as a new baseline.

Smoothing of the data

Some roads are more even than others, newer lasers and calibrators are better than old ones. However, for visualisation, the uneven-unsmoothed road is still the most honest way to go (Figure 7). Smoothing loses detail, regardless how it is achieved and in most cases, more detail is better. Nonetheless, there are at least two reasons why smoothing would have a positive effect:

- by smoothing the data, we reduce unique data points, thus we make rendering of the output easier for the processor. This really comes in play in 3D rendering, which isn't directly focused upon in this study. Be that as it may, compression itself is primarily meant for draftsmen who would probably like to visualise data in 3D
- compression works far better when the line is smoother, making it the primary reason to smooth our data (Figure 8)

Paragraph 2.2.5.1 described many line simplification filters to smooth one-dimensional data. Subsequent paragraph 2.2.5.2 discussed two-dimensional image smoothing algorithms, which proved not to serve our purpose well. While the problem of many line simplification algorithms is a good one to have, having options still requires a choice at the end. For this, there are two characteristics we can compare to evaluate and choose the best smoothing algorithm. Paragraph 2.2.5.4 described the smoothness factor and different ways to compare the similarity between curves. The most suitable way to find the similarity must be chosen through testing and logic. This should conclude with the most suitable smoothing algorithm.

Reducing data

Douglas-Peucker is one of the most used line simplification algorithms to date, used heavily in robotics to compress laser measurements (section 2.2.6.1), however there is concerning factor about the performance of it. It is often cited of having the worst-case complexity $O(n^2)$, but this statement hides some of detail. With n being the number of vertices and m being the vertices of the simplified line the complexity is O(n*m). In practice, this is triggered only for very zig-zaggy lines [50], which our smoothing algorithm would remove. The expected time is however O(nlogn). Additionally,

knowing that the complexity is output dependent is a valuable insight. When m is small, this algorithm will be very fast. Bigger output results in a slower run. This can be a problem - even for a 200m road, we have approximately 3600-4000 profiles, each profile we might need 2-5 iterations to achieve desired result. Although subsequent runs are much faster due to smaller output, the combined result will still be slow.

Solution to combat that is radial distance simplification, which is a linear way to compress data (2.2.6.1). It's not as sophisticated as Douglas-Peucker or any other algorithm, but extremely fast. Using alone, it would not serve our purpose well. But using it as a pre-compressor would reduce the output for Douglas-Peucker, making the combined result fast and high-quality.

To measure the quality of compression a different approach compared to smoothing algorithm should be taken (2.2.6.2). There are a couple of reasons to counter the initial intuition here. Firstly, fewer suitable algorithms are available for robustly comparing vectors of different length. Second and more important, it's difficult to establish a threshold of how similar the input and output have to be. Is it ok to have 90% similarity, or maybe we need 99%? This could be tested and some % can be derived, but a more sophisticated way is needed. Compression is solely meant for road draftsmen, in order to simplify 3D rendering and make additional calculations. For this to work, having the highest fine-grain detail is not necessarily the desired input. What's needed is to get the information of road characteristics, having the volume of bumps and rails intact. A better way for a threshold is to define an allowed percentage of volume change for some areas of the profile. Which areas exactly will be determined. Additionally, a good compression should ideally give a result that is visually indistinguishable from the initial input, meaning a visual test would serve a purpose here as well.

3 Building the toolset

The goal of this paragraph is to walk the reader through the decisions made based on the theoretical analyses done on section 2. Each chapter concentrates on implementing a different part of the toolset. Section 3.1 combines the knowledge from methods described in 2.2.4 and proposes a new robust method to calculate the relative heights. This method is then implemented and a code snippet is shown to aid the reader in understanding. This is followed by section 3.2, which implements the algorithm to calculate approximate quantities of filling, described in 2.2.7, to make the road even using minimal amount of fill. Section 3.3 achieves two goals: both a suitable smoothing algorithm and the means to compare different outcomes are chosen. Finally, section 3.4 uses the smoothed data and compresses it for further study. For this, both the Douglas-Peucker and the Radial distance line simplification algorithms, described in section 2.2.6.1, are used to achieve a compressed version. For a threshold to use, a custom way is proposed and implemented, which compares integrals of before and after of different areas (first mentioned and analysed in section 2.2.6.2).

3.1 Visualising and calculating relative heights

Before diving into the implementation of a relative height calculation, let's first go over the need to calculate them with an example. One of the first things to do, to make sense of any data, is to visualise it. Taking the raw heights as an example, I tried to visualise those. First, I reformat the code to a more convenient two-dimensional matrix form, using Code 9. Next, as the laser data is in 3D, it came naturally to try to visualise everything in 3D. This, as it turns out, is rather slow. By using Code 7 Plotting measurement as a 3D surface, I was able to produce a 3D mesh of sample 2*3.8 metre road surface, shown in Figure 14. Even this small sample size was rather slow and the same code failed for a more respectable size of 200m long road. It was clear at this point, that 3D visualisation is not the way to go forward, as we need to visualize un-compressed and un-smoothed data.

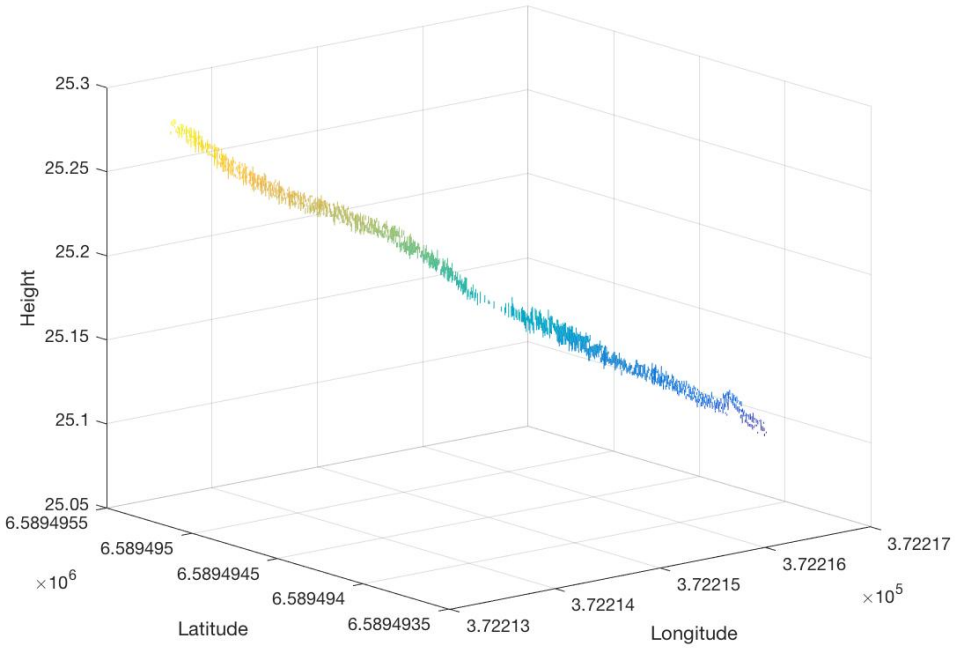

Figure 14 3D height mesh of a 2*4m road surface (10 000 data points)

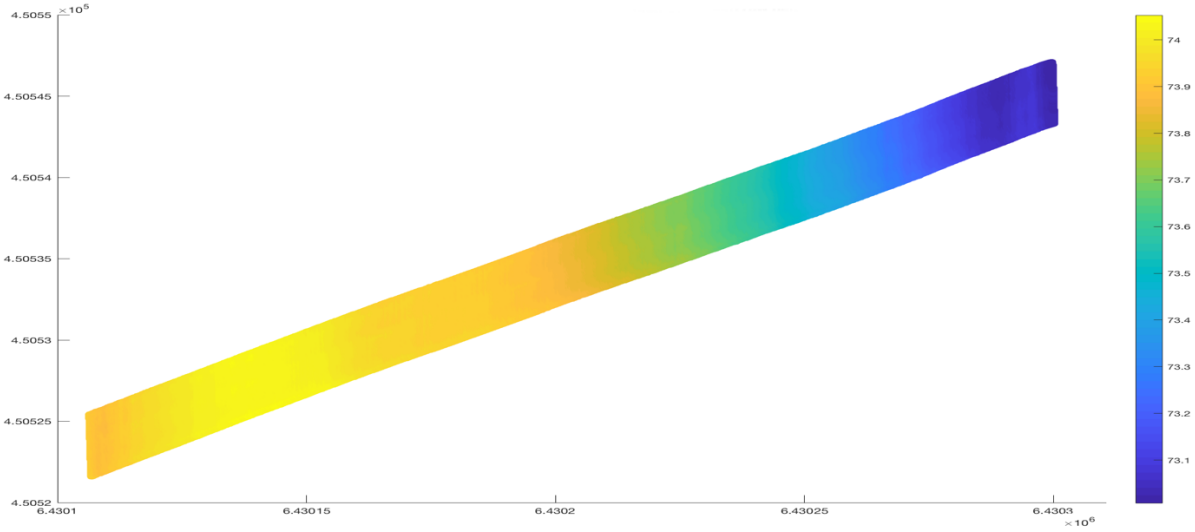

Figure 15 Visualizing the road using raw heights

Next was 2D visualisation, using Code 8 Plotting measurement as a 2D scatter plot. This is relatively free from performance implications, having no problem in drawing a scatter plot of 2 million data points. This is also much better to grasp for human eye, as the heights are pictured using colours. This is however, also the place where the need for relative heights became clear. As can be seen from Figure 15, absolute heights don't

give any insight into the characteristic of the road, as the height can be seen changing as much as 1 meter in the span of 200-metre long road. This fact produced the need to determine and calculate relative heights.

As the relative height calculation is done before other operations and the quality of other operations depend on the result of relative heights, then it is necessary that the determined base line is not affected by noise.

All the methods described in paragraph 2.2.4 suffer from some downside. Water method is difficult to implement and loses information on the peaks. Ruler and string methods are flexible but rely heavily on two peaking points on both side of the road which are used for the base line calculation. Angle method does not have those downsides, but uses a fixed angle to be followed, which is not to be trusted judging by sample data and consultancy from Teede Tehnokeskus. The angle of the road, while ideally being 2.5%, often changes depending on the road, increasing during corners and decreasing during bigger intersections. It should never be too small, otherwise rain wouldn't fall off, but the constant changing does affect the stability of the angle method.

Combining the information learned from theoretical analysis about different methods and the need to also avoid being hurt by noise in the data, there is a need to produce a new way of calculating relative data. It has to use a stable base line, without being affected by noise and it should not rely on key points of the data. Additionally, it would be necessary, that the angle of the road would also be cancelled out, otherwise changing of the angle would also change the relative heights. By using a median of the sides of the road and using the string method we get the wanted result.

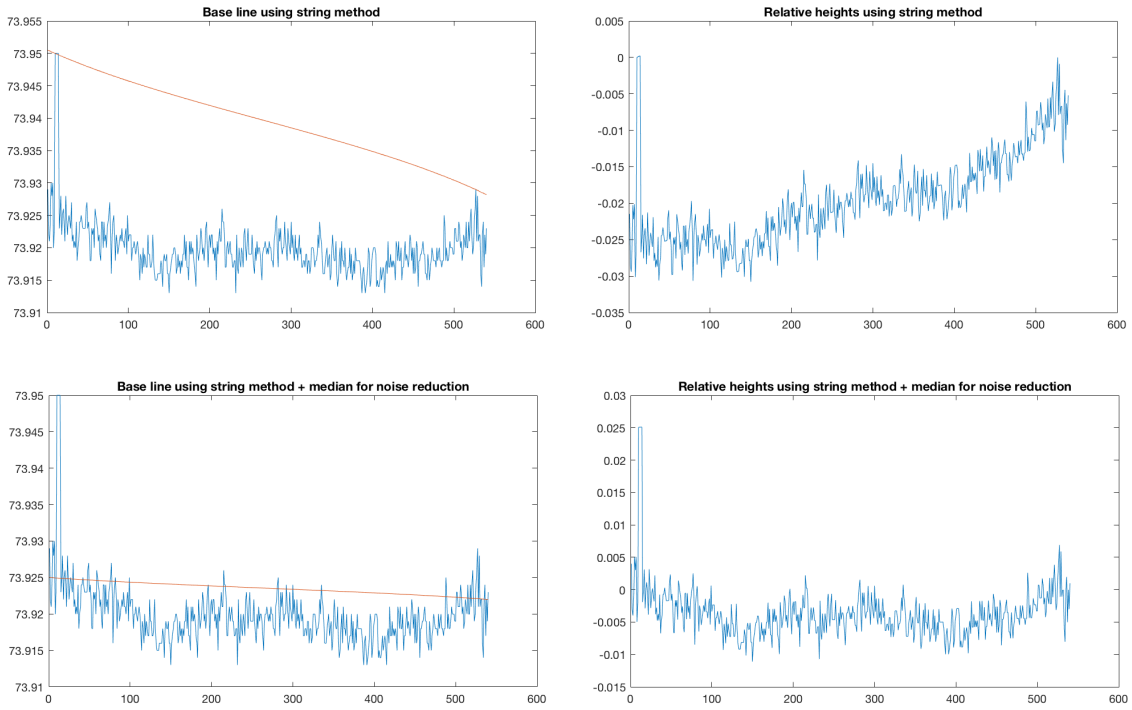

Figure 16 Noise safe, angle cancelling, peak and valley preserving relative height calculation method

To calculate the base line, I use median of the heights of 5% of the data on both sides of the road. This cancels any possible stones or other noise that might have a huge effect on the base line calculation. As can be seen from Figure 16, even a small noise in the data can alter the base line a lot, which in turn completely messes up the relative heights. By using medians, it is possible to greatly increase the reliability of the string method. To calculate the linear line, I used a simple two-variable linear equation:

$$y = a * x + b$$

Equation 3 Linear two-variable equation

By knowing two median points from the opposite sides of the profile, I can derive both a and b and calculate the baseline. For every data point, the difference between the point and the baseline is the new height, called relative height. Full code can be seen in paragraph Code 1 Calculating relative heights.

Visualizing the relative heights can be seen in Figure 21, having a much better detail about the road characteristics, showing the rails and even having the road markings visible as they are higher than the rest of the road. The range of the heights for this sample road segment can be seen to be about 4.5 cm.

3.2 Calculating approximate filling and milling quantities

To calculate the minimum filling volume in order to smooth the road, I used the method described in sections 2.2.7 and 2.3. For a proof of concept, it is acceptable for Teede Tehnokeskus to input the desired thickness of the new road. This means we can use the base line, calculated with the relative heights, as an angle to be followed in order to restore the road.

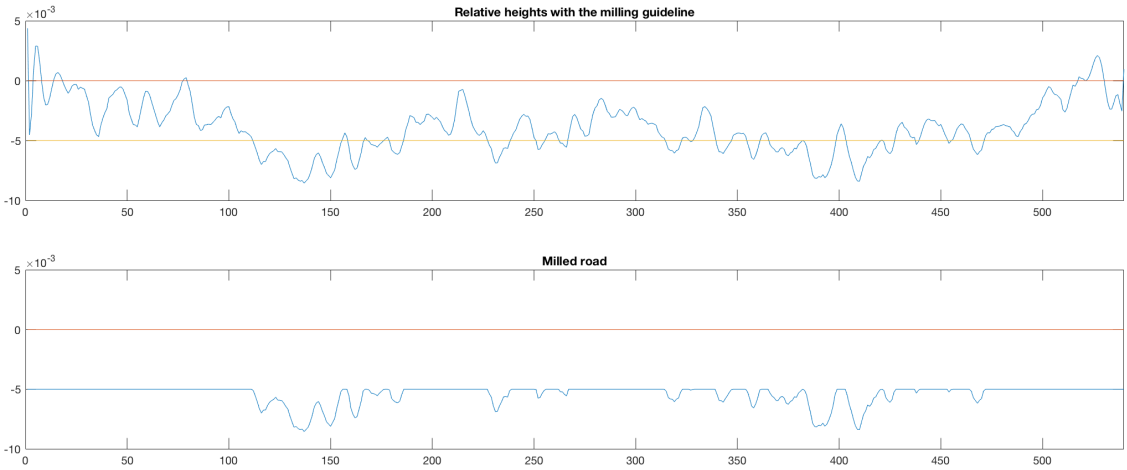

Figure 17 Milling of the road

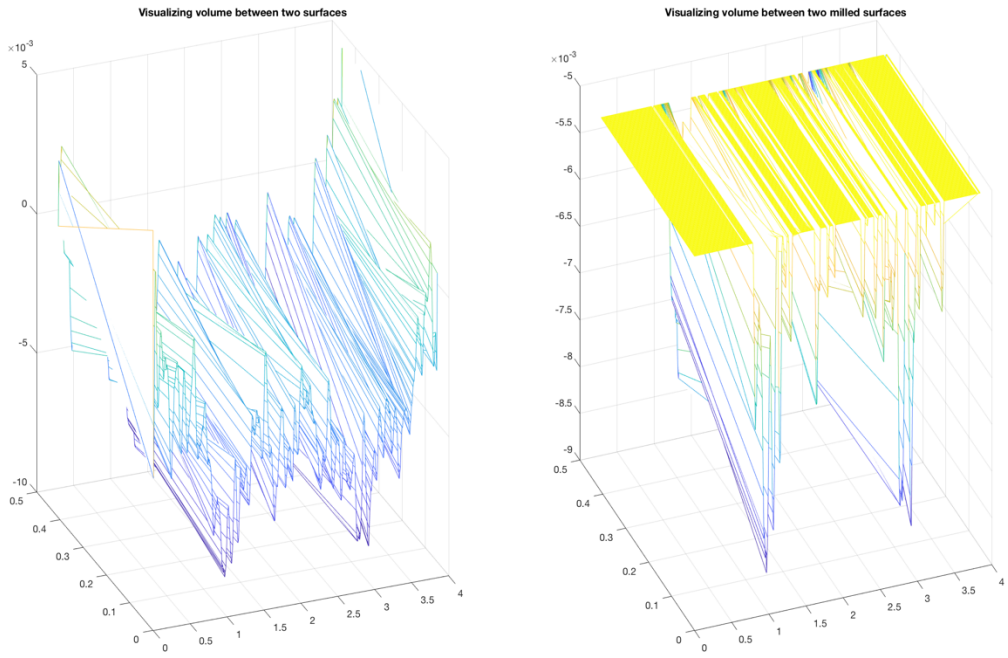

Figure 18 Volumes between two profile cuts before and after milling

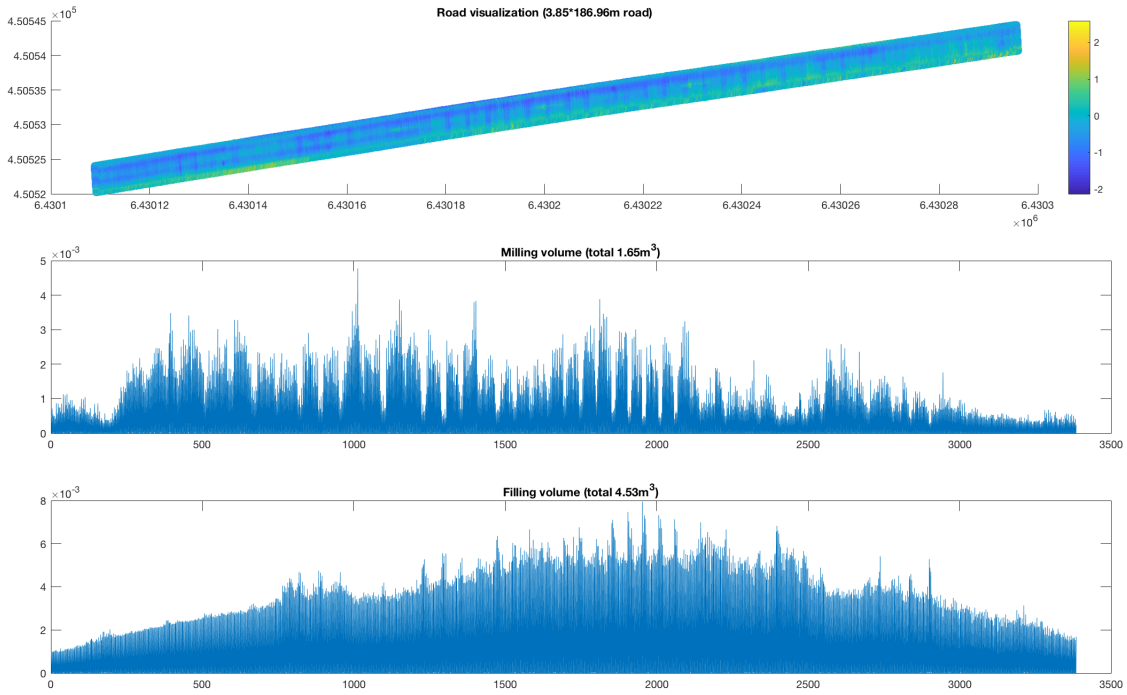

Figure 19 Milling and filling volume for a sample road

First we need to mill the road, which is pictured in Figure 17. For current sample, a user defined value for the new road layer was 0.005m. Usually, it would be higher, but the sample road is already relatively smooth, so no repairs should be needed. Nevertheless, as it still has visible rails, the sample could still be used for demoing purpose. Thickness of the new asphalt corresponds also to the minimum depth of milling. Adding any bumps over the baseline would yield the milling volume.

Full implementation can be seen in Code 2 Calculating filling and milling volume. This method does multiple things. Needed quantities are calculated per adjacent profile pairs. We'll calculate the Z-coordinates for both milling and filling surface, but first there is a need to calculate the width and the length of the road.

a) To calculate the length, we can use the difference of the latitudes of the profiles. For a 195m road, with 3600 line measurements, it is approximately 0.05m. For the best accuracy, this is calculated for every profile as it depends on the speed of the car that the laser is situated.

b) To calculate the width, there is a need to take into account the movement of the car. A simple Pythagoras theorem can help to calculate the width, which is the

square root of the sum of two components: the difference of longitudes (width of the road) squared and the previously calculated length squared.

After the width and length have been calculated, we can calculate both the milling and filling volumes.

c) We first mill the road, using the original baseline, which for relative heights is the straight line where y = 0. Everything above a new modified line, which is the desired thickness of the road below original baseline, will be milled.

d) To get the volumes, I use double integrals. More precisely, to get the best precision we could fit the data with high polynomial functions and calculate the actual integrals. But there really isn't any point of doing that as our data has a high number of data points. This means it is perfectly suitable to use trapezoidal rule [51], which is a technique for approximating definite integral.

Figure 18 shows the area between two adjacent profiles on the left side and on the right, the same profiles after the milling process. Figure 19 shows both the volume of milling and filling needed to smooth out a sample road: approximately 1.65m^3 of road should be milled and 4.53m^3 of fill is needed to achieve a minimum thickness of 0.005 new asphalt. Also, to verify visually using the same figure, it can be seen that the more rails a road has, the more fill is needed for that area. Milling is more related to bumps on the road and of course ultimately both of them are heavily dependent on the desired thickness of the new road.

3.3 Implementing and validating line smoothing algorithm

As our data might consist outliers which are subject to hurting the non-robust smoothing algorithms, it makes sense to first clear the data from outliers. Like described in 2.2.5.3, there is suitable intuitive algorithm for that, called Hampel identifier, which uses local window size and the distance between the median and a level some standard deviations away. Using the same sample pictured in Figure 10 and by using the Hampel identifier with the default parameters of 3 for window size and sigma of 3 standard deviations (Code 6 Remove outliers from input), results in clean data shown in Figure 20. This allows us to move on to choose the most suitable smoothing algorithm.

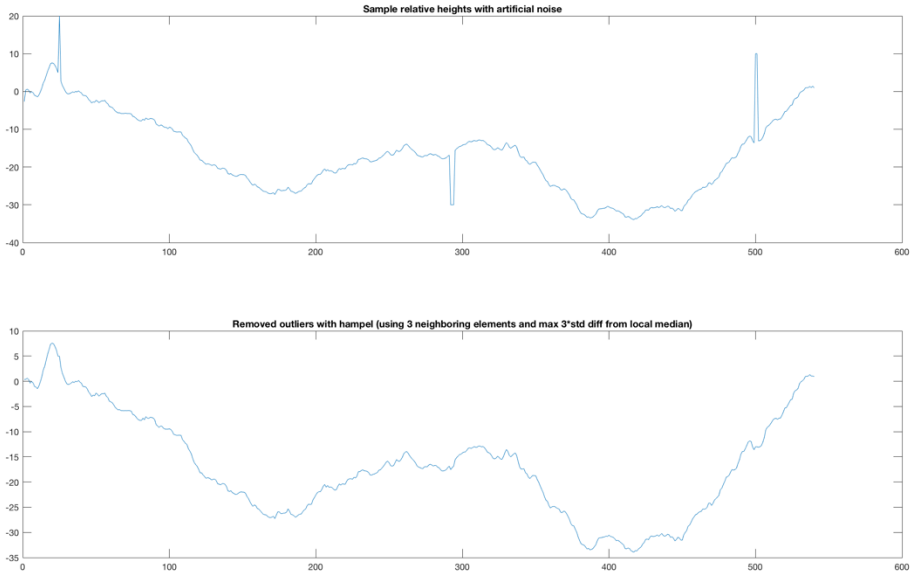

Figure 20 Removing outliers with Hampel identifier

Moving on with the actual smoothing process, section 2.2.5.4 describes 4 different ways of measuring the similarity or dissimilarity of two datasets. While the sum of absolute differences uses the absolute function, making it uncomfortable to use for analytical purposes, the other 3 are all suitable for comparing smoothing algorithms. Hausdorff distance measures the maximum shortest distance between the datasets, meaning it's a dissimilarity metric. Pearson distance is a measure of linear correlation between the datasets, but by taking the inverse of it, we again get a dissimilarity metric. Sum of squared errors is perhaps the easiest of those, but it is wildly used method of measuring the variation/error within a cluster. As they all work differently there isn't really a clear indication what would work best. Performance wise, Hausdorff distance is the heaviest, but the complexity is still linear O(n+m) [37]. For this reason, I have decided not to exclude any of them and compare all smoothing algorithms using all three of them to get the best of all worlds.

Table 3 shows the results in comparing the smoothing algorithms with Hausdorff, Pearson and SSD. Ultimately, a decision has to be made whether to value more the reduction of smoothing factor or the minimal difference between the initial and smoothed data. Out of curiosity, I also added a popular 2-dimensional smoothing filter, called Gaussian filter to the mix. As can be seen, the reasoning mentioned in 2.2.5.2

turned out to be justified, as the smoothed line is a lot more different compared to 1-dimensional smoothing filters.

There were 4 different smoothing algorithm settings that stood out as the best in smoothing while also not deferring from the original input too much. These 4 were the following:

- Lowess with 10 period window
- Savitzky-Golay 40 period, 11^{th} degree polynomial
- Savitzky-Golay 50 period, 11^{th} degree polynomial
- Savitzky-Golay 60 period, 11^{th} degree polynomial

Those 4 went on to the second round of comparison, where each of them were run on three different road samples and the results were averaged together (shown in Table 4). Again, the results were close, and mirrored the first test: better smoothness factor results in an output, that is further from the original. From the smoothness factor alone, Savitzky-Golay with setting 60/11 was the best one, followed by Lowess 10, Savitzky-Golay 50/11 and Savitzky-Golay 40/11. And although the difference between the result was not a big one, the golden route seems to be the middle man, Savitzky-Golay with a window size of 50 and an 11^{th} degree polynomial, which I have decided to pick as the best one to use. Result can be seen from Figure 21 and judging by the eye, it's actually clearer, for example the road markings can be identified more easily.

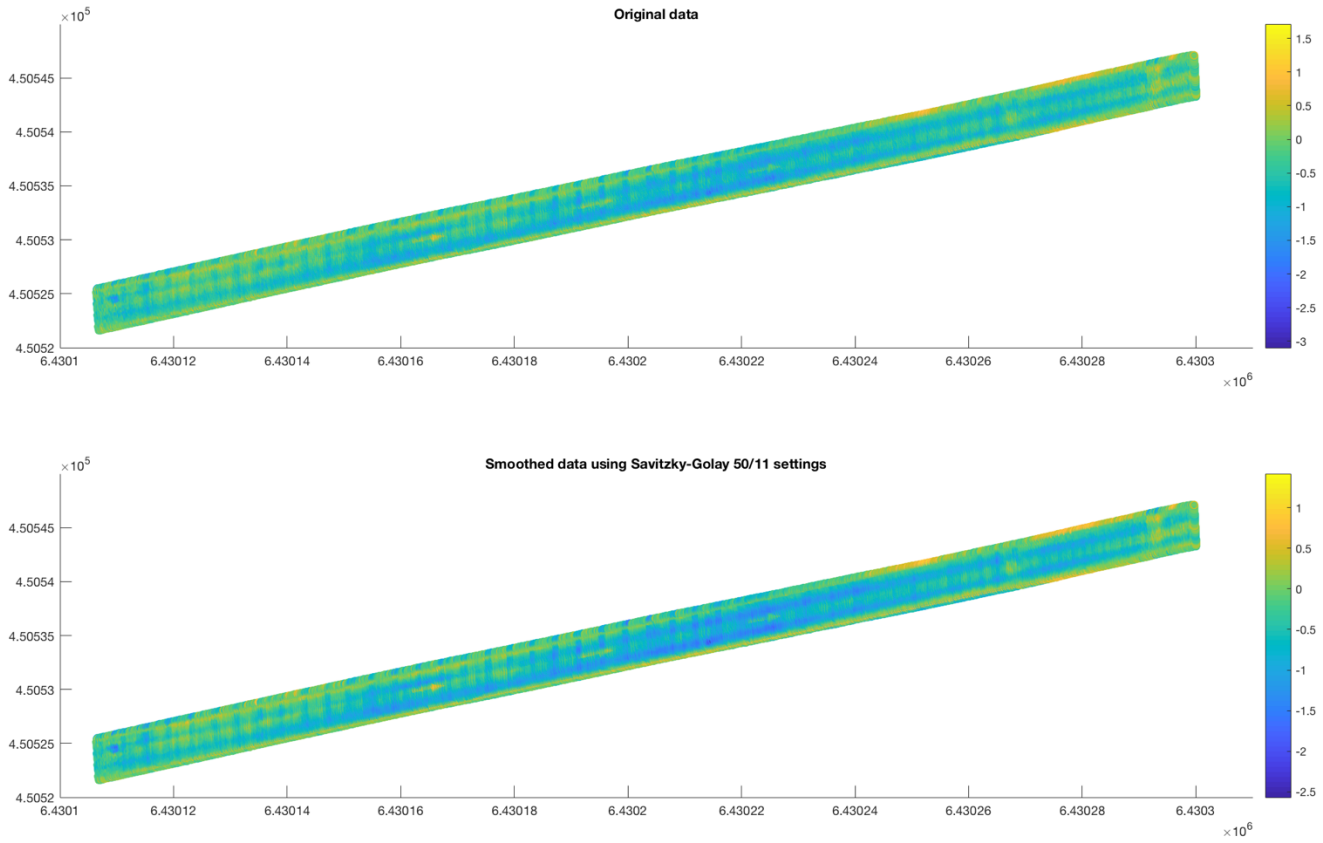

Figure 21 Raw vs smoothed road visualization

3.4 Reducing data and evaluating the result

Compression process that I have implemented consists of multiple parts: evaluating compression, pre-compression and compression. This paragraph describes all three and explains the reasoning behind the implementation.

3.4.1 Calculating surface areas for evaluation

Following the description in 2.2.6.2 and 2.3 about the evaluation, using pre-existing algorithms to calculate the similarity of two datasets is not optimal. A better way would be to measure the loss in the road characteristics that the draftsmen would later use – areas for bumps and rails. A base lane was calculated in section 3.1, that marks the angle of the road. Two areas are created, above and below the line. Due to the logistics of the base line drawn, the area below the lane is a lot bigger. To eliminate the possibility of a compressed rail, which uses different tolerance for calculation, having an unwanted effect on losing too much detail in the case of bumps, it makes sense to separate those areas and compress them differently. At the end, we would like to

preserve the characteristics of both bumps and rails, meaning we can compress them separately and then combine the results together.

The code to calculate the surface area is shown in Code 3 Calculating upper and lower surface areas. This method can be used to calculate surface areas for all of the data or it can also be used per surface, which is used in evaluating the compression later on. The method also uses trapz to estimate the integral. It does it twice, once for the upper and once for the lower area. In both cases, the opposite side is substituted with zeros.

3.4.2 Pre-compression

Following the evaluation of theoretical analysis in section 2.3, to reduce the size of data points Douglas-Peucker has to handle I decided to use Radial distance to first pre-compress the data. Code 4 Radial simplify algorithm shows the implementation, which compares all elements with their adjacent element and if the distance is under given tolerance will exclude the succeeding element from the result. It then continues until an element is found which falls out of the tolerance. That element is kept and replaced as the anchor which to compare against all the following elements.

This algorithm is called many times to achieve the best wanted result, or at least close to it, as some estimations has to be done to achieve good performance. Segment of the code can be seen in Code 5 Pre-compressing upper and lower area of data using radial simplify. As radial simplify is linear it can be used quite comfortably. The way I achieve the best result is to reduce the tolerance until the acceptable precision is reached. I also use a minimum difference between before and after areas, as sometimes the differences can be small in quantity, but percentage wise can still be bigger than say 1%, which also seems like a good acceptable error. By default, the minimum acceptable difference is 0.0001 m^2 (1 cm^2), which seems reasonably small error to accept the outcome.

Figure 22 shows both pre-compression and the actual one, side by side for upper and lower areas. For this surface, the achieved compression rate was 20x, all while keeping the precision, which are the differences between upper and lower area changes, minimal.

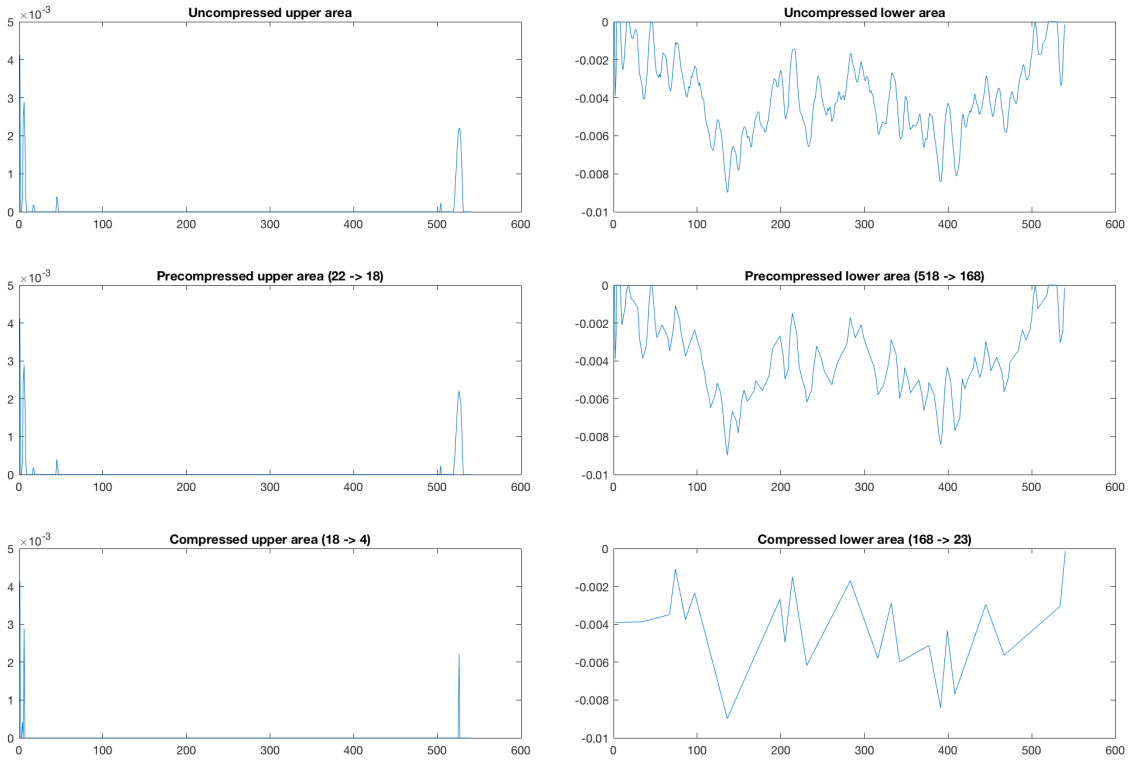

Figure 22 Compression of upper and lower areas

3.4.3 Compression

The idea behind implementing the compression follows the logic described in the previous paragraph. Both upper and lower areas are to be compressed separately and then the result is merged together. For the compression, the Douglas-Peucker algorithm is used, implemented in Code 10 Douglas-Peucker algorithm to simplify a line.

To achieve a good result for a given surface profile, a somewhat similar approach is used compared to pre-compression. The difference is that Douglas-Peucker isn't a linear algorithm, so it's best to both minimize the amount of times the algorithm is called and also minimize the input it is called with. To achieve that, we increase the tolerance until the result is no longer acceptable and then take the last acceptable result. This gives an ability to use the result of one Douglas-Peucker call as an input for the other call, reducing the size of the input of any following method call. Not only does the pre-compression reduce the input, every Douglas-Peucker compression reduces the input for the next call. Code segments of how this is achieved can be seen from Code 11 Compressing a line using Douglas-Peucker until acceptable precision is achieved.

3.4.4 Merging the result

When both pre-compression with radial distance and regular compression using Douglas-Peucker is done, we need to merge the results together. The root method for compression can be seen from the appendix, Code 12 Pre-compress (a), compress upper, lower parts (b), merge compressions (c). Besides calling the compression algorithms for each profile, this function also calculates the total size of the resulting data points and reshapes those indexes in one-dimensional form, for later use.

A sample visualization of a road before and after compression can be seen from the following drawing, Figure 23 and an another in appendix, Figure 28

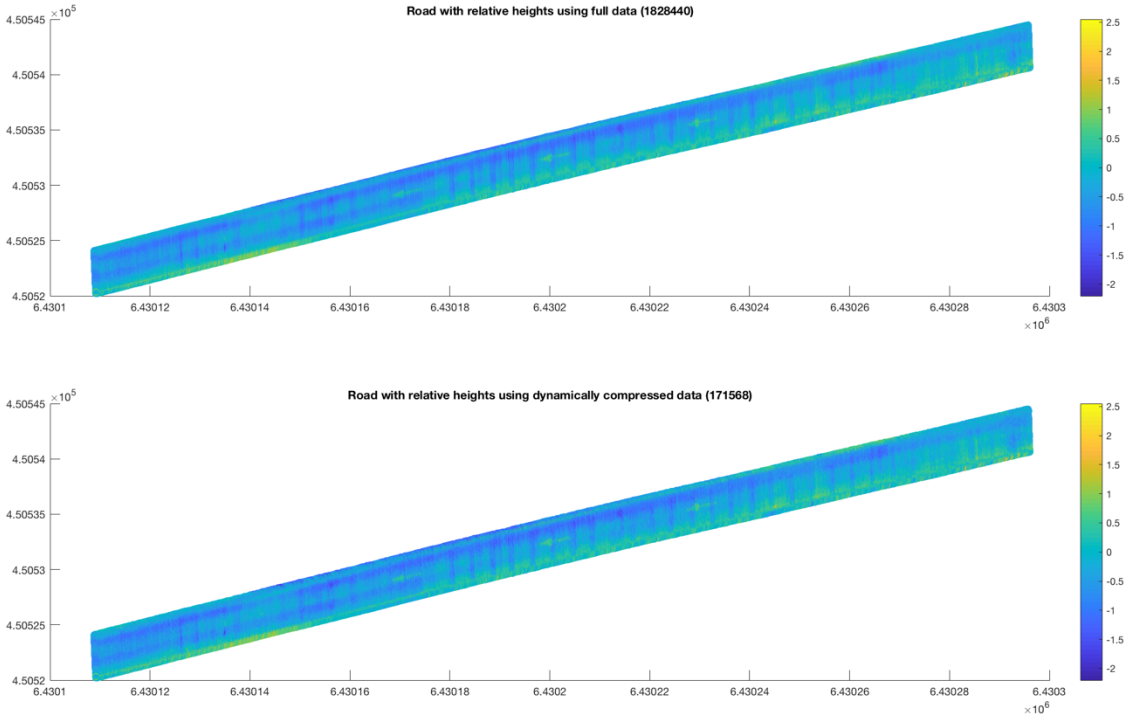

Figure 23 Sample road before and after compression (with a rate of 10.6x)

4 Examining and validating results

This paragraph combines together the theoretical research done in section 2 and implementation done in section 3 and analyses the achieved result both from the performance and the quality aspect. Future work is also laid out for Teede Tehnokeskus to actually start using the laser outputted data in production to achieve better result going forward.

4.1 Quality of results

Overall the author of this study is happy with the results. Comparing the visualization with the official software ViaPPS Desktop shows that no compromises were done. I would argue that both relative height calculation and smoothing was executed in a better way and there was no compression functionality present in the given software. Additionally, road segments up to 1000m can be visualized without any performance problem, instead of say 50m 3-dimensional segments, which are not really useful.

Relative height calculation implemented in 3.1 is a robust way to introduce a new base line. The more "honest" way would be to use an ideal angle of the road as a baseline. However, the ideal angle is mostly unknown (formally 2.5%, but it's constantly changing depending on the road) and would have to be a user input or somehow calculated from external data. Ideally, this is something that might be investigated in the future.

A lot of time and effort was put into smoothing functionality, which was a rather wide topic. What made it difficult was the vague understanding of the reason why smoothing was necessary at the first place. After realizing that the compression is the biggest reason why we need smoothing I went through both 1- and 2-dimensional algorithms, eventually staying with 1-dimensional realm, as it offers more flexibility and better accuracy. Through testing with various different ways of comparing before and after datasets I finally arrived what seemed like a golden route – Savitzky-Golay smoothing filter with a window size of 50 and 11^{th} degree polynomials. Visualizing the before and after data, pictured in Figure 21, gave an indication that the characteristics of the road remained the same regardless of smoothing. Due to some of the noise being removed in the process, the output seems clearer judging by the eye.

Achieving a good compression rate required to think about multiple aspects. First, how to compare a given compression algorithm and how far should the algorithm compress. For this I devised my own method of comparing the surface area changes above and below the base line calculated in relative height section. This makes intuitive sense, as the base line indicates the angle of the road we think is ideal. So, repairing the road would follow the baseline, making it logical to keep the areas relatively unchanged during compressing. Going forward, a decision for compression algorithm had to be done. For this I used a very popular Douglas-Peucker algorithm together with radial distance compression to achieve better performance. Coupled that with parallel computing resulted in a fast and effective compression algorithm, reaching over 10x compression rate for three different datasets (shown in Table 2) and showing minimal data loss (less than 1%, pictured in Figure 23)

4.2 Performance

Section 2.2.2 compared real-time vs batch processing, which resulted in the decision that for pre-processing the data, batch processing can be used. This reduces the restrictions from the performance aspect and lets us concentrate more on the quality, rather than being fast. That being said, a lot can be done with matrix calculations, which has many optimized libraries in most of the popular languages. Everything except compression, in our toolset uses either matrix calculations or linear complexity algorithms, making them fast to use.

For compression, there is a need to work on individual profiles and opportunities for matrix operations are limited. However, to increase the performance, parallel computing can be used in places where matrix calculation cannot be utilized. The major part that compresses a line can be processed separately by a single thread. Every language has some support for parallel computing, Matlab has parpool, “which creates a special job on a pool of workers, and connecting the MATLAB client to the parallel pool” [52]. It can also be used in cloud, sending assignments to workers in the cloud, maximizing the performance.

Table 2 compares 4 different ways of compressing data: Douglas-Peucker (DP), parallel DP, radial+DP and parallel radial+DP. As can be seen, the performance increases both with parallel computing and using radial distance for pre-compression. Every worker

added to the parallel pool decreases the duration. Using either more common programming languages where thread pool could be easily configured or MatLab itself, means performance of the compression can be scaled by adding more threads. Additionally, radial distance pre-compression helps with the performance a lot, like initially proposed in section 2.2.6.1.

		Douglas-Peucker	Parallel DP	Radial + DP	Parallel DP+Radial
Data 1	Time (seconds)	163.270	98.714	70.994	45.350
	Compression rate	10.211		10.580	
	Avg precision loss	0.972		0.976	
Data 2	Time (seconds)	161.539	106.699	59.008	38.715
	Compression rate	11.714		12.580	
	Avg precision loss	0.954		0.963	
Data 3	Time (seconds)	248.381	126.248	62.688	39.475
	Compression rate	14.880		15.340	
	Avg precision loss	0.607		0.620	

Table 2 Compression performance comparison

Going forward, performance does not seem to be a problem for multiple reasons. Firstly, the concept of batch processing allows for more time for chosen algorithms to run. Secondly, smoothing algorithms together with most other calculations can be implemented using optimized matrix calculations, making them very fast within similar quantities of data. Compression is really the only part of the toolset that suffers from slower algorithms that can't be resolved with vector mathematics. But using pre-compression and parallel computing achieves a reasonable result that can be used with production data as well.

4.3 Future work

This section lays out some of the problems and opportunities for the future work. Full code with the images and underlying thesis can be retrieved from git: https://bitbucket.org/eerikpotter/masterthesis. To get access to the repo, contact eerik@potter.ee

Having a digital model of the roads in Estonia offers huge opportunities for sophisticated analysis to better fix and build new roads. However, there are some hurdles still on the way:

a) One of the most difficult aspects about the current study was its exploratory nature where neither Teede Tehnokeskus nor author had precise requirements on the output. Because of this there was a lack of information about the tools and tasks of road draftsmen. A big part of the benefit of having the digital model is the fact that ideally there is no need to even go on the field for manual measurements. This requires clear input from draftsmen, who can tell exactly what is needed and why. Ideally, some of their tasks can also be automated, but even more, with a good enough model, this would really eliminate the mundane task of going to the field to make less detailed manual measurements. Having clear requirements about the eventual analysis would have the biggest priority going forward, in my opinion. This work is a step towards mutual understanding of the requirements and forming of specific use cases.

b) Copying 300m road segments manually to clipboard from ViaPPS Desktop is not the best solution. Ideally, this would be done with a separate software, either 3rd party or self-made. For a proof of concept, it's acceptable, but for longer roads it's not really optimal.

c) Without 3rd party tool to process the input, there are no geo-coordinates on the file. These would need to be added, as without the coordinates, some of the other tasks that could be automatized cannot be resolved.

d) Improvement on the visualization could be achieved by rendering the calculated relative heights to a map. This would require recalculating the width and length of the road, but seems doable. If this is achieved it would give a really good visualization of the whole road, however, calculations are still done separately by draftsmen and the need for this kind of sophisticated visualization is not clear at the moment.

e) Knowing more information about the decision road draftsmen do to determine the thickness of the road would help improve the automatic calculation of milling/filling quantities. As of now, there are lot of variables that the decision

depends on. Having a proof of concept for the filling calculation allows now to build on top of it. Reducing the variables might result in a solution that the algorithm can itself use some external data or user data to calculate even more accurate filling volume. This should be possible and would really be a game changer, but it requires the input from draftsmen and knowing the whole procedure how some of the decisions are made.

In my opinion, the most important aspect going forward in order to be maximally efficient and really revolutionize the way road projects are done, would be to map out the goals of the laser mapping system in a clearer manner. More technical analysis is required to determine what is done manually in each step. Pre-processing blindly limits the potential this system has. Knowing more information would give valuable insight into implementing some, if not all of it automatically. This would increase the quality, reduce the cost and make the whole procedure much faster. Although the technology itself is new, this work describes the technical scope and base line required for future work. I feel optimistic that the laser acquired by Teede Tehnokeskus will be put to good use.

Summary

Technology companies not realizing the full potential of intelligent data usage are a rarity these days. In a field of road infrastructure, Teede Tehnokeskus is a company which provides engineering consultation services. During the year of 2016, they acquired a mobile laser system called ViaPPS, enabling to scan Estonian roads digitally for further processing. This decision of mapping the roads digitally comes from the need of eliminating manual measurements done on the field, requiring expensive and inefficient expertise. The solution of using laser mapping for measurements produces a new problem – how to handle the enormous amounts of data produced.

This study concentrated on analysing the possibilities of extracting valuable information from laser outputted data. The highest value for Teede Tehnokeskus from gathered data is the height measurement for georeferenced data points on the road. Out of this grew the goals for the toolset this study aimed to create: visualize the road, calculate filling and milling volumes to smooth out the road and compress the data as much as possible for later processing.

It turned out quickly, that initial idea of 3D visualization to get a sense of the road characteristics is both unnecessary and slow to render. A better way is to use a 2D scatter plot, where the 3rd dimension is shown using colours. Still, a way to calculate relative heights was needed to counter the absolute initial heights. For this an algorithm using median of both edges of a surface profile was implemented. This forms a base line, which indicates the angle of the road and is used for all further processing.

Calculating the filling and milling volumes relies on the aforementioned base line. Unfortunately, not everything is fully documented on how the decisions of calculating those volumes are done manually. A recurring theme seems to be that there are many dependencies in making any decisions. Therefore, this was a rather difficult task to resolve. Consulting with Teede Tehnokeskus provided a clearer goal of using the base line as an ideal angle of the road and having the user input a desired thickness of new

road. The resulting implementation therefore uses provided coordinates and surface profiles and calculates both milling and filling volumes to achieve a smooth road.

It seemed initially rather intuitive, that there is no need to smooth the data, as it undesirably loses information. However, testing with compression performance indicated that noisier data results in impractical compression rates. Additionally, for uneven roads, the smoothing factor actually made the visualization better, judging by the eye. Multiple problems had to be solved when picking a smoothing algorithm. Through theoretical analysis it was derived that using 1D processing over 2D gives more control and a better result, leaving still a decision of choosing between many different algorithms. Using a combination of Hausdorff distance, Pearson correlation coefficient and the sum of squared distance to limit the error produced by smoothing, what seemed like a golden route was chosen: Savitzky-Golay smoothing filter with a window size of 50 and 11^{th} degree polynomial. Tests show that this achieved the best smoothing factor, while maintaining as much of the original road characteristics as possible.

Compression was done next, where a popular Douglas-Peucker compression algorithm was used. To counter the possible performance problems [worst case $O(n^2)$] a linear Radial distance pre-compression was applied. Additionally, as vector calculations were limited for compression, to further increase the performance, parallel computing was used. Tests showed that performance with those improvements increased 3-6 times, while achieving compression rates of around 10-15x. A lot of thought was put into measuring the quality of compression. None of the available algorithms out there seemed particularly useful for allowing to specify a suitable precision threshold. Again, the baseline calculated earlier offered a solution. By comparing the differences of both the upper and lower areas in respect to the base line, a logical way of evaluating the compression was achieved. This seemed to be successful, resulting 10x compression rates with absolutely no loss judging by eye.

Overall, the author of this thesis is satisfied with the results. Implemented algorithms were justified through theoretical research and best of them tested against each other until the most suitable was chosen. Looking to the future of this system offers multiple areas of potential success. Be it an automatic add-on for Google maps, using external

APIs for automatic input into toolset or switching to a general programming language for a more flexible development flow.

The underlying thesis serves multiple purposes going forward. Firstly, the initial vagueness of detail and the lack of overall structure was improved throughout writing and implementing this work. Some questions about the structure and handling of data were answered, but also new ones were created bringing added functionality. The goal was never to write a fixed solution in stone, rather it was to understand and explore new possibilities of the laser outputted road data, visualizing the results and pre-processing the data for later manual work. From here on, technical analysis of both manual work of draftsmen and use cases has to be conducted before continuing with implementation of more complex solutions.

References

[1] J. Bossler, C. Goad, P. Johnson, and K. Novak, “GPS and GIS map the nation’s highways,” *Geo Info Syst.*, vol. 1, no. 3, pp. 27–37, 1991.

[2] D. A. Grejner-Brzezinska, “Mobile mapping technology: Ten years later (part one),” *Surv. Land Inf. Syst.*, vol. 61, no. 2, pp. 75–92, 2001.

[3] D. A. Grejner-Brzezinska, “Peer-reviewed Papers-Mobile Mapping Technology: Ten Years Later (Part Two),” *Surv. Land Inf. Syst.*, vol. 61, no. 3, pp. 137–152, 2001.

[4] C. V. Tao and J. Li, *Advances in mobile mapping technology*, vol. 4. CRC Press, 2007.

[5] N. El-Sheimy, “Mobile Multi-Sensor Systems: Final Report (1995-1999),” *Int. Assoc. Geod. IAG Spec. Comm.*, vol. 4, 1999.

[6] C. Ellum and N. El-Sheimy, “Land-based mobile mapping systems,” *Photogramm. Eng. Remote Sens.*, vol. 68, no. 1, pp. 13–17, 2002.

[7] H. Kaartinen *et al.*, “Mobile mapping–Road environment mapping using mobile laser scanning,” *EuroSDR Off. Publ.*, vol. 62, pp. 49–95, 2013.

[8] A. Jaakkola, J. Hyyppä, H. Hyyppä, and A. Kukko, “Retrieval algorithms for road surface modelling using laser-based mobile mapping,” *Sensors*, vol. 8, no. 9, pp. 5238–5249, 2008.

[9] “Annual Laser Market Review & Forecast: Can laser markets trump a global slowdown?” [Online]. Available: http://www.laserfocusworld.com/articles/print/volume-52/issue-01/features/annual-laser-market-review-forecast-can-laser-markets-trump-a-global-slowdown.html. [Accessed: 14-Apr-2017].

[10] “ViaPPS - ViaTech AS.” [Online]. Available: http://www.viatech.no/products.aspx?lang=en&id=6. [Accessed: 19-Mar-2017].

[11] “Clipboard (Windows).” [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ms648709(v=vs.85).aspx. [Accessed: 14-Apr-2017].

[12] U. Franke, “Real time 3D-road modeling for autonomous vehicle guidance,” in *Selected papers of the 7th Scandinavian conference on image analysis*, 1992, pp. 277–284.

[13] K. Fatahalian, J. Sugerman, and P. Hanrahan, “Understanding the efficiency of GPU algorithms for matrix-matrix multiplication,” in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, 2004, pp. 133–137.

[14] D. Weiskopf, *GPU-Based Interactive Visualization Techniques*. .

[15] Nvidia, “Introduction to Image Processing on the GPU,” 16-May-2005. .

[16] T. O’Haver, *A Pragmatic introduction to signal processing*. 1997.

[17] A. Verma and A. Mishra, “Image Compression using Gaussian Smoothing Filter and Median Filter,” *Int. J. Recent Innov. Trends Comput. Commun.*, vol. 3, no. 11, Feb. 2015.

[18] L. N. Trefethen, *Approximation theory and approximation practice*. Siam, 2013.

[19] J. S. Simonoff, *Smoothing methods in statistics*. Springer Science & Business Media, 2012.

[20] S. W. Smith and others, “The scientist and engineer’s guide to digital signal processing,” 1997.

[21] R. J. Hyndman, “Moving averages,” in *International Encyclopedia of Statistical Science*, Springer, 2011, pp. 866–869.
[22] W. S. Cleveland and S. J. Devlin, “Locally weighted regression: an approach to regression analysis by local fitting,” *J. Am. Stat. Assoc.*, vol. 83, no. 403, pp. 596–610, 1988.
[23] R. Isnanto, “Comparation on several smoothing methods in nonparametric regression,” *J. Sist. Komput.*, vol. 1, no. 1, pp. 41–48, 2011.
[24] R. W. Schafer, “What is a Savitzky-Golay filter?[lecture notes],” *IEEE Signal Process. Mag.*, vol. 28, no. 4, pp. 111–117, 2011.
[25] C. K. Larive and J. V. Sweedler, *Celebrating the 75th Anniversary of the ACS Division of Analytical Chemistry: A Special Collection of the Most Highly Cited Analytical Chemistry Papers Published between 1938 and 2012*. ACS Publications, 2013.
[26] W. S. Cleveland and C. Loader, “Smoothing by local regression: Principles and methods,” in *Statistical theory and computational aspects of smoothing*, Springer, 1996, pp. 10–49.
[27] “Low-Pass Filtering (Blurring).” [Online]. Available: https://diffractionlimited.com/help/maximdl/Low-Pass_Filtering.htm. [Accessed: 31-Mar-2017].
[28] “soundsilence/ImageSmoothing,” *GitHub*. [Online]. Available: https://github.com/soundsilence/ImageSmoothing. [Accessed: 14-Apr-2017].
[29] “Image Filtering & Neighborhood Processing—Wolfram Language Documentation.” [Online]. Available: http://reference.wolfram.com/language/guide/ImageFilteringAndNeighborhoodProcessing.html. [Accessed: 14-Apr-2017].
[30] S. I. Olsen, “Estimation of noise in images: An evaluation,” *CVGIP Graph. Models Image Process.*, vol. 55, no. 4, pp. 319–323, 1993.
[31] V. Barnett and T. Lewis, *Outliers in statistical data*. Wiley, 1974.
[32] “Outlier removal using Hampel identifier - MATLAB hampel.” [Online]. Available: https://www.mathworks.com/help/signal/ref/hampel.html. [Accessed: 01-Apr-2017].
[33] I. Ben-Gal, “Outlier detection,” *Data Min. Knowl. Discov. Handb.*, pp. 131–146, 2005.
[34] “Distance Metrics.” [Online]. Available: https://numerics.mathdotnet.com/distance.html. [Accessed: 09-Apr-2017].
[35] P. Sedgwick and others, “Pearson’s correlation coefficient,” *Bmj*, vol. 345, no. 7, 2012.
[36] National Council on Measurement in Education, “Glossary of Important Assessment and Measurement Terms.” [Online]. Available: http://www.ncme.org/ncme/NCME/Resource_Center/Glossary/NCME/Resource_Center/Glossary1.aspx?hkey=4bb87415-44dc-4088-9ed9-e8515326a061#anchorC. [Accessed: 09-Apr-2017].
[37] “Hausdorff distance,” *Wikipedia*. 28-Jul-2016.
[38] “What is data reduction? - Definition from WhatIs.com,” *SearchDataBackup*. [Online]. Available: http://searchdatabackup.techtarget.com/definition/data-reduction. [Accessed: 02-Apr-2017].
[39] J. Muckell, J.-H. Hwang, C. T. Lawson, and S. Ravi, “Algorithms for compressing GPS trajectory data: an empirical evaluation,” in *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2010, pp. 402–405.

[40] N. Meratnia and R. De By, “A new perspective on trajectory compression techniques,” in *Proc. ISPRS Commission II and IV, WG II/5, II/6, IV/1 and IV/2 Joint Workshop Spatial, Temporal and Multi-Dimensional Data Modelling and Analysis*, 2003.
[41] “psimpl - radial distance simplification.” [Online]. Available: http://psimpl.sourceforge.net/radial-distance.html. [Accessed: 09-Apr-2017].
[42] V. Nguyen, S. Gächter, A. Martinelli, N. Tomatis, and R. Siegwart, “A comparison of line extraction algorithms using 2D range data for indoor mobile robotics,” *Auton. Robots*, vol. 23, no. 2, pp. 97–112, 2007.
[43] R. B. McMaster, “A statistical analysis of mathematical measures for linear simplification,” *Am. Cartogr.*, vol. 13, no. 2, pp. 103–116, 1986.
[44] E. R. White, “Assessment of line-generalization algorithms using characteristic points,” *Am. Cartogr.*, vol. 12, no. 1, pp. 17–28, 1985.
[45] R. B. McMaster, “Automated line generalization,” *Cartogr. Int. J. Geogr. Inf. Geovisualization*, vol. 24, no. 2, pp. 74–111, 1987.
[46] Z. Li, “An algorithm for compressing digital contour data,” *Cartogr. J.*, vol. 25, no. 2, pp. 143–146, 1988.
[47] J. E. Hershberger and J. Snoeyink, *Speeding up the Douglas-Peucker line-simplification algorithm*. University of British Columbia, Department of Computer Science, 1992.
[48] M. Visvalingam and J. Whyatt, “Line generalisation by repeated elimination of the smallest area,” *Cartogr. J.*, vol. 30, no. 1, pp. 46–51, 1992.
[49] “Home - Curve Simplification.” [Online]. Available: http://web.cs.sunyit.edu/~poissad/projects/Curve/index.php. [Accessed: 09-Apr-2017].
[50] “recursion - Line that triggers Worst-Case for Douglas-Peucker Algorithm? - Stack Overflow.” [Online]. Available: http://stackoverflow.com/questions/31516058/line-that-triggers-worst-case-for-douglas-peucker-algorithm/31566048. [Accessed: 14-Apr-2017].
[51] M. Sugihara, “A class of functions for which the trapezoidal rule gives the exact value of integral over the infinite interval,” *J. Comput. Appl. Math.*, vol. 20, pp. 387–392, Nov. 1987.
[52] “Create parallel pool on cluster - MATLAB parpool.” [Online]. Available: https://www.mathworks.com/help/distcomp/parpool.html. [Accessed: 22-Apr-2017].

Appendix 1 - code examples

```
function [data] = calculateRelativeHeights(data, edgeRangeInPercent)
    ......
    startingGround = median(data.heights(1:int64(data.dataPointsInLine * edgeRange), :));
    endingGround = median(data.heights(int64(data.dataPointsInLine * (1 -
edgeRange)):data.dataPointsInLine, :));

    for n = 1 : data.totalDataLines
        xn = data.longitudes(:, n);
        an = (startingGround(n) - endingGround(n)) / (xn(1) - xn(data.dataPointsInLine));
        bn = startingGround(n) - an * xn(1);
        data.correctionLine(:, n) = an*xn + bn;
        data.relHeights(:, n) = data.heights(:, n) - data.correctionLine(:, n);
    end
end
```

Code 1 Calculating relative heights

```
function [volumes] = calculateMillingAndFilling(data, roadThickness)
    ......
    for i = 2:data.totalDataLines
        distY = abs(data.latitudes(1,i-1) - data.latitudes(1, i)); %distance to move forward
        distX = sqrt((abs(data.longitudes(1, i-1) - data.longitudes(dataPointsInLine, i)))^2
+ distY^2); % width of the road
        y = [0, distY];
        x = linspace(0, distX, dataPointsInLine);

        %calculate milling volume, everything above {baseline - road thickness}
        millingSurf = data.smoothedHeights(:, i);
        millingSurf(millingSurf<-roadThickness) = -roadThickness;
        millingSurf = millingSurf + roadThickness;
        volumes(1, i-1) = abs(trapz(x, trapz(y, [previousMillingSurf millingSurf]')));
        previousMillingSurf = millingSurf;

        %calculate filling volume, everything below {baseline - road thickness}
        fillingSurf = data.smoothedHeights(:, i);
        fillingSurf(fillingSurf>-roadThickness) = -roadThickness;
        volumes(2, i-1) = abs(trapz(x, trapz(y, [previousFillingSurf fillingSurf]')));
        previousFillingSurf = fillingSurf;
    end
end
```

Code 2 Calculating filling and milling volume

```
function [surfaceAreas] = calculateSurfaceAreas(relHeights, longitudes, indexes)
    surfaceAreas = zeros(2, size(relHeights, 2));
    for n = 1 : size(relHeights, 2)
        %calculate upper area
        tmp = relHeights(:, n);
        tmp(tmp(:, 1) <= 0) = 0;
        surfaceAreas(1, n) = trapz(longitudes(indexes, n), tmp);

        %calculate lower area
        tmp = relHeights(:, n);
        tmp(tmp(:, 1) >= 0) = 0;
        surfaceAreas(2, n) = -trapz(longitudes(indexes, n), tmp);
    end
end
```

Code 3 Calculating upper and lower surface areas

```
function [reducedLine, indexes] = radialSimplify(initialLine, tolerance)
    ......
    count = 1;
    for i = 2 : size(initialLine, 1)
        point = initialLine(i);
        if(point == 0 %workaround to keep all the points from other area
             || getDistanceBwElements(point, prevPoint) > tolerance
             || i == size(initialLine, 1))
            count = count + 1;
            indexes(count) = i;
            reducedLine(count) = point;
            prevPoint = point;
        end
    end
    ......
    function dist = getDistanceBwElements(point, prevPoint)
        dist = abs(point - prevPoint);
    end
end
```

Code 4 Radial simplify algorithm

```
function [compressedHeights, indexes] = preCompressLine(relHeights, longitudes,
surfAreas, approxPrecisionLoss)

    %PRE-PROCESS UPPER AREAS
    tol = 0.0005;
    condition = true;
    tmpUpper = relHeights;
    tmpUpper(tmpUpper < 0) = 0;
    while(condition)
        [tmpCompressed, tmpUpperIdx] = radialSimplify(tmpUpper, tol);
        tmpSurfAreas = calculateSurfaceAreas(tmpCompressed, longitudes,
tmpUpperIdx);
```

```
        tmpPrecisionLoss = 100 * abs((surfAreas(1) - tmpSurfAreas(1)) /
surfAreas(1));
        if(tmpPrecisionLoss < approxPrecisionLoss)
            condition = false;
            tmpUpperIdx = tmpUpperIdx(tmpCompressed>0);
            tmpUpper = tmpUpper(tmpUpperIdx);
        else
            tol = tol / 2;
        end
    end

    %PRE-PROCESS LOWER AREAS
    ......

    %COMBINE UPPER AND LOWER AREAS
    indexes = zeros(size(relHeights, 1), 1);
    indexes(tmpLowerIdx) = tmpLowerIdx;
    indexes(tmpUpperIdx) = tmpUpperIdx;
    indexes = indexes(indexes > 0);
    compressedHeights = relHeights(indexes);
end
```

Code 5 Pre-compressing upper and lower area of data using radial simplify

```
function [robustHeights] = removeOutliers(heights)
    neighboringElements = 3;
    nsigma = 3;
    robustHeights = hampel(heights, neighboringElements, nsigma);
end
```

Code 6 Remove outliers from input

```
function plot3dMesh(lat, lon, measurement, measurementLabel)
    [lat_idx,~,simplified_lat] = unique(lat);
    [lon_idx,~,simplified_lon] = unique(lon);
    Z = accumarray([simplified_lat simplified_lon], measurement, [], @mean);
    Z(Z == 0) = NaN;
    mesh(lon_idx, lat_idx, Z);
    xlabel('Longitude');
    ylabel('Latitude');
    zlabel(measurementLabel);
end
```

Code 7 Plotting measurement as a 3D surface

```
relHeightsAdj = reshape(relHeights(:), [dataPointsInLine * totalDataLines, 1]);
```

```
latsAdj = reshape(latitudes, [dataPointsInLine * totalDataLines, 1]);
longsAdj = reshape(longitudes, [dataPointsInLine * totalDataLines, 1]);
lightsAdj = reshape(lights, [dataPointsInLine * totalDataLines, 1]);
scatter(latsAdj, longsAdj, [], relHeightsAdj);
```

Code 8 Plotting measurement as a 2D scatter plot

```
dataPointsInLine = 540;
totalDataLines = size(data, 1) / dataPointsInLine;
heights = reshape(data(:, 3), [dataPointsInLine, totalDataLines]);
lights = reshape(data(:, 4), [dataPointsInLine, totalDataLines]);
longitudes = reshape(data(:, 2), [dataPointsInLine, totalDataLines]);
latitudes = reshape(data(:, 1), [dataPointsInLine, totalDataLines]);
```

Code 9 Reformating to two-dimensional matrix

```
function [reducedLine, indexes] = dpSimplify(initialLine, initialIndexes, epsilon)
    ......
    function [reducedPtList] = dpSimplifyRec(ptList, n)
    ......
        %Find the max perpendicular distance from the line between the edges
        for k = 2:n-1
            d = perpendicularDistance(ptList(k,:), ptList([1,n],:));
            if d > dmax
                dmax = d;
                idx = k;
            end
        end
        %If max distance is greater than epsilon, recursively simplify
        if dmax > epsilon
            recList1 = dpSimplifyRec(ptList(1:idx,:), idx);
            recList2 = dpSimplifyRec(ptList(idx:n,:), n-idx+1);
            reducedPtList = [recList1;recList2(2:end,:)];
        else
            reducedPtList = ptList([1,n],:);
        end
    end

    function d = perpendicularDistance(pt, lineNode)
        ......
    end
end
```

Code 10 Douglas-Peucker algorithm to simplify a line

```
function [compressedRelHeights, idx, precisionLoss, dpCount] = compressLine
      (relHeights, longitudes, indexes, surfArea, approxPrecisionLoss, allowedDiff, isUpper)
```

```
    ......
   % 1) when decreasing tolerance (and thus precision) we continue until precision is
smaller than acceptable precision loss
   % 2) when initial compression and the loss is bigger than acceptable
     % 2.1) and bigger than max allowed, then we decrease tolerance
     % 2.2) but smaller than max allowed then we end the while loop
   % We give a small buffer of 1.5x the acceptable precision loss on the initial compression
   condition = true;
   while (condition)
       [tmpCompressedRelHeights, tmpIdx] = dpSimplify(tmpCompressedRelHeights, tmpIdx, tol);
       dpCount = dpCount + 1;
       tmpSurfAreas = calculateSurfaceAreas(tmpCompressedRelHeights, longitudes, tmpIdx);
       if(isUpper); tmpSurfArea = tmpSurfAreas(1); else; tmpSurfArea = tmpSurfAreas(2); end
       if(abs(tmpSurfArea - surfArea) < ATM_ALLOWED_DIFF)
           tmpPrecisionLoss = approxPrecisionLoss - 0.01;
       else
           if(isUpper); tmpPrecisionLoss = 100 * abs(1-(tmpSurfArea/surfArea));
           else; tmpPrecisionLoss = 100 * abs(1-(tmpSurfArea/surfArea)); end
       end
       if(precisionLoss == 0 && tmpPrecisionLoss > approxPrecisionLoss)
           if(tmpPrecisionLoss < MAX_PRECISION_LOSS_ALLOWED)
               assignOutput(tmpCompressedRelHeights, tmpIdx, tmpPrecisionLoss);
               return;
           else %first compression was too much, rewind and start with a smaller tolerance
               tol = tol/2;
               tmpIdx = indexes;
               tmpCompressedRelHeights = relHeights;
               tmpPrecisionLoss = 0;
           end
       else
           tol = tol*2;
       end
       if(tmpPrecisionLoss < approxPrecisionLoss)
           assignOutput(tmpCompressedRelHeights, tmpIdx, tmpPrecisionLoss);
           if(dpCount > 5) %will skip further compression to increase performance
               condition = false;
           end
       else
           condition = false;
       end
   end

   function assignOutput(tmpCompressedRelHeights, tmpIdx, tmpPrecisionLoss)
       ......
   end
end
```

Code 11 Compressing a line using Douglas-Peucker until acceptable precision is achieved

```
function [compressedData] = compressData(data, approxPrecisionLoss, allowedErrorInMetresSq)
  ......
```

```
  tic %start timer
  parfor i = 1 : totalDataLines %use parallel computing to increase performance
      %pre compress data with radial distance: A
      [preCompressedHeights, preCompressedIndexes] = preCompressLine(
          smoothedHeights(:, i), longitudes(:, i), surfAreas(:, i), approxPrecisionLoss);

      %compress upper part: B
      tmpUpper = preCompressedHeights;
      tmpUpper(tmpUpper<0) = 0;
      [tmpCompressedUpper, tmpUpperIdx, diffUpper(1, i), dpCountUpper(1, i)] = compressLine(
          tmpUpper, longitudes(:, i), preCompressedIndexes, upperSurfAreas(i),
approxPrecisionLoss, allowedErrorInMetresSq, true);

      idxUpper = tmpUpperIdx(tmpCompressedUpper>0);

      %compress lower part: B
      ......

      %combine upper and lower part through the extracted indexes: C
      tmpHeights = smoothedHeights(:, i);
      indexes = zeros(dataPointsInLine, 1);
      indexes(idxLower) = idxLower;
      indexes(idxUpper) = idxUpper;
      idx{i} = indexes(indexes > 0);
      compressedRelHeights{i} = tmpHeights(idx{i});
  end

  % calculate the total size of the result (total data points): D
  totalSize = 0;
  for i = 1 : data.totalDataLines
      totalSize = totalSize + size(idx{i}, 1);
  end

  % calculate the indexes in an adjusted 1-dimensional form: D
  j = 1;
  for i = 1 : size(data.relHeights, 2)
      compressedData.indexesAdj(j:(j-1) + size(idx{i}, 1), 1) = (i-1) * 540 + idx{i};
      j = j + size(idx{i}, 1);
  end

  % combine the output struct
  ......
end
```

Code 12 Pre-compress (a), compress upper, lower parts (b), merge compressions (c)

Appendix 2 – tables

		Dissimilarity, smaller means similar			Inverse of correlation coefficient, smaller means similar			Sum of squared errors, smaller means similar			Coefficient
Algorithm	Smoothing factor	Haussdorff diffMed	Hausdorff diffAvg	Hausdorff diffMax	Pearson Inv diffMed	Pearson Inv diffAvg	Pearson Inv diffMax	SSD diffMed	SSD diffAvg	SSD diffMAx	product
5MA	0.00040	0.00260	0.00284	0.01360	0.00248	0.00284	0.00109	0.00160	0.00171	0.00425	3.57529
7MA	**0.00029**	0.00300	0.00339	0.01643	0.00293	0.00337	0.00129	0.00189	0.00203	0.00517	12.01419
10MA	**0.00020**	0.00360	0.00397	0.01888	0.00328	0.00378	0.00147	0.00213	0.00228	0.00602	28.59351
14Ma	**0.00021**	0.00420	0.00469	0.01888	0.00357	0.00407	0.00160	0.00230	0.00246	0.00644	67.43193
Lowess 3	0.00200	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
Lowess 5	0.00071	0.00157	0.00185	0.01140	0.00142	0.00157	0.00058	0.00091	0.00095	0.00203	0.05324
Lowess 7	0.00045	0.00212	0.00239	0.01266	0.00197	0.00222	0.00083	0.00127	0.00134	0.00306	0.54020
Lowess 10	**0.00027**	**0.00275**	**0.00308**	**0.01600**	**0.00262**	**0.00300**	**0.00118**	**0.00169**	**0.00181**	**0.00451**	**4.57502**
Lowess 13	**0.00023**	0.00294	0.00329	0.01749	0.00283	0.00325	0.00128	0.00183	0.00196	0.00502	8.05597
Loess 3	0.00200	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
Loess 5	0.00200	0.00045	0.00058	0.00455	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
Loess 7	0.00092	0.00130	0.00150	0.00980	0.00120	0.00131	0.00047	0.00079	0.00079	0.00168	0.01365
Loess 10	0.00051	0.00194	0.00223	0.01231	0.00192	0.00216	0.00080	0.00124	0.00130	0.00304	0.44372
Loess 13	0.00042	0.00217	0.00249	0.01459	0.00217	0.00246	0.00092	0.00140	0.00148	0.00351	1.19252
Sgolay (10per, 9deg)	0.00200	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
Sgolay (20per, 11deg)	0.00076	0.00120	0.00146	0.01104	0.00163	0.00182	0.00070	0.00105	0.00110	0.00259	0.09133
Sgolay (30per, 11deg)	0.00046	0.00160	0.00189	0.01080	0.00230	0.00261	0.00101	0.00149	0.00158	0.00376	0.81015
Sgolay (40per, 11deg)	**0.00033**	**0.00187**	**0.00218**	**0.01411**	**0.00267**	**0.00306**	**0.00121**	**0.00173**	**0.00185**	**0.00478**	**2.89923**
Sgolay (50per, 11deg)	**0.00027**	**0.00206**	**0.00238**	**0.01305**	**0.00291**	**0.00333**	**0.00130**	**0.00188**	**0.00201**	**0.00507**	**4.09598**
Sgolay (60per, 11deg)	**0.00023**	**0.00223**	**0.00253**	**0.01424**	**0.00305**	**0.00351**	**0.00138**	**0.00197**	**0.00211**	**0.00531**	**5.95300**
Gaussian 0.5	0.00101	0.00152	0.00171	0.00947	0.00058	0.00067	0.00028	0.00042	0.00046	0.00333	0.00171
Gaussian 1.0	**0.00033**	0.00318	0.00354	0.01968	0.00278	0.00321	0.00135	0.00204	0.00237	0.02451	103.40797
Gaussian 1.5	**0.00021**	0.00375	0.00412	0.02214	0.00342	0.00395	0.00162	0.00258	0.00307	0.03584	441.47657
Gaussian 2.0	**0.00017**	0.00413	0.00449	0.02318	0.00371	0.00427	0.00174	0.00286	0.00348	0.04244	868.32095

Table 3 Smoothing comparison

		Dissimilarity, smaller means similar			Inverse of correlation coefficient, smaller means similar			Sum of errors, smaller means similar			
Algorithm	Smoothness	Haussdorff diffMed	Hausdorff diffAvg	Hausdorff diffMax	Pearson diffMed	Pearson diffAvg	Pearson diffMax	SSD diffMed	SSD diffAvg	SSD diffMAx	Coefficient
Lowess 10 - 1	0.00027	0.00275	0.00308	0.01600	0.00262	0.00300	0.00118	0.00169	0.00181	0.00451	
Lowess 10 - 2	0.00013	0.00059	0.00102	0.20704	0.00189	0.00226	0.06157	0.00004	0.00012	0.11838	
Lowess 10 - 3	0.00021	0.00361	0.00379	0.01876	0.08712	0.09910	0.33718	0.00129	0.00134	0.00265	
Lowess 10 -Avg	0.00020	0.00232	0.00263	0.08060	0.03054	0.03479	0.13331	0.00101	0.00109	0.04185	6.48620
Sgolay 40/11 -1	0.00033	0.00187	0.00218	0.01411	0.00267	0.00306	0.00121	0.00173	0.00185	0.00478	
Sgolay 40/11 -2	0.00016	0.00034	0.00043	0.03440	0.00131	0.00163	0.05752	0.00003	0.00005	0.01976	
Sgolay 40/11 -3	0.00028	0.00318	0.00335	0.01896	0.08840	0.10070	0.32233	0.00131	0.00136	0.00260	
Sgolay 40/11 - Avg	0.00025	0.00180	0.00199	0.02249	0.03079	0.03513	0.12702	0.00102	0.00108	0.00905	0.28213
Sgolay 50/11 -1	0.00027	0.00206	0.00238	0.01305	0.00291	0.00333	0.00130	0.00188	0.00201	0.00507	
Sgolay 50/11 -2	0.00015	0.00037	0.00047	0.04070	0.00158	0.00198	0.07327	0.00003	0.00008	0.02981	
Sgolay 50/11 -3	0.00021	0.00349	0.00365	0.01961	0.09635	0.11033	0.40427	0.00142	0.00147	0.00306	
Sgolay 50/11 - Avg	0.00021	0.00197	0.00217	0.02445	0.03361	0.03855	0.15961	0.00111	0.00119	0.01265	0.74705
Sgolay 60/11 -1	0.00023	0.00223	0.00253	0.01424	0.00305	0.00351	0.00138	0.00197	0.00211	0.00531	
Sgolay 60/11 -2	0.00013	0.00044	0.00057	0.05479	0.00247	0.00316	0.13913	0.00005	0.00015	0.09330	
Sgolay 60/11 -3	0.00017	0.00371	0.00388	0.02014	0.10158	0.11643	0.42690	0.00149	0.00155	0.00316	
Sgolay 60/11 - Avg	0.00018	0.00213	0.00233	0.02972	0.03570	0.04103	0.18914	0.00117	0.00127	0.03392	3.64350

Table 4 Smoothing comparison across 3 different datasets

Appendix 3 – smoothing comparisons

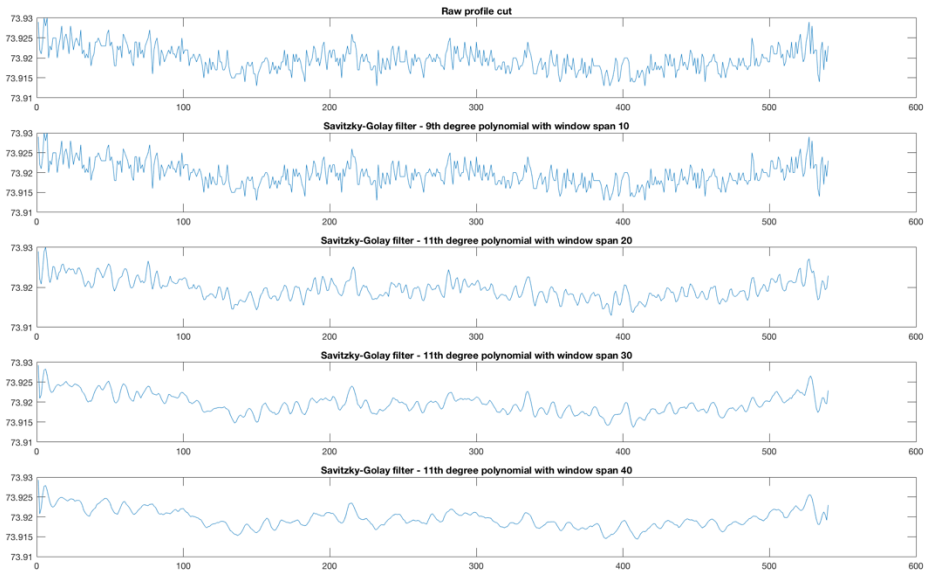

Figure 24 Savitzky-Golay filter on sample profile

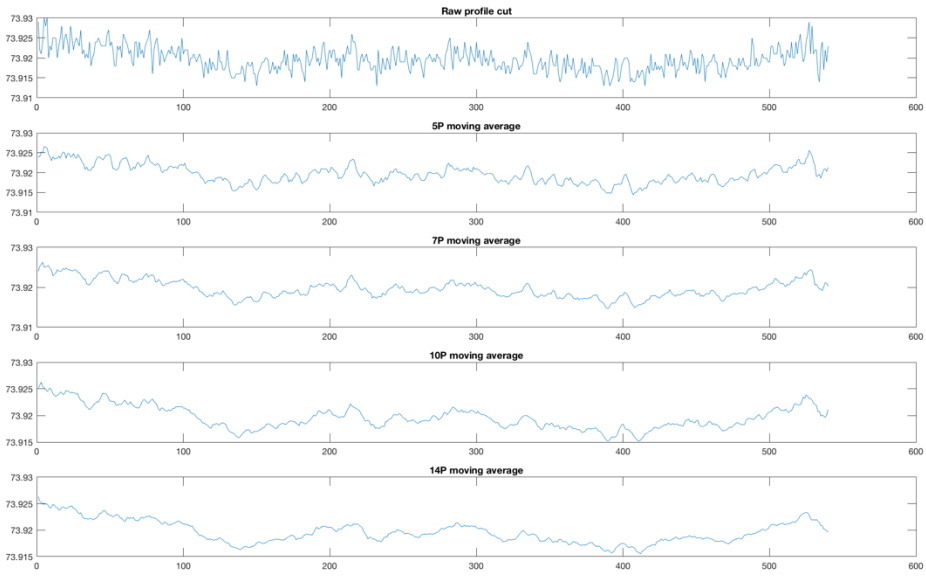

Figure 25 Moving average on sample profile

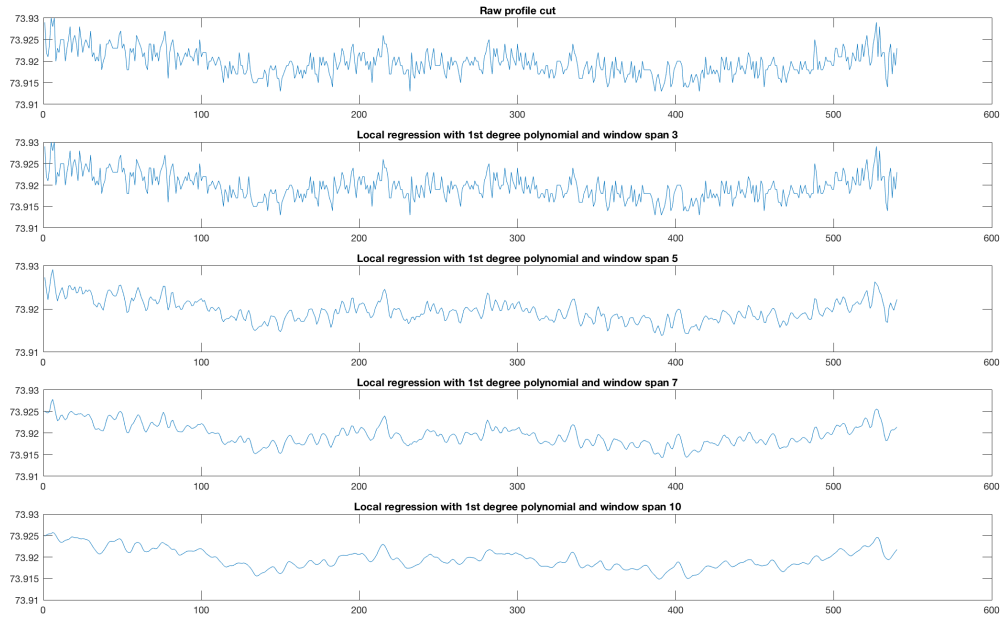

Figure 26 Local regression with 1st degree polynomial on sample profile

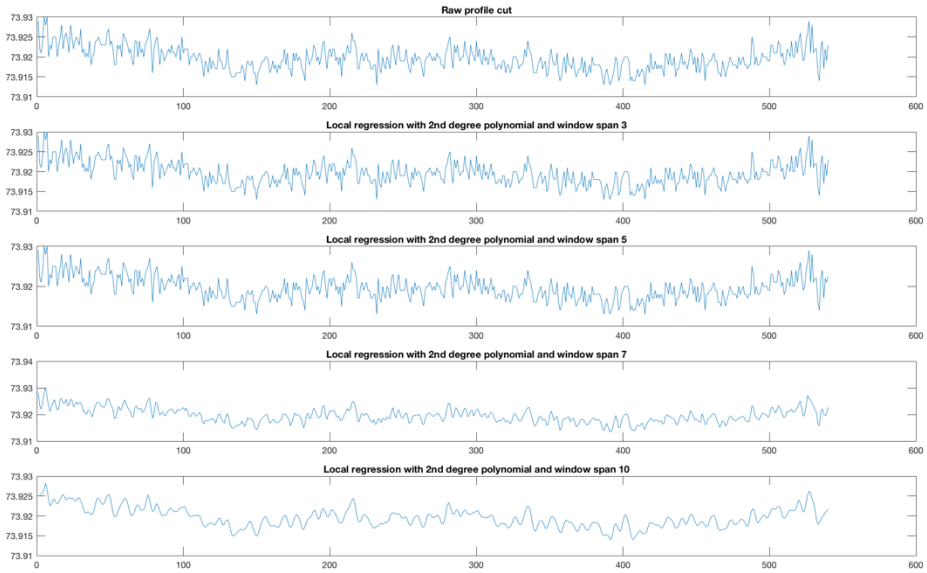

Figure 27 Local regression with 2nd degree polynomial on sample profile

Appendix 4 – visualizations

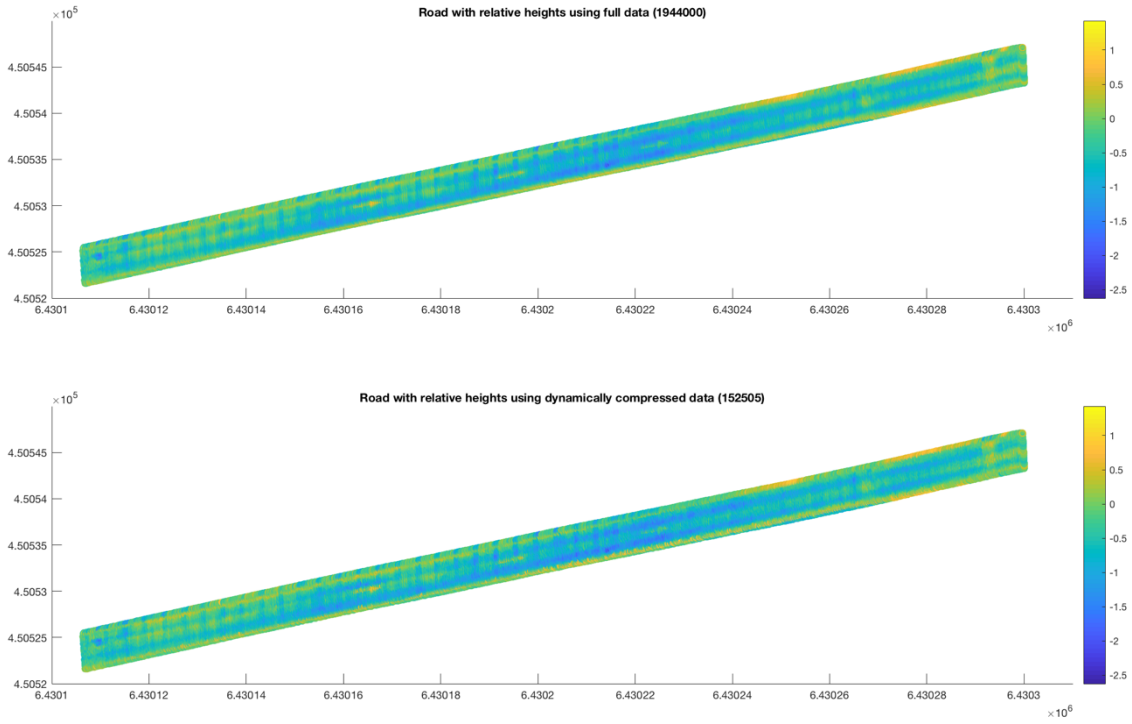

Figure 28 Sample road before and after compression (with a rate of 12.7x)

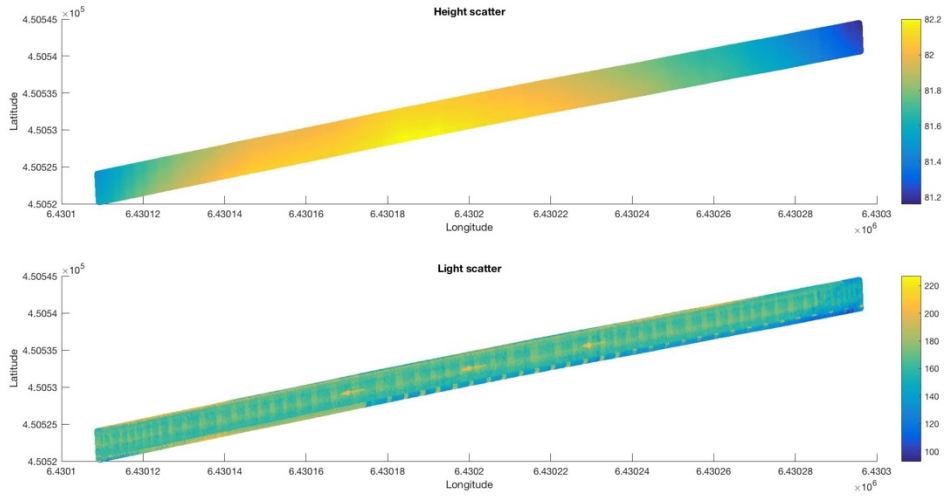

Figure 29 2D scatter plot of height and light

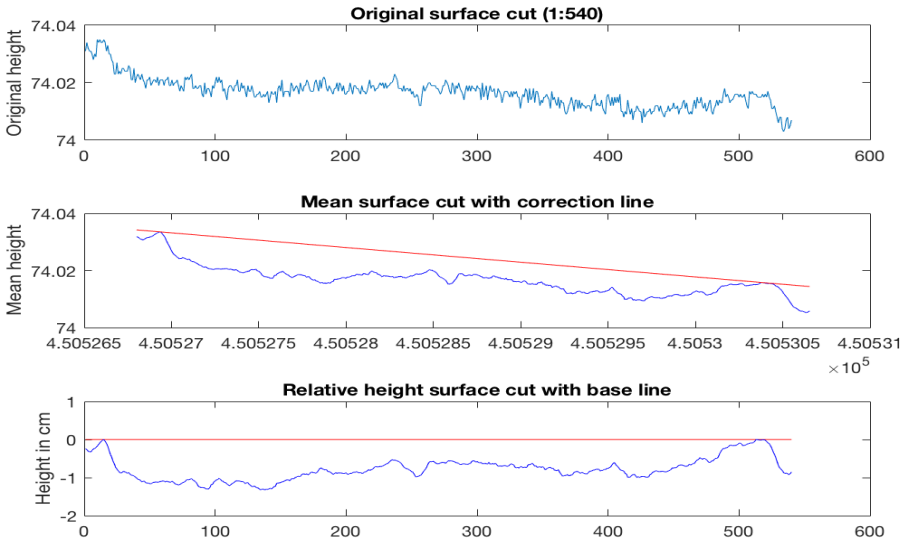

Figure 30 Using high-edge method to calculate relative heights

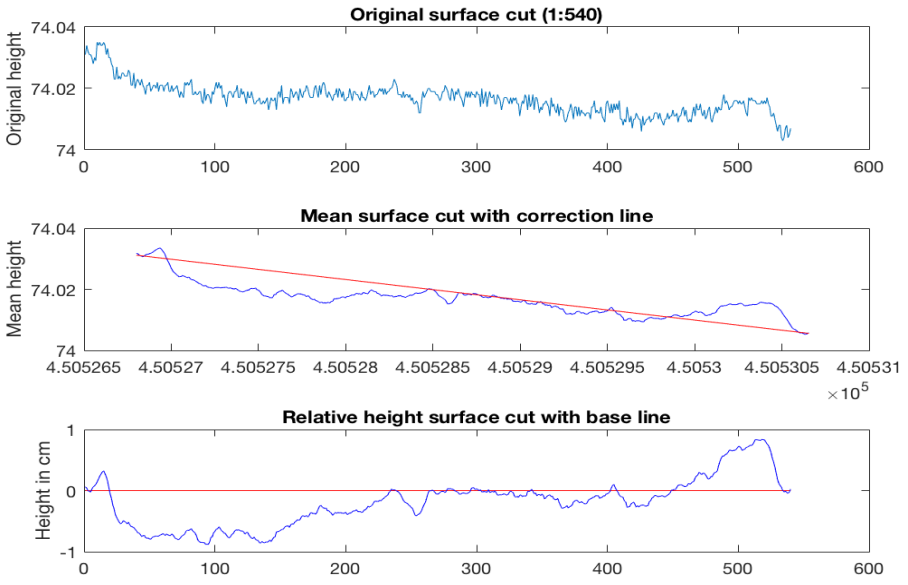

Figure 31 Using simple-edge method to calculate relative heights

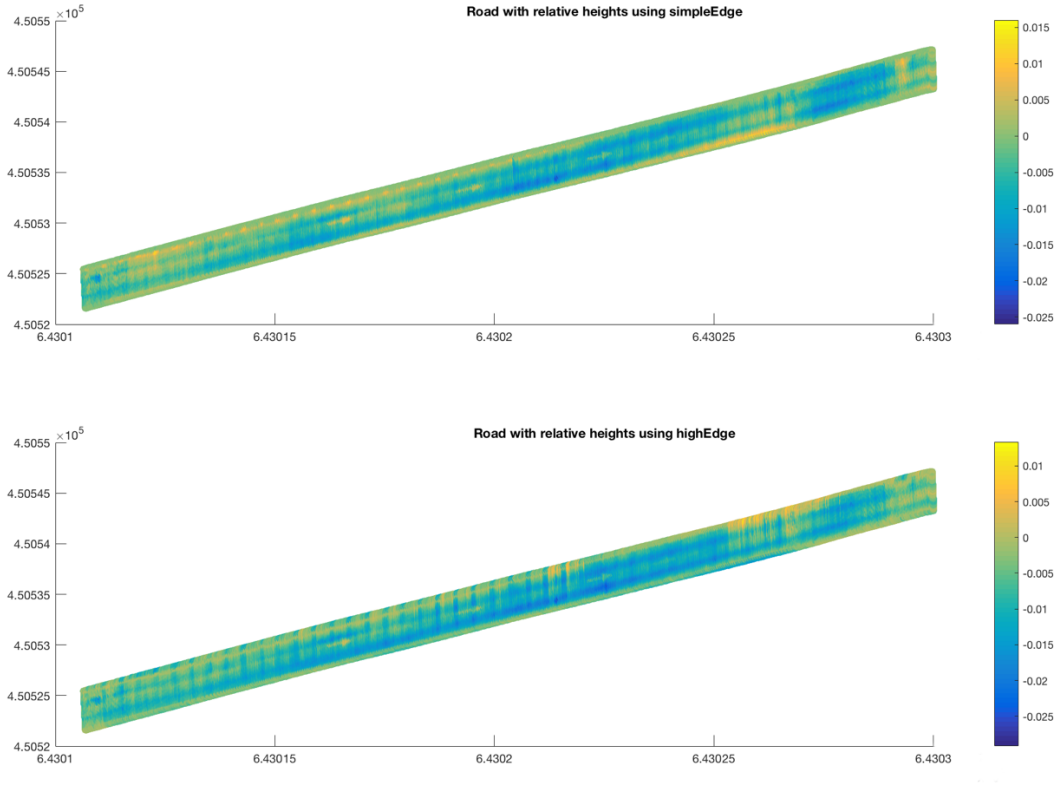

Figure 32 Visualisation comparison of simple and high edge method