A thesis submitted in fulfillment of the requirements for the degree of Master of Science (M.Sc.) in Applied Geoinformatics.

Department of Environmental Remote Sensing & Geoinformatics University of Trier

Automatic Detection of Line Structures in Airborne LiDAR Data

First supervisor	Prof. Dr. Thomas Udelhoven
Second supervisor	Dr. Johannes Stoffels
Author	Thomas Rommel
Submitted	July 27, 2015
Matriculation number	1103878
Mail address	thomasrommel89@gmail.com
City	54295 Trier

ISPRS Int. J. Geo-Inf. **2015**, *xx*, 1-x; doi:10.3390/——

OPEN ACCESS

ISPRS International Journal of
Geo-Information
ISSN 2220-9964
www.mdpi.com/journal/ijgi

Article

Automatic Detection of Line Structures in Airborne LiDAR Data

Thomas Rommel

Department of Environmental Remote Sensing and Geoinformatics, University of Trier, Behringstraße 21, 54296 Trier, Germany

Received: xx / Accepted: xx / Published: xx

Abstract: Airborne LiDAR-derived digital terrain models can be used to detect skid trails overgrown by dense forest vegetation. This study proposes a model approach for an automatic detection and vectorization of skid trails using a 0.5 meter digital terrain model (DTM) of Rhineland-Palatinate, Germany. Due to the steep terrain and the resulting terrain height differences, a moving window filter method, using the height differences along a defined distance to extract skid trails from the DTM, is created. For the vectorization of the resulting binary image, digitized start lines are used for an automatic connection of related nodes, with a given search angle and a maximum distance. Appropriate results are obtained and evaluated by features derived from hillshades of the test area. In comparison to an active contour model, the derived feature had a length of 2,193 meters, with a completeness value of 67 percent, a correctness value of 68 percent and a root mean squared error value of 2.7 meters.

Keywords: airborne LiDAR; digital terrain model (DTM); feature detection; line structure detection; moving window filter; skid trail detection

1. Introduction

Forest roads are one of the most important parts of forestry and forest management. They are not only used for traveling in the forest, but for much more purposes and depending on the task of the roads, their size and structure differ. The smallest kind of roads in the forest, the skid trails, which are mainly created for logging, are the center of this study.

Skid trails are used by heavy machinery, ranging between 10 to 25 tonnes [1,2], and therefore,

different aspects, concerning geomorphic and hydrologic effects occur in direct proximity or on a larger scale. These concerns are soil disturbance and compaction, reduction of species richness and plant recovery [3]. Furthermore, they can be the cause of sedimentation and erosion in catchment areas by redirecting surface flows [4,5].

The knowledge about the location of skid trails plays an important role in Germany, which is covered by 32 percent of forest [6]. In order to minimize the described problems concerning skid trails, and to comply with PEFC or FSC standards [7,8], it is necessary to detect, localize and reuse those trails. Especially in the test area, which is located in the national park Hunsrück-Hochwald, sustainable forestry techniques need to be developed and utilized. The large forested areas in Germany, and in the the federal state Rhineland-Palatinate with more than 839,000 hectares, causing massive problems for foresters to catalogue existing trails. The sheer size of the area and the missing of expert knowledge are the main reasons for an automatic and fast detection method of skid trails. Furthermore, a lot of the desired structures are not visible in the terrain, because they have already been overgrown by plants and trees. The terrain and tall trees also complicate the use of GPS to manually track skid trails on-site [9]. Therefore, the used data must be available on a large scale, must have a high spatial resolution and has to be able to detect the forest soil. Airborne derived Light Detection and Ranging (LiDAR) data is used in remote sensing to acquire topographic data on a large scale with a high accuracy. LiDAR is highly valuable for forested areas by providing the ground surface elevation [10]. Airborne LiDAR data has been the topic for many different research topics like harvesting and road design [11], archaeology [12] or for resource managers [13].

A lot of research and projects have been carried out with the general topic of feature detection utilizing different techniques and data. White, *et al.* applied LiDAR derived digital elevation models and their products to manually digitize forest roads in steep terrain by visually interpreting the contrast differences [14]. Hierarchical classification techniques are commonly used to extract roads from LiDAR point clouds. Clode, *et al.* used this technique to classify lidar points into non-road and desired road points. A combined method employing the height and intensity value of the LiDAR input data was presented. Furthermore, the consistency and homogeneity was made use of to finalize the method [15,16]. Computer vision and computer graphics methods were used by Rieger, *et al.* to differentiate between roads and non-roads in digital elevation model derived slope models. Edge extraction filters were applied to detect break lines. Due to the noisy nature of the slope model, pre-processing of the input like edge-enhancing filters were applied [17].

Not only LiDAR derived data is used for feature detection, but also high resolution multispectral imagery. Gradient operations and the skeletal ray formation is adopted by Mangala and Bhirud for an automatic road extraction. Furthermore, thresholding and filtering is carried out. The final roads are extracted by morphological operations of the colored result [18]. Doucette, *et al.* also applied gradient operations for an automated road extraction algorithm [19]. The main centerline of road features in high resolution satellite imagery is extracted by Hu and Tao applying a hierarchical grouping strategy technique, which does not group all parts at once, but they are grouped incrementally [20]. Road seeds and vectorial information are also utilized for road detection [21,22].

More complex techniques for feature detection are *Active Contour Models*. Introduced by Cohen [23], active contour models are used in wide range of different fields. Several variations have been

developed [24–27] for different tasks like tracking of moving objects [28], feature detection [29] and medical informatics [30–35]. Active contour models are also used in remote sensing for feature extraction. Seppke, *et al.* applied active contours to extract coastlines in Synthetic Aperture Radar, SAR images [36]. The segmentation of roads was the purpose of several papers [37–40].

2. Material and Methods

2.1. Coordinate System

This study uses the the following coordinate system and therefore, all maps have the following coordinate system and projection.

Table 1. Coordinate system and projection of the case study

Variable	**Value**
Coordinatesystem	WGS 84 / UTM zone 32N
Projection	Transverse Mercator
Date	WGS 1984
Unit	Meter
EPSG	32632

2.2. Study Area

The study site is located in the western part of Rhineland-Palatinate, Germany and about 30 kilometers east of the state's capital Trier. It is in the center of the triangular arrangement of the three villages Deuselbach (northwest), Allenbach (northeast) and Tranenweiher (south). With a border length of 290.8 meters and a width of 211.7 meters, the place includes an area of 61,559.2 square meters. It is part of the recently created national park Hunsrück, with a size of roughly 10,000 hectares [41]. Parts of the national park Hunsrück consist out of bogs. The chosen test area is wet as well. Figure 1 shows the location of the test site and the national park on an European and on a regional scale. Ranging from 656 to 684 meters above sea level, the elevation depends on the direction. Comparing the four chosen directions (center north to south, northwest to southeast, northeast to southwest and center west to the east of the test site), which are displayed in figure 2, it can be said that, the north to south and northwest to southeast extents are more steep than the other two, which are directly compared, more flat. The test area is densely vegetated (see figure 1). In Germany, 32 percent of the state (11.4 Mio. hectares), are forest areas. Rhineland-Palatinate, in which the test area is located, is covered by 42 percent (839,796 hectares) of forest [6]. Typical of the greater region, the Hoch- and Iderwald, are woodrush-beech forests (*Luzulo-Fagetum*) with atlantic birch mires. Today's forest is defined by a large amount of coniferous trees, mainly *Picea abies*, which extends from southwest to northeast. Spruce form the predominant tree species in those forests. Last amounts of beeches, which define the natural vegetation, are still existing near the former forestry departments Hermeskeil and Kempfeld [42]. The maximum tree heights are

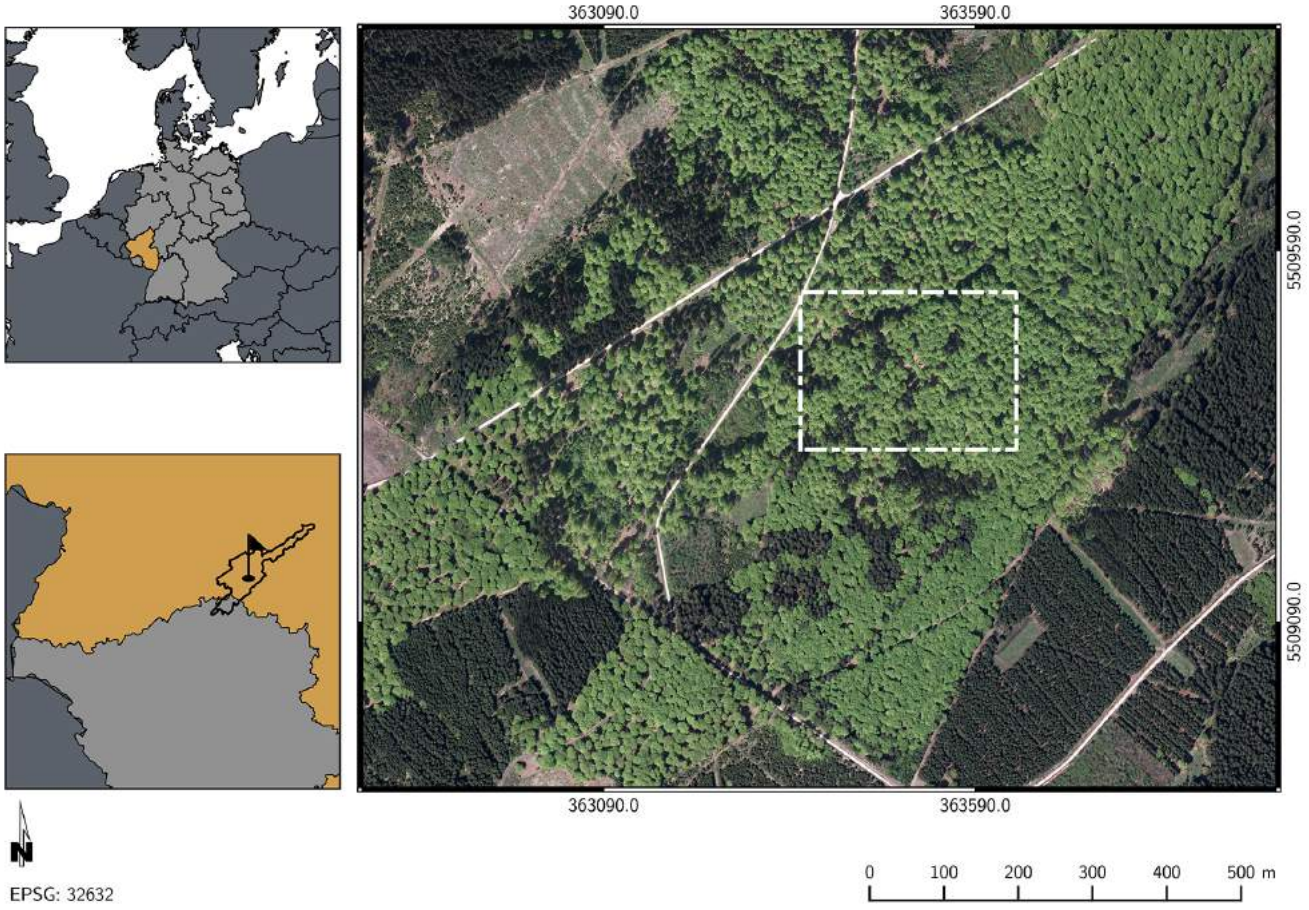


Figure 1. Test area (white) and national park boundaries (black) shown within the context of Germany and Europe

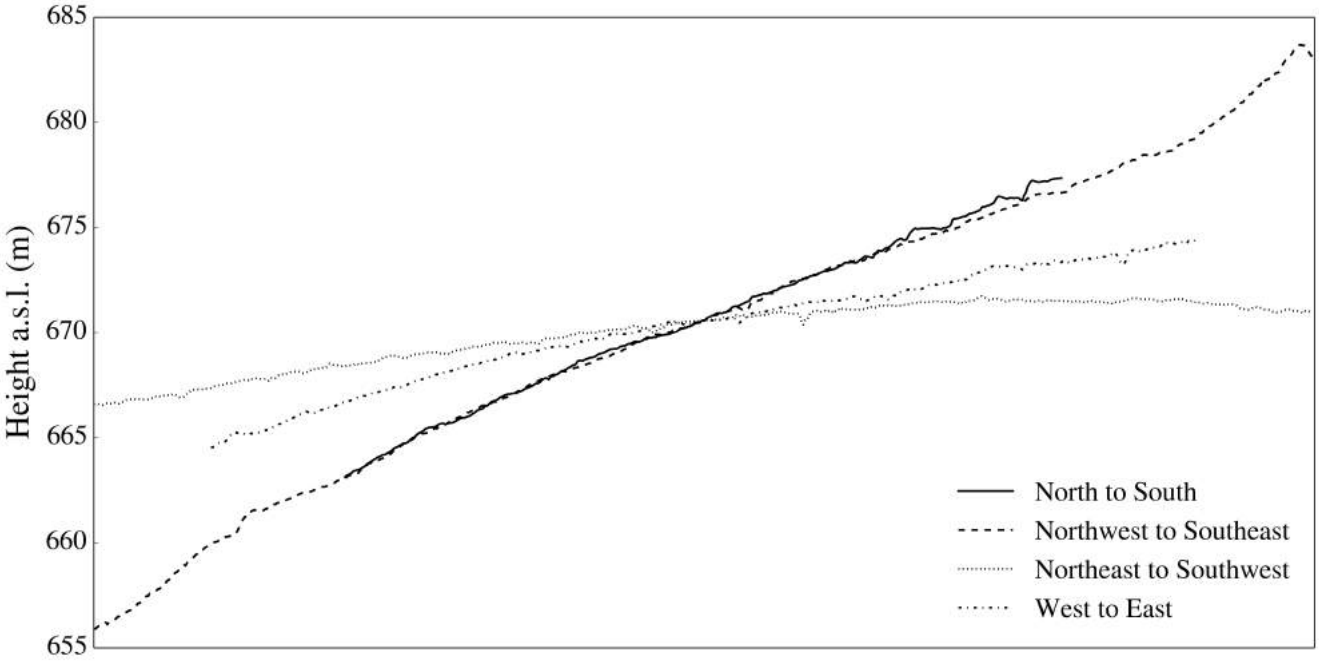
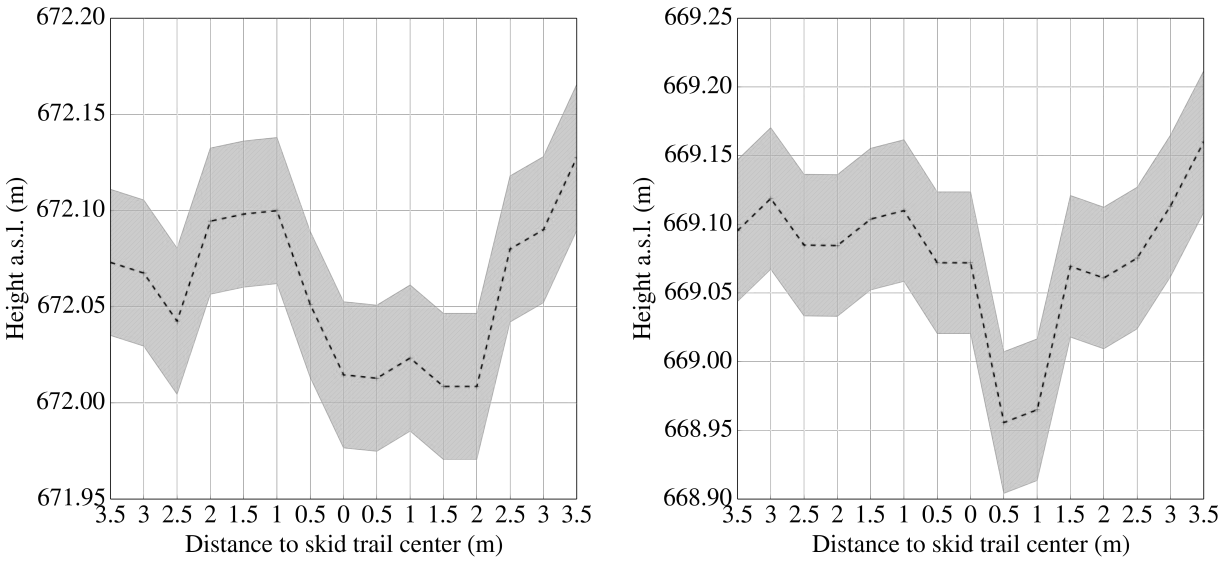


Figure 2. Terrain height of the study site in meter above sea level for four directions

about 35 meters with a mean of 16.7 meters and a standard deviation of 9.8 meters.
This particular test area was chosen due to the skid trail courses, parallel, and on a first glance arbitrary (see figure 5), and the quality of the data. No visual errors were recognized in this test site. The size of the area was a trade-off between running time of the algorithm and the existing amount of skid trails.

2.3. Skid Trails

Skid trails are unpaved and open roads in the forest, which are used by heavy machinery e.g. harvesters and forwarders. Normally, these roads are created without any structural measures in a distance between 20 to 60 meters, and are used every three to ten years. In order to reach selected trees to cut down and harvest, these roads are needed and grant a regulated removal of the logs to the next forest road [43]. Figure 4a shows the target road feature of this study. Due to the dense vegetation of the test area, these roads are not visible on aerial or satellite imagery. Field measurements revealed a basic width of up to 4 meters, and a maximum depth of 30 to 45 centimeters. Perpendicular height profiles of skid trails can confirm these measurements (see figure 3). Skid trails are deeper than the surrounding forest floor,



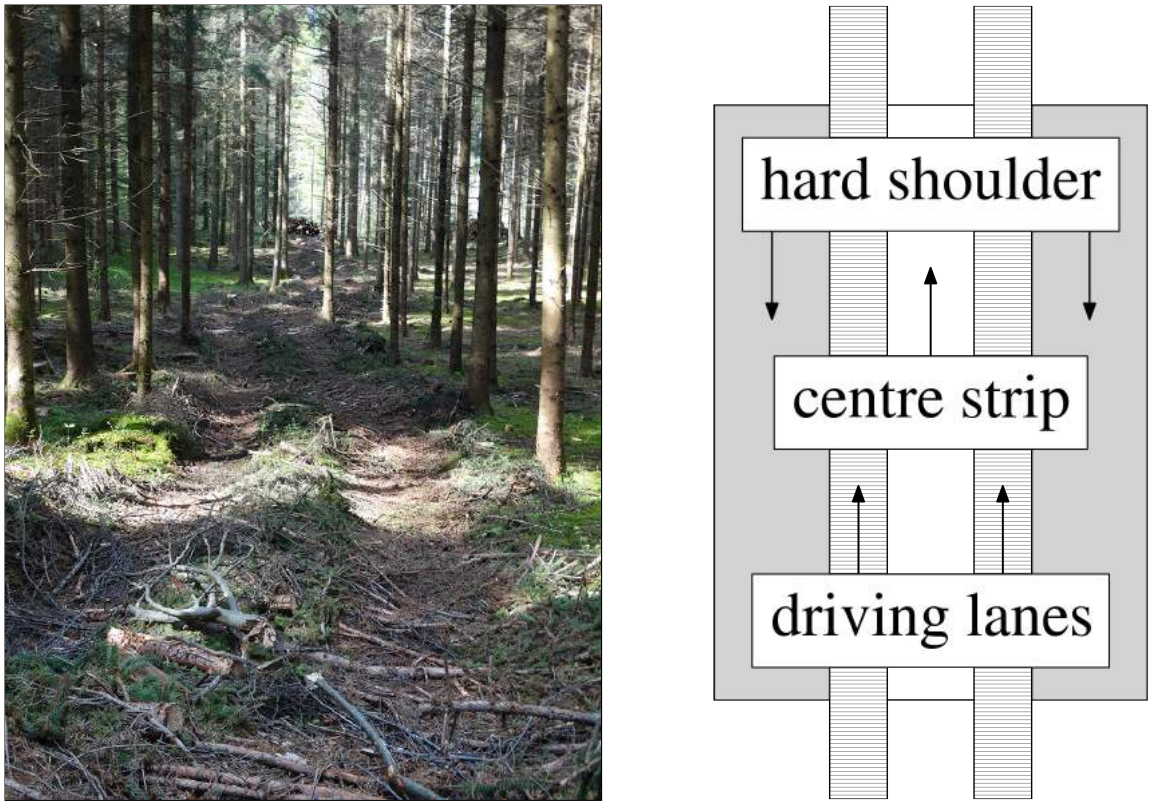
(a) First skid trail example

(b) Second skid trail example

Figure 3. Height profile for two selected skid trails of the test area. Gray hatched area above and under the actual height line shows one standard deviation more and less of the height

because they are used by heavy machinery, e.g. 17.5 t [1] - 22.8 t [2] for logging. The height profile can be seen in high resolution digital terrain models. Skid trails consist of three parts. The first part, the middle of the trail, called center strip, has the same height as the surrounding forest soil and is not visible in digital terrain models. The second part, the driving lanes, is defined by a lower height profile. On this part of the soil, the wheels touch the ground and compress the soil. The last part, the hard shoulders of the track, cannot always be well identified. It is defined as the part of the forest soil from the driving lanes to the adjacent forest stands. Figure 4b illustrates these three parts. The technique of using heavy machinery causes problems for the soil and adjacent forest. Several studies addressed this topic and deal with the occurring issues. These problems include soil compaction and disturbance, plant recovery, stem

densities and species richness, to name a few [3]. Furthermore, they are the cause of sedimentation and erosion in catchment areas. Regarding the missing drainage, skid trails increase the risk of redirecting surface flows or washing out of loose sediment [4].



(a) Target road feature: skid trail

(b) Schematic representation of a skid trail

Figure 4. Skid trail in the test area and the schematic representation (based on [43])

Terrain profiles of skid trails play an important role in the design of the developed algorithm. Figures 3a-3b show the height above sea level at a 90 degree angle from the center of the skid trail to 3.5 meters to each side of if. The gray hatched area above and under the actual height line shows one standard deviation more and less of the height. The creation of these graphics included two basic steps. First, the construction of the perpendicular lines and second, a point sampling along these lines with a certain distance between. Both required steps are described below.

For each line feature, three perpendicular lines are created (start point, middle point and end point). The perpendicular lines for these three points were calculated by two different methods. Both algorithms include the use of intersections of lines and circles.

To construct a perpendicular at the end of line $\overline{AB}$ (line equation: $y = m \cdot x + b$) (point A), a point above (or under) the line is created in a first step. The exact location of this point is not important. It is only needed as an auxiliary point D. Afterwards, the distance d between the auxiliary point D and A is computed and used for the creation of circle C (circle equation: $r^2 = (x - h)^2 + (y - k)^2$

). C is defined by its radius d and the center point A. In a next step, the intersection point I of circle C and line $\overline{AB}$ is calculated using equation 1.

$$x = (x - h)^2 + ((m \cdot x + b) - k)^2 - r^2 \tag{1}$$

This point is, like point D, used as an auxiliary point and is only needed to calculate the intersection K between line $\overline{ID}$ and circle C. Two points are needed to determine a line. In order to find the linear equation ($y = m \cdot x + b$), which is defined by the two points of line $\overline{AB}$, two variables, the slope m, and the y-intercept b, have to be calculated. The slope m of the line can be computed by applying the following equation:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{2}$$

Afterwards the calculated slope m is used to determine the y-intercept b. One point of line $\overline{AB}$ and slope m is employed to solve the ensuing equation:

$$b = y - m \cdot x \tag{3}$$

This equation is used in the following step to calculate the interception point K by applying formula 1. In a last step, the linear equation for line $\overline{KA}$ is computed, and used in equation 8 to determine the demanded length of the perpendicular length.

The other perpendicular line, which is based on the center of the line, can be computed by the following steps. First, the center of the line point K is used to create a circle C with a radius, which is smaller than half the length of the line $\overline{AB}$. Circle C is used to compute the two intersection points P and Q with line $\overline{AB}$. P and Q are afterwards used as the origin for two new circles, $C1$ and $C2$ with a radius larger than the length of $\overline{KP}$ or $\overline{KQ}$, but less than $\overline{PQ}$. With the use of equation 4, the intersection point R of both circles, $C1$ and $C2$, are calculated. R is part of the requested perpendicular line. The final length of the perpendicular can be determined by using equation 8.

$$x_1 = a + \frac{e \cdot k}{p} + \frac{f}{p} \cdot \sqrt{(r^2 - k^2)}\; y_1 = b + \frac{f \cdot k}{p} - \frac{e}{p} \cdot \sqrt{(r^2 - k^2)} \tag{4}$$

$$x_2 = a + \frac{e \cdot k}{p} - \frac{f}{p} \cdot \sqrt{(r^2 - k^2)}\; y_2 = b + \frac{f \cdot k}{p} + \frac{e}{p} \cdot \sqrt{(r^2 - k^2)}$$

A point along a line, with a certain distance d, from a point (x_0, y_0) of the line, can be computed by the use of vectors. The basic equation is defined as:

$$P = (x_0, y_0) + d \cdot u \tag{5}$$

In order solve to equation 5, two variables have to be calculated in a previous step. Let

$$v = (x_1, y_1) - (x_0, y_0) \text{ and } u = \frac{v}{\|v\|} \tag{6}$$

with the length of the vector v

$$\|v\| = \sqrt{v_1^2 + v_2^2} \tag{7}$$

After that, the equation for calculating a point along a line, with a certain distance, can be written as

$$(x_0, y_0) + d \cdot (\frac{v_1}{\sqrt{v_1^2 + v_2^2}}, \frac{v_2}{\sqrt{v_1^2 + v_2^2}}) \quad (8)$$

The use of vectors for the calculation has the advantage of not including a subtraction of values. If, for example x_0 equals x_1, no problem occurs by dividing by zero.

2.4. Raw Data

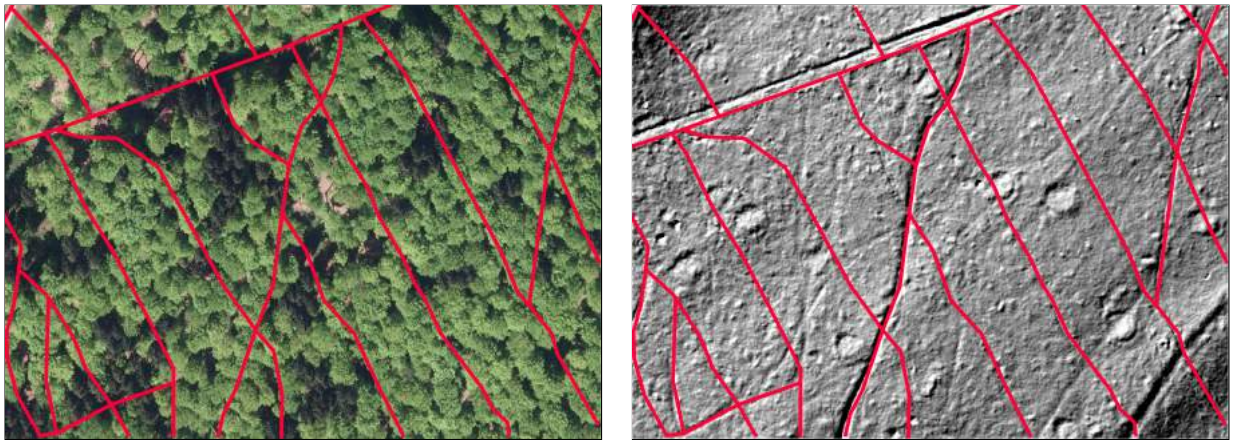
This paper is using an airborne LiDAR derived digital terrain model, DTM, from the *Landesamt für Vermessung und Geobasisinformation Rheinland Pfalz* [44]. For the whole federal state, the data was acquired between the years 2002 and 2013. The area, in which the test site is located, was part of the data acquisition project in the year 2008. The resulting digital terrain model has a spatial resolution with a pixel resolution of 0.5 meter.
Table 2 lists the main properties of the used DTM of the test site. The height is ranging from 683.8 to 655.8 meters with a mean height value of 670.3 meters and a standard deviation of 4.9 meters.

Table 2. Basic information about the digital terrain model of the test site

Height	**Value [m]**
Maximum	683.8
Minimum	655.8
Mean	670.3
Standard deviation	4.9

2.5. Reference Data for Accuracy Assessment

To evaluate the results of the developed algorithm, validation data is needed. For this task, all skid trails were manually digitized using hillshades of the test area. Hillshades allow the operator to recognize the trails in the area, thus, it is possible to use them for validation.The differentiation between skid trails and natural rifts were not made during this process.
2,193.7 meters of line features were visually classified and digitized. Assuming that the width of a skid trails is in between four to six meters, a buffer of three meters to each side of the line feature was created and the resulting was area compared to the entire size of the test area. 20.86 percent of the test area consists of skid trails (including natural rifts and valleys). Figure 5 shows the digitized line features on the hillshade of this area.



(a) Aerial image and the digitized line features **(b)** Hillshade and the digitized line features

Figure 5. Digitized line features in the test area for validation

Edge detection is an important topic in computer vision. It is defined as the acquisition of edges in an image, not generated by noise, but by scene elements itself. Edge points, often abbreviated as edges, are pixels at which the values are characterized by a sharp variation. The higher the local intensity change at a certain position in the image, the higher are the chances, that an edge is located at this position [45]. Often the term edge is used to relate to chains of edge points which are connected to each other. These chains are called *contour fragments*. In literature, edge points are also described as *edgels*, which is a short form for edge elements.

A lot of different reasons exist for detecting edges in images. All elements from an image, like shadows, solid objects, or other marks on surfaces generate edges and thus, contours. Furthermore, these edges are used for object recognition and detection, motion analysis and other scientific topics. The process of edge detection includes three steps. Noise smoothing, edge enhancement and edge localization.

The first step, noise smoothing, aims at suppressing as much noise of the image as possible and not destroying features of the image and thus, edges. If there is no further information about the quality of noise available, it is considered *Gaussian*. Secondly, a filter is needed that detects edges. It has to detect homogeneous regions' boundaries, which are based on texture and intensity [46]. The pixel values of the output have to be low where no edges occur and high, for actual edges. This ensures the location of the edges (local maximum) in the filter's result. Finally, the edge localization has to determine, whether the chosen local maximum is a result caused by noise or are edges. This step involves a nonmaximum suppression, which thins wider contours to a one pixel width and afterwards performs a thresholding [47].

Three edge detection filters will be explained in the following chapter. These are, in appearing order, *Prewitt* and *Sobel* edge filters, *Canny* edge filter and finally, the *Laplacian of Gaussian (log)* filter.

To understand these filters, some basics have to be explained. It is necessary to have a certain knowledge about *partial derivatives*, *the gradient* of an image and *derivative filters*. A *partial*

derivative is the derivative of a multidimensional function. It can be taken along both of the image's coordinate axes u an v (equation 9).

$$\frac{\partial I}{\partial u}(u,v) \text{ and } \frac{\partial I}{\partial v}(u,v) \tag{9}$$

The *gradient vector* at the position (u,v) of function I can be computed with equation 10.

$$\nabla I(u,v) = \begin{bmatrix} \frac{\partial I}{\partial u}(u,v) \\ \frac{\partial I}{\partial v}(u,v) \end{bmatrix} \tag{10}$$

The magnitude $|\nabla I|$ (equation 11) of the image gradient (equation 10) is not effected by any rotation of the test image and therefore, it is not dependent on the direction of image structures. This is the main reason for being the underlaying basis of many different edge detection functions.

$$|\nabla I|(u,v) = \sqrt{(\frac{\partial I}{\partial u}(u,v))^2 + (\frac{\partial I}{\partial v}(u,v))^2}. \tag{11}$$

The components of equation 10 are colums along the vertical and horizontal axes and the first derivative, $f'(x) = \frac{df}{dx}(x)$, of the image. The coefficient matrix

$$H_x^D = [-0.5\ \mathbf{0}\ 0.5] \ = \ 0.5 \cdot \ [-1\ \mathbf{0}\ 1] \tag{12}$$

can be used to approximate the first horizontal derivative (equation 13)

$$\frac{df}{du}(u) \approx \frac{f(u+1) - f(u-1)}{(u+1) - (u-1)} = \frac{f(u+1) - f(u-1)}{2} \tag{13}$$

and to create a linear filter. The coefficients of equation 12 -0.5 and $+0.5$ are valid to $I(u-1,v)$ and $I(u+1,v)$. The center pixel $I(u,v)$ of image I is ignored due to its weighting with zero. The second, the vertical component can be calculated by the linear filter (equation 14) as well [45].

$$H_y^D = \begin{bmatrix} -0.5 \\ \mathbf{0} \\ 0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} -1 \\ \mathbf{0} \\ 1 \end{bmatrix} \tag{14}$$

Prewitt and Sobel edge filters are one of the most commonly used filters for contour detection and only slightly differ. Therefore, both filters are described in one chapter. Several different variations of the Sobel filter are available (see [48] or [49]), which decrease the sensitivity to noise or increase the overall quality. This chapter will explain the basic variant of the filter. Both of them use linear filters as described above. These filters sprawl over three contiguous columns and lines to intercept errors due to noise, which would result out of "simple" gradient operators (equation 13 and 14). The *Prewitt* method applies the filters,

$$H_x^P = \begin{bmatrix} -1 & 0 & 1 \\ -1 & \mathbf{0} & 1 \\ -1 & 0 & 1 \end{bmatrix} \text{ and } H_y^P = \begin{bmatrix} -1 & -1 & -1 \\ 0 & \mathbf{0} & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{15}$$

which computes an average gradient of three adjacent columns or lines. When the filters are broken down into its components,

$$H_x^P = \begin{bmatrix} 1 \\ \mathbf{1} \\ 1 \end{bmatrix} * \begin{bmatrix} -1 & \mathbf{0} & 1 \end{bmatrix} \text{ and } H_y^P = \begin{bmatrix} 1 & \mathbf{1} & 1 \end{bmatrix} * \begin{bmatrix} -1 \\ \mathbf{0} \\ 1 \end{bmatrix}, \tag{16}$$

H_x^P is performing a box smoothing before the computation of the gradient x. H_y^P is as well smoothing before computing the y gradient. The smoothing could be applied after the computation of the gradients as well due to the commutativity of linear convolution.
The *Sobel* method uses nearly the same filters. Only the smoothing values are higher.

$$H_x^S = \begin{bmatrix} -1 & 0 & 1 \\ -2 & \mathbf{0} & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ and } H_y^S = \begin{bmatrix} -1 & -2 & -1 \\ 0 & \mathbf{0} & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{17}$$

An appropriate scaling is afterwards used to estimate local gradient components

$$\nabla I(u,v) \approx \frac{1}{6} \cdot \begin{bmatrix} (I * H_x^P(u,v)) \\ (I * H_y^P(u,v)) \end{bmatrix} \tag{18}$$

for the *Prewitt* method and

$$\nabla I(u,v) \approx \frac{1}{8} \cdot \begin{bmatrix} (I * H_x^S(u,v)) \\ (I * H_y^S(u,v)) \end{bmatrix} \tag{19}$$

for the *Sobel* edge detection.
In a next step, the *strength* and *orientation* of the filter results have to be computed. Let

$$D_x = H_x * I \text{ and } D_y = H_y * I$$

be the scaled filter results. For both, D_x and D_y, the local edge strength $E(u,v)$ can be described as the gradient magnitude

$$E(u,v) = \sqrt{(D_x(u,v))^2 + (D_y(u,v))^2}. \tag{20}$$

The orientation of the resulting edge is defined as the angle $\Phi(u,v)$ and can be calculated with

$$\Phi(u,v) = \tan^{-1}\left(\frac{D_y(u,v)}{D_y(u,v)}\right) = \arctan\Big(D_y(u,v), D_x(u,v)\Big). \tag{21}$$

Often these two values are used to visualize the edges by coding the orientation angles as color hues and the color saturation is controlled by the edge strength. Due to the simple implementation and good results, these two filters are used commonly and are available in many software packages and image processing tools [45].
The results from edge detection filters and the edges, which humans identify, often differ. Two main reasons cause this problem. First, only local intensity differences are gathered by filters, while the

human visual system is capable of extending edges with vanishing contrast or with minimal contrast. Secondly, edges are defined over different scales and, are not fixed at a certain resolution or scale. Edge filters, like the *Sobel* filter, only respond to differences in the image intensity, which occur in their $3x3$ pixel filter region. In order to detect contours on a larger scale, either the image has to be scaled and the small filters are used, or larger edge detection methods are needed. If both of these techniques are combined, then they are called *multiresolution* filters. Various scales are used to detect edges and afterwards the dominant contours at each scale are selected [45].
Filters, which are based on the first derivative of an image I, most of the time produce edges which have the same width as the intensity transition of the underlaying feature. The second example for edge filters, which is based on the second derivative of the function, is called ***Canny edge*** [50] filter. A trade-off between edge localization and noise reduction is addressed by the filter. A reduction of noise entails an increasing error in the localization of edges, and vice versa [51].
The curvature of a function is described by its second derivative. The basic idea is that edges are located at zero positions, or in a perfect manner at the zero crossings of the image function's second derivative. The top part of figure 6 shows the function $f(x)$ of the image I. Subsequently, the first and second derivative $f'(x)$ and $f''(x)$ are outlined. The zero crossings of the second derivative can be seen as well in figure 6.

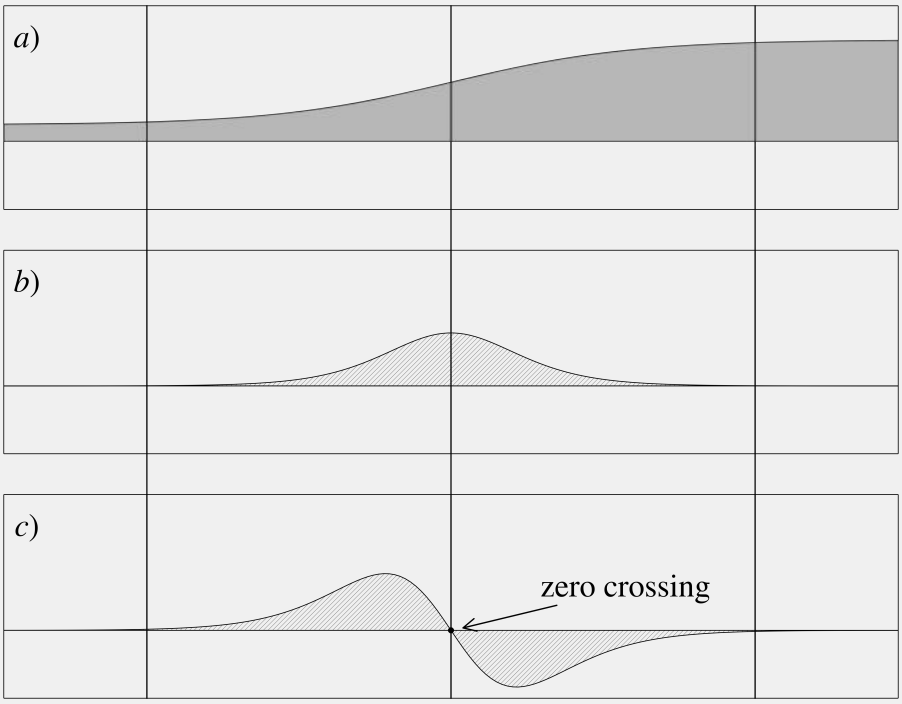


Figure 6. Function (a) and its first (b) and second derivative (c)

Due to the fact, that second derivatives have a tendency to amplify noise, it is necessary to apply a suitable low-pass filter to remove noise in a previous step. A *Gaussian low pass* filter, with the smoothing parameter $sigma$, which defines the filter width and the amount of smoothing [51], is used by the *Canny* filter. An example for such a technique is the *Laplacian-of-Gaussian (LoG)* [52]. This edge filter applies the combination of a *Gaussian* filter and the second derivative. The following

chapter will explain the *LoG* edge filter in more detail.

The *Canny* edge filter is an example for multiresolution filters. It does not only use one scale of the image for the detection of edges, but several image resolutions are used, and the result of each individual filter run is afterwards merged to a global edge map. Three main goals are attempted to achieve by the filter. First, the number of false edge points are tried to be minimized. Second, a good edge localization is aimed at and third, only one mark on each edge is delivered. The basic principle of the filter is a gradient method, which is based on the first derivative. For a precise location of edges, the second derivative of the image function is utilized. Nevertheless, the algorithm is often used as a single-scale implementation with the smoothing parameter σ. Even the single-scale version of the filter is superior to most edge detection filters [45].

An image can be envisioned as a surface with the height defined by the corresponding pixel gray levels. The first order derivative of this surface, calculated for example by the *Sobel* operator, measures the slope of the surface on both, x and y directions. But the second-order derivative an be computed as well. This can be used to localize edges, because the rate of the slope of the gray level surface change is the subject of the second derivative. Figure 6 shows such a visualization of the gray level as a surface and the first and second order derivative. Because the gray level change is not abrupt, a broad peak can be seen in the first derivative. The signs of the second order derivative provide information about the location of the edge, whether it is the dark or bright side of the edge. This feature can be used to sharpen the image as well by adding some percentage back to it, and is called *unsharp masking*. It increases the detail visibility by applying a blurred positive of the image to subtracting the local brightness average in a specific area [53]. Negative values occur on the bright side, whereas the dark side of the edge is defined by positive values. At the center of the edge, the sign changes. This position is called *zero crossing* and is used for edge detection. Most of the time, exact zero pixel values are not available, but two neighboring pixels with opposite signs. The pixel with the smaller, absolute pixel value, then is defined as the zero crossing [51]. The second order derivative of an image function f is combined in the Laplacian, and is shown in the following equation [54].

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}. \tag{22}$$

The kernel of an approximated $3x3$ neighborhood Laplacian can be written as

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & \mathbf{4} & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{23}$$

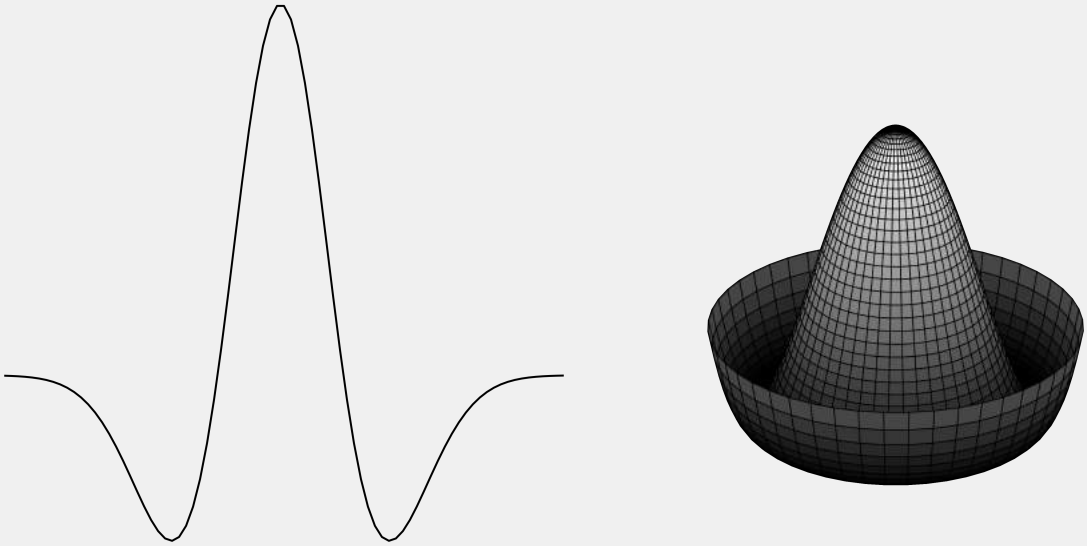
and is very similar to a high pass filter. Due to the fact, that the *Laplacian* is very sensitive to noise, most of the time it is not used for edge detection on its own. Therefore, the combination with a *Gaussian* filter is chosen. In a first step, the gray level surface is blurred by a *Gaussian* filter. Afterwards, the *Laplacian* enhances the edges, which are localized in a last step by the zero crossings of the second order derivative. A two dimensional radially symmetric *Gaussian* [51] is defined as

$$h(r) = \exp(\frac{r^2}{2\sigma^2}) \tag{24}$$

with the variable $r^2 = x^2 + y^2$. The *Laplacian of Gaussian* can be written as [46,51,52]

$$\nabla^2 h = (\frac{r^2 - \sigma^2}{\sigma^4}) \cdot \exp(\frac{-r^2}{2\sigma^2}). \tag{25}$$

It is very common to invert the the *Laplacian of Gaussian* function (equation 25) and replace the minimum in the center by a maximum. The resulting shape of the function is called *mexican hat* and can be seen in figure 7.



(a) Mexican hat: two dimensional visualization **(b)** Mexican hat: three dimensional visualization

Figure 7. Two and three dimensional representation of the Mexican hat

The variable σ defines the width of the *Gaussian* filter and therefore, specifies the amount of smoothing. σ also enables a detection of edges at different scales. Even with a small amount of smoothing chosen, the filter is very computationally expensive. An approximation is the subtraction of two *Gaussians* with different widths, and is called *difference of Gaussians (DoG)*.
Most of the edge detection filters, which have been described above, return the angle of the edge and an edge strength value. Two methods are going to be described briefly, which allow an identification of contours and larger image objects. The first method is the *contour following*. In principle, it is quite a simple idea to follow contours along the detected edge points. A pixel with a high edge strength is chosen and the edge is followed in both directions until the two directions meet each other and thus, a closed structure is created. For all practical purposes its a very difficult task with three major problems. Edges are not perfectly featured and perhaps end in a vanishing manner. Secondly, several edges can intersect each other and therefore, they can be ambiguous for the algorithm. Lastly, edges can split up into several branches, which are leading into several directions. Due to these setbacks, this technique is only applied to very simple circumstances, like binary images with a clearly defined background, but not to continuously valued edge or original images [45].
Edge maps are carried out to select points, whether they are edge points or normal image pixels. The

easiest technique, a binary method, is using a threshold procedure. By applying an adaptive or even a fixed threshold value to the edge strength values (returned by the filter), edge maps or binary edge images, are obtained. Perfect contours are hardly ever received, but interrupted or unconnected edge fragments are the reality due to not enough edge strength. Empty positions in the edges do not contain any edge information anymore after thresholding and can therefore not be used in a further step, such as linking matching edges. Because of its simplicity, thresholding is often employed despite the occuring problems. Some of the missing links can be closed by postprocessing techniques, like a Hough transformation [45].

After the explanation of the four different edge detection methods, the results will be shown. Figure 8 shows the test image. It was chosen due to the complexity of the edges on different levels and the contrast variation. Lastly it does not only contain the desired feature, the boat, but also contains noise, like the foreground or the clouds in the background. The described methods were applied to the hillshade of the test area, but did not lead to any results.

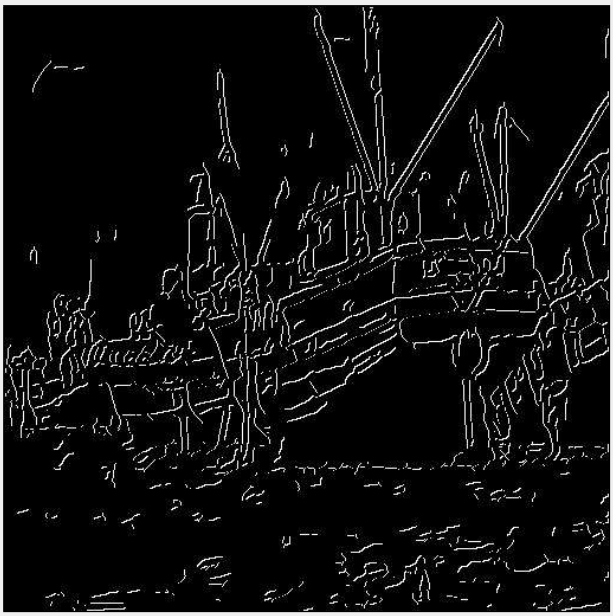
Figure 8. Chosen test image for the different edge detection methods

Figure 9 shows the four results of the edge detection methods. All four techniques recognized the majority of the boat. As mentioned above, the test image 8 did not only contain the feature, but noise in the background and foreground as well. The background, which contains the sky with clouds did not effect the results of either one of the edge filters. All four of them did not detect edges of the clouds. The noise in the foreground, the sea floor, created various problems for three of the them. Only the first one, the *Prewitt* method, see figure 9a, did not register any structures of it. The other three, *Sobel*, *Canny* and *Laplacian of Gaussian*, had problems with the noisy part of the test image. Figure 9c handled it the best in comparison to the other two. The noise is only partially existing in the lower right side of the image. *Laplacian of Gaussian*, which used the kernel size of 2.5 and $\sigma = 1.5$, not only detected most parts of the ship, but a lot of the noisy sea floor. Lastly, the *Sobel* edge detection, figure 9b, detected more of the thinner edges, for example at the top right part and the

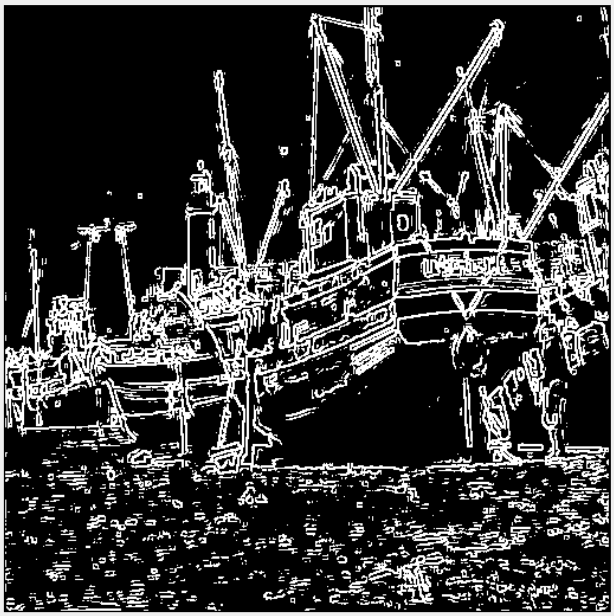
discerned boat contains more details. This method had the same problems with the noisy foreground as the other two edge filters.

(a) Result of the *Prewitt* method

(b) Result of the *Sobel* method



(c) Result of the *Canny* method



(d) Result of the *Laplacian of Gaussian* method

Figure 9. Results of the four different edge detection methods of the chosen test image 8

2.6. Distance Regularized Level Set Evolution, DRLSE

This work uses the framework of *active contour models* as introduced and described in [55].

A program was developed, which uses the underlaying code from [56]. It is available at github.com/sTAKKLE5/masterthesis.
Further information about the basics of level sets, level set evolutions, geodesic active contours and the *DRLSE* method is provided by [23,24,26,28–30,32,55].

Active contour models are not only used in computer vision and computer graphics, in but the fields of medical informatics and bioinformatics as well [31,33–35]. Anatomical structures, appearing in computed tomography (CT) or magnetic resolution (MR) scans, are most of the time segmented or extracted and used for simulations, therapy evaluation, diagnosis and even for the surgical planning [34]. Furthermore, images derived by x-ray, ultrasound and angiography techniques are widely utilized. These techniques are not only applied for segmentation, but for tracking objects, like growing of neurites or even the motion of the heart [35]. Figure 10 shows the detected contour of an magnetic resolution image by the *DRLSE* method.

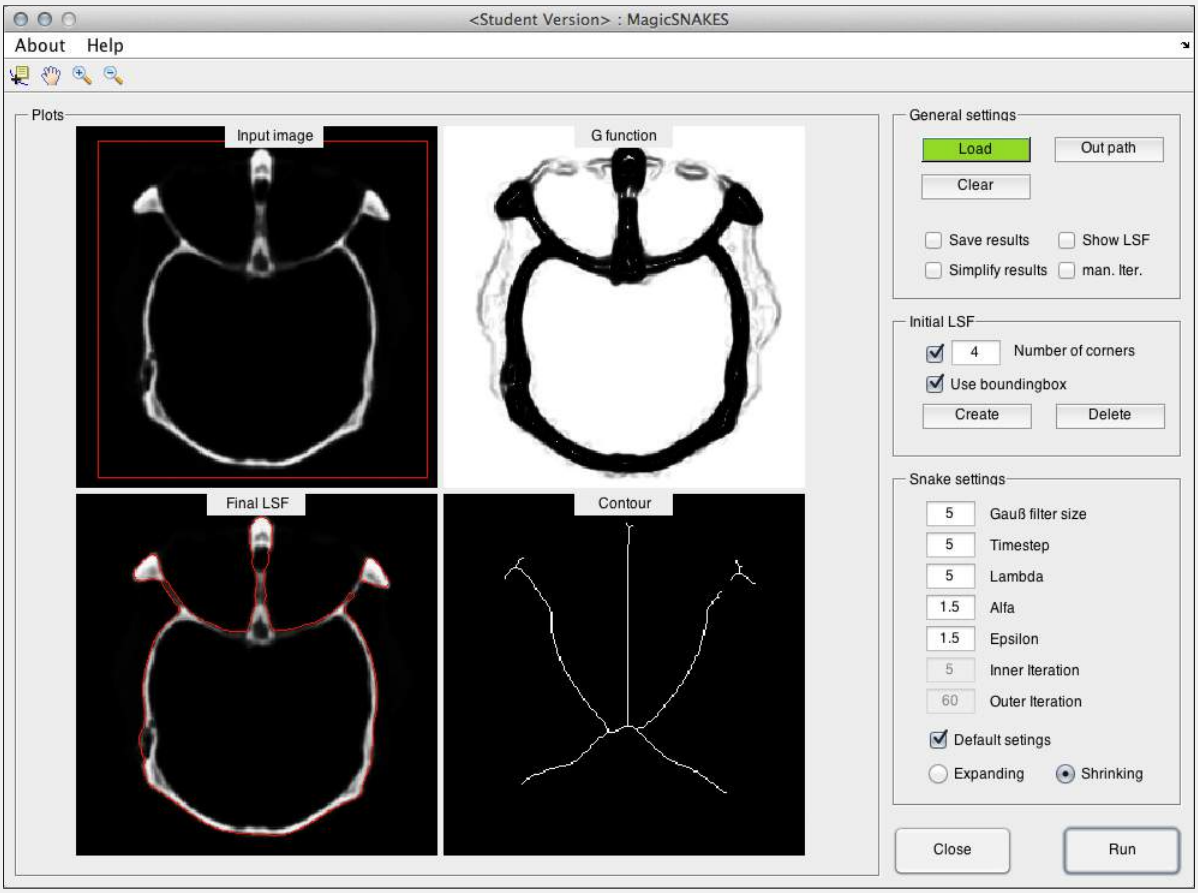


Figure 10. Extraction of the contours using *DRLSE*

The first *level set method* for shape detection was published by Osher and Sethian [32] in 1987. The general idea of level sets is to depict the contour of a desired feature as the zero level set of a function with a high dimension, which is called a *level set function (LSF)*. The evolution of this level set function represents the motion of the contour. Caselles, *et al.* [57] introduced the first level

set method, *active contour* or *snake models*, in computer vision and image processing. These early models, curve evolutions, can be be described as

$$\frac{\partial C(s,t)}{\partial t} = FN \tag{26}$$

with $C(s,t) : [0,1] \times [0,\infty] \rightarrow \mathbf{R}^2$, where s ranges from zero to one and is called spatial parameter. The spatial parameter is used for parameterization of the points laying on the contour. The temporal parameter t is defined as $t \in [o,\infty]$. One of the most important parts of the equation, its motion, which is controlled by the speed function F. Furthermore, the inward normal vector N of the curve C has to be calculated. The curve evolution, equation 26 is converted into a a level set function by integrating a LSF $\phi(x,y,t)$, is dependent on the time t. The inward normal vector N, for positive values inside the zero level contour, and negative for the outside of the embedding LSF ϕ, can be written as $N = -\nabla\phi/|\nabla\phi|$, with the gradient operator defined as ∇. Equation 26 can then be reformulated as

$$\frac{\partial \phi}{\partial t} = F|\nabla\phi| \tag{27}$$

It is called a *partial differential equation (PDE)* and is defined as a level set evolution equation.
An advantage to parametric active contour models is the ability to cope with merging and splitting, or any other topological change of the contour. Furthermore, the points on the contour do not have to be parameterized due to the computation on a predefined cartesian grid. During the evolution of the level set function, often a lot of irregularities occur. These irregularities are created during the evolution and can lead to numerical errors and thus, even can destruct the stability of the level set evolution. In order to prevent these numerical issues, *reinitialization* is used to re-establish the stability of the LSF and to obtain a constant level set evolution, too. The *reinitialization* method is computed by pausing the evolution and the demoted level set function is reshaped by a signed distance function. The following equation shows this *reinitialization* process.

$$\frac{\partial \psi}{\partial t} = sign(\phi)(1 - |\nabla\psi|) \tag{28}$$

ψ is the level set function, which has to be reinitialized and the sign function is represented by $sign(\cdot)$. The perfect solution for equation 28 is a steady state. A problem, which concerns the *reinitialization* method, is that the zero level set could lead to an incorrect position of the final result and thus, this method should carefully be used. *Reinitialization* causes some practical, and also even theoretical problems during the application. Due to these issues, see [58–60], level set methods are used, which do not rely on *reinitialization*.
One method, which does not rely on *reinitialization* are *geodesic active contour, GAC* models. Further information about these models are provided by [29]. This technique is not using iterations and an evolution of the underlaying equation, but the LSF is updated at each step by calculating an optimization issue.
However, this paper applies a level set with a more general formulation with an external energy, that moves the zero level contour towards the desired features and a distance regularization term. It contains a potential function with the outcome, that the shape of the resulting level set function is

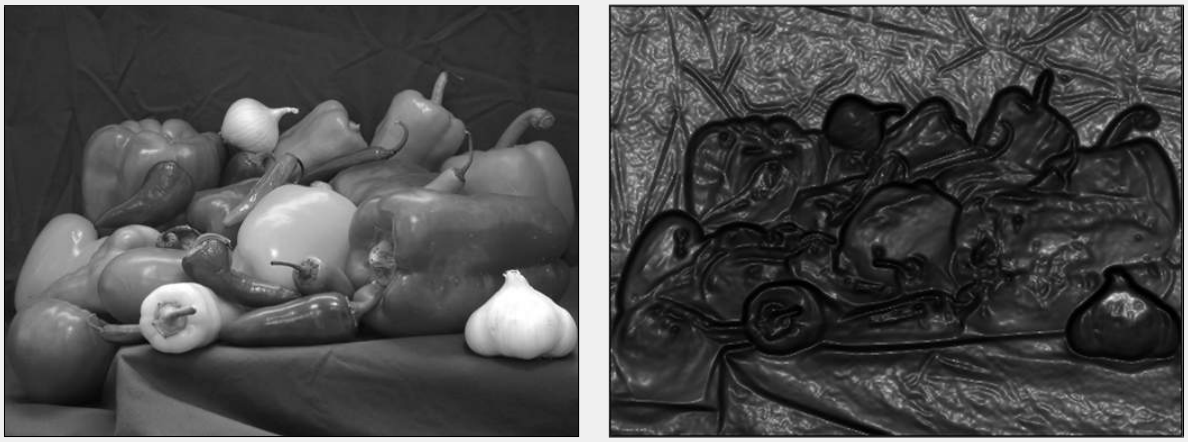
obtained. A double well potential function is used for this part. The gradient flow with a minimizing energy function is used to derive the level set evolution. Furthermore, the distance regularization term sustains the level set function by forward-and-background diffusion. Therefore, *reinitialization* is not needed and negative issues concerning it, do not occur. This distance regularization term is eponymous for the method, which is called *distance regularized level set evolution*, *DRLSE*. This technique can be applied in a more efficient way in image processing, due to the possibility to utilize larger time steps, which reduces the absolute number of iterations and therefore, the computation time of the method. Besides, the numerical accuracy is obtained as well [55]. Equation 29 shows the *DRLSE* method.

$$\frac{\partial \phi}{\partial t} = \mu div(d_p(|\nabla \phi|)\nabla \phi) + \lambda \delta_\epsilon(\phi) div\big(g\frac{\nabla \phi}{|\nabla \phi|}\big) + \alpha g \lambda \delta_\epsilon(\phi) \tag{29}$$

with g as the edge indicator function, see equation 30, $\lambda > 0$, $\alpha \in \mathbf{R}$, δ the Dirac delta function and ϕ the level set function ($\phi : \Omega \rightarrow \mathbf{R}$).

$$g \triangleq \frac{1}{1 + |\nabla(G_\sigma * I)|^2} \tag{30}$$

with the image I and G_σ as a *Gaussian* kernel and the standard deviation σ. The edge indicator function g is used for smoothing the image and to reduce the noise. Usually it takes smaller values at the boundaries of an object than at any other locations. Furthermore, it slows down the the expansion or shrinking when the zero level contour reaches the smaller values at the desired boundary of the feature. The variable α defines, whether the zero level contour expands, for negative values, or shrinks, for positive values [55]. Figure 11b shows the edge indicator function of figure 11a.



(a) Test image **(b)** Edge indicator function using $\sigma = 1.5$

Figure 11. Test image and it's edge indicator function

The last and final step is the *skeletonization* of the resulting feature shape. For this specific case, the desired result is not the bounding box of the object, but the centerline. To achieve this task, several methods of the skeletonization can be utilized. Many different algorithms, such as *curve evolution*, *morphological operators*, *curve evolution* or *thinning* are utilized. This paper applies the

thinning method to the obtained binary image.
The *hit-or-miss transformation*, equation 31, describes the thinning of a set A by a structural element B, which can be formulated as $A \otimes B$.

$$A \otimes B = A - (A \odot B) = A \cap (A \odot B)^c \tag{31}$$

Structuring elements can be defined as the specific details of the impact of an filter method on the image. Sometimes it is also called *kernel* and consists out of a pattern, which is specified by the coordinates of a number of discrete points relatively laying to an origin. This origin does not have to be the center of the shape, but most of the time is.
As we are dealing with a binary image, only two values, one and zero occur, and therefore, no further background operation is needed to perform the hit-or-miss transformation. The sequence of structural elements can is applied as a more practical expression

$$\{B\} = \{B^1, B^2, B^3, ..., B^n\} \tag{32}$$

with B^i equals the rotated version of B^{i-1}. This concept is now used to determine thinning by structural elements

$$A \otimes B \triangleq ((...((A \otimes B^1) \otimes B^2)...) \otimes B^n) \tag{33}$$

The origin of the structural elements are translated to every possible pixel location in the image. At each of those positions, it is compared to to the underlaying background image pixels. If they are exactly the same (foreground and background pixels), then the pixel of the image laying under the origin of a structuring element is defined as zero and determined as background. If both of them do not perfectly patch, the pixel value is not changed. To have any effect, the structuring element has to have a pixel value of either zero or one. These steps describe one step of the algorithm. In order to skeletonize all of the image, this step is repeated until no further changes occur [54].
To test the *DRLSE* algorithm, underlaying code from [56], a user interface was created to simplify the creation of the initial level set and the general parameters. The program is written in *Matlab* and is available at github.com/sTAKKLE5/DRLSE.
Figure 12 and 10 show the graphical user interface of the program and the results. They explain the features of the program.

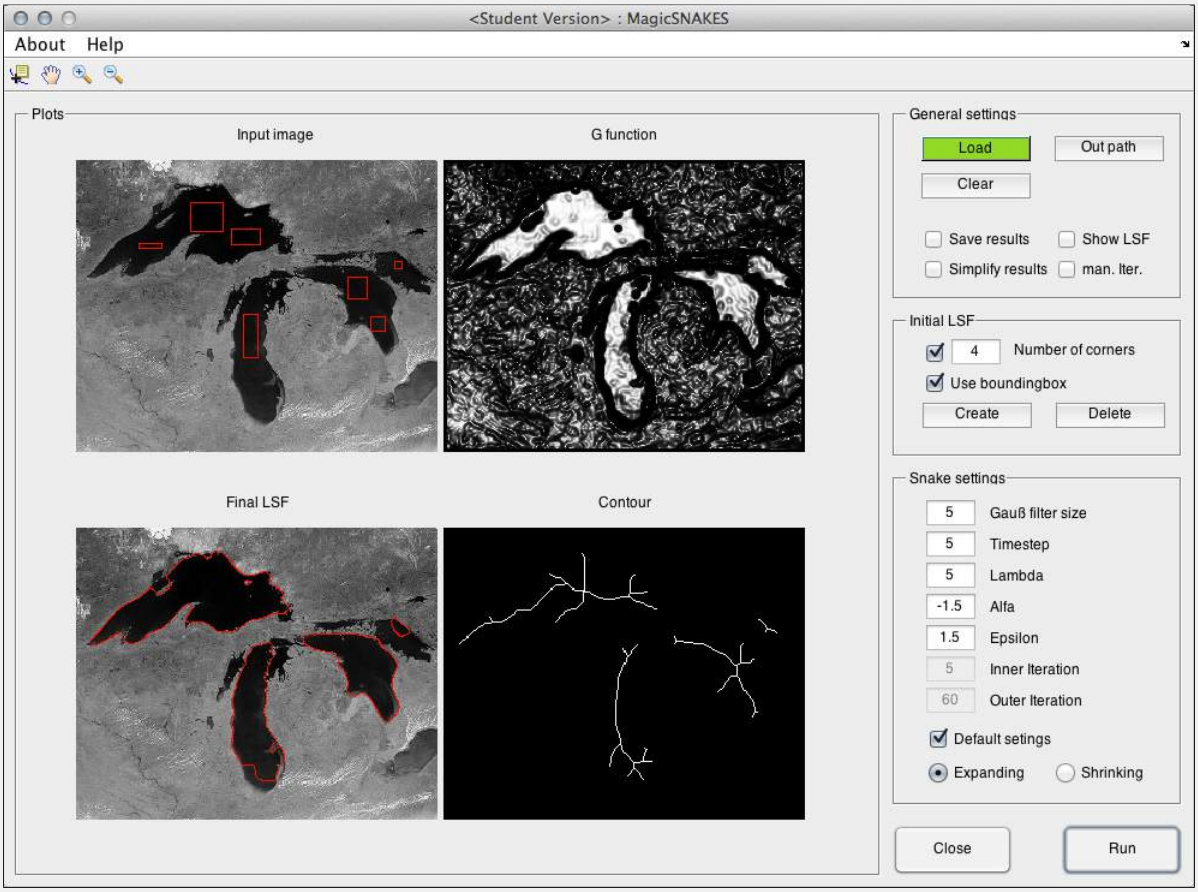


Figure 12. Extraction of the contours of the great lakes in the United States using the *DRLSE* method

The program is divided into four parts. The *Plots*, which constitutes the most part of the program, the *general settings* in the top right corner, the *Initial LSF* section and the settings for the active contour model called *Snake settings*. The major part is taken up by the four plots. On the top left side, the chosen image is shown. It is used by the user to draw the initial level set on it. In this specific case, seven level set functions were drawn in order to extract the contours. On the top right side, the result of the edge indicator function, using equation 30, is shown. After the iterations, the final level set function is displayed on the bottom left part of the *Plots* section. At last, the result of the skeletonization method, using equation 31, is displayed in the bottom right figure.
Secondly, the *General settings* section is applied for the basic settings of the program like defining the input and output image. Furthermore, the user can choose whether the results should be saved and in a previous step simplified. The two other options are to show the three dimensional visualization of the zero level contour during the calculation process, option *Show LSF*, and whether the user wants to predifine the amount of iterations or if he wants to manually stop the process.
In the next step, the user has to define the initial level set function. For this task, he/she can choose the number of corners for it whether to use the bounding box or not. Not only one function can be created, but as many as necessary.
Fourth and lastly, the settings for the *DRLSE* method, have to be defined. In this case, due to the

homogeneous texture inside the lakes, the expanding version has been chosen (negative alpha value). The number of iterations are set to infinite and the user has to stop the process manually when the final level set function matches the desired contours.

*2.7. Development of the Moving Window Standard Deviation Maximum Distance (*MWSD*) Filter*

Skid trails can be determined by using their most characteristic feature, which separates them from the surrounding forest floor, the height difference. This height difference is used as the underlaying basic information of the *MWSD* filter. In order to generally apply this technique, no absolute values, but the standard deviation, is used. A certain range has to be specified, e.g. $[1 * standard\ deviation; 1.5 * standard\ deviation]$, by the users. Not only the height difference range needs to be given, but a second range, too, which is used in order to check if the distance of the currently evaluated pixel combination lays in this certain range. It is defined as $[distance_{min}; distance_{max}]$, the minimum and the maximum distance.

The underlaying basic principle, for the detection is to check for significant, not natural height differences, between the pixels P_N and P_{N+1}. It is aimed at a perpendicular intersection of the trails by laying out artificial lines $\overline{P_N P_{N+1}}$, which are described by all unique combinations of pixels.

In a first step, the input digital terrain model is split into smaller parts. The size of these parts is defined by the user, and is dependent on the features to track. Afterwards, the standard deviation σ of the height for each tile is calculated and stored. In a subsequent step, all unique combinations for each pair of pixels in the tile is determined, and their coordinates are saved for later computations. Then, the absolute height difference for each line combination $\overline{P_N P_{N+1}}$ is calculated. If this difference is in the user defined range, $[standard\ deviation_{min}; standard\ deviation_{max}]$, the length of the combination is calculated as well. In case, the length is in the given range, $[distance_{min}; distance_{max}]$, the start and end point of this particular combination is saved. There is a significant height difference between these two points and it is assumed, that this specific line intersects a skid trail. In order to calculate the center point of the trail, a point sampling along the line is carried out to get the point with the smallest height value. The point sampling can be achieved in different ways. Either a fixed amount of points on the line, or points every specific unit can be used. If the height difference is not laying in the given range, or the distance is either too larger or too small, the next pixel combination of the window is analyzed. As a result of the *MWSD* filter, a binary raster with the pixel value one for detected skid trails and zero values for not significant pixels, is created.

This technique is applied on all points in the window and for all windows in the original raster. In a last step, the coordinates of the calculated minimum on the line combinations are saved, and the result can be used in further proceedings. Figure 13 shows the algorithm as a flowchart with the required inputs, the procedure and the resulting outcomes of the *MWSD* filter. Figure 14 visualizes the different steps. It shows a sample from a larger image and every pixel of it. A hypothetical skid trail, with lower height values than the surrounding area, is displayed in the lower right part of the image in gray with black borders. For the first step, figure 14a, only six combinations for two different pixels, N and O have been chosen. For each of the two pixels, N and O, the lines $\overline{NP_1}$ to $\overline{NP_6}$ and $\overline{OP_1}$ to $\overline{OP_6}$ have been

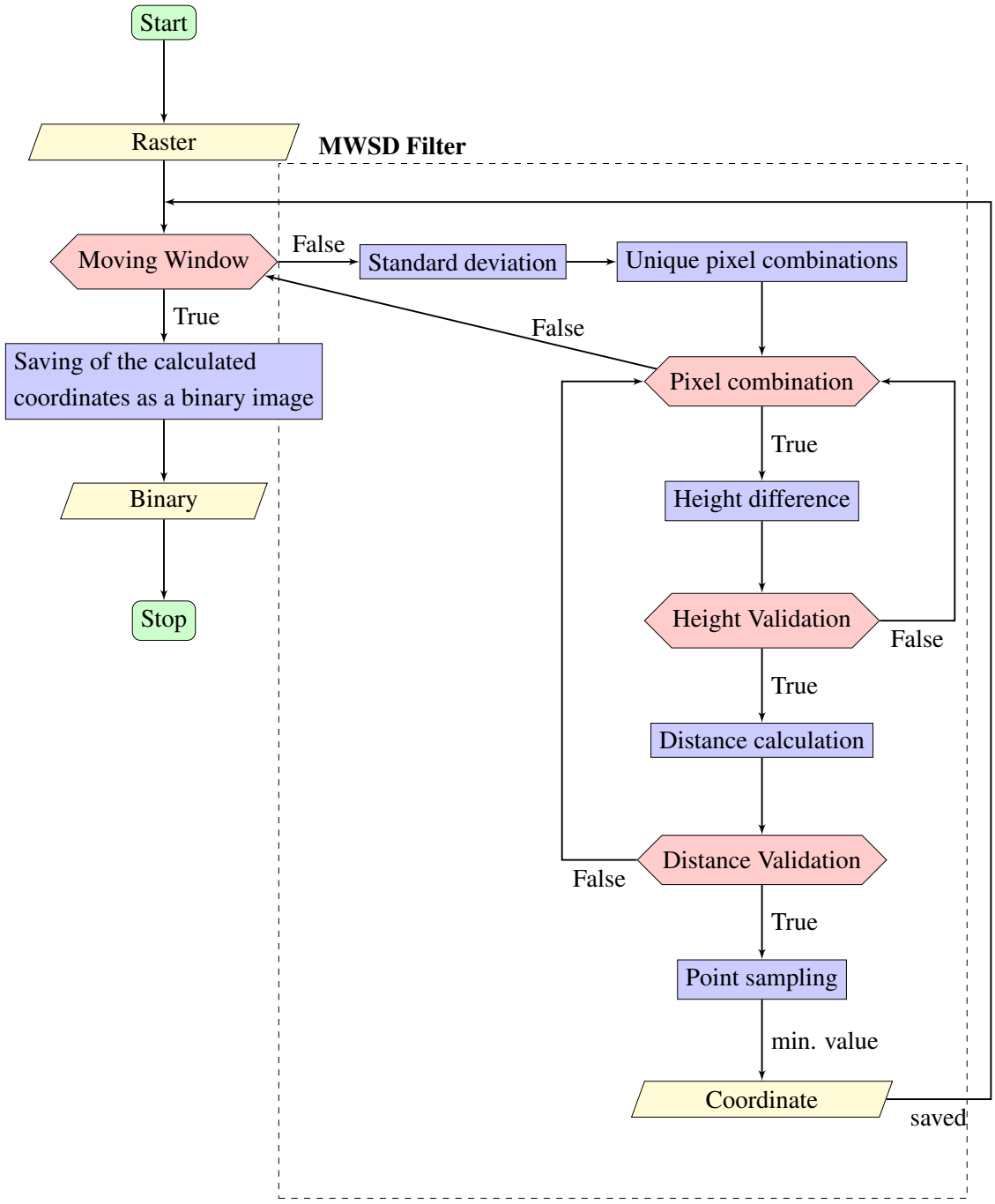


Figure 13. Flowchart of the *MWSD* algorithm

created. For the second step, it is assumed, that the height difference of the start and end points of $\overline{NP_1}$, $\overline{NP_5}$, $\overline{NP_6}$, $\overline{OP_1}$, $\overline{OP_4}$, $\overline{OP_5}$ and $\overline{OP_6}$ are not significant higher and they are displayed as dotted lines in figure 14b. Therefore, the algorithm does not consider these lines anymore, figure 14c, and continues with the next step and the remaining lines $\overline{NP_2}$, $\overline{NP_3}$, $\overline{NP_4}$, $\overline{OP_2}$ and $\overline{OP_3}$. In this example, five points on each line, displayed in figure 14d, have been chosen to sample the underlaying height information of the terrain model. Lastly, the points with the lowest height value have been chosen and named as R_1 to R_5, see figure 14e.

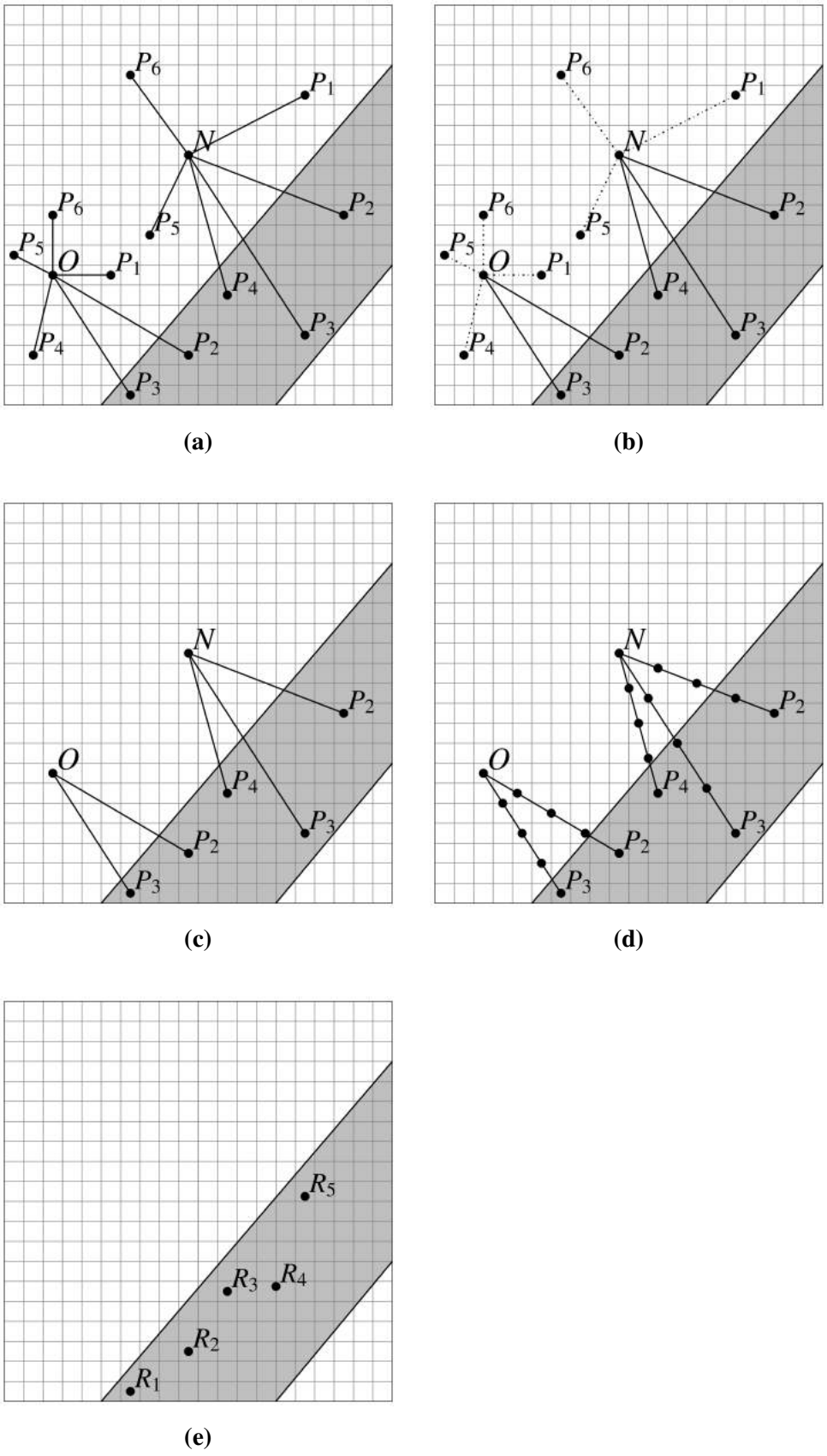


Figure 14. Visualization of the *MWSD* filter for a sample of pixel combinations for two pixels, N and O

Several methods can be used in order to **improve and speed up Python code**. These methods are necessary in order boost the running time of the moving window filter. The first, and most important part of improving the source code, is the identification of the bottleneck of the code, which can be described as the limiting factor of the program in terms of performance. The bottleneck of the filter is the creation of the pixel combinations and the calculation of the distance and height differences. Increasing the performance of a program by adding further lines codes, does not only speed it up, but sometimes can decrease the readability of the structure of the code, as well.
The following methods will include some of the key features of a programing language, local and external variables, loops and external modules.
Three types of variables are available in *Python*. Local namespaces, which are only available and specified in a class method or a function, global namespaces, which are available in the current module and built-in namespaces, which are globally defined in all modules. This is the order in which *Python* will search for specific variables. If for example x is defined as a global variable, *Python* will first verify whether x is local or not. As a result, *Python* will recognize, that the given variable x is not local and continues with global namespaces and will hit the local property of the variable. If the defined variable is classified as built-in, *Python* would continue to look it up. Because *Python* starts with local namespaces, these types of variables are found first and therefore, decreases the run time of an algorithm. External variables can be used to increase the efficiency, too, by storing and precomputing computational expensive calculations. This technique includes saving the results of computations locally into a file. In this specific case, the method was applied to save the combination of pixels and the according height difference while changing the variables of the algorithm e.g. standard deviation or maximum distance. Thus, the crucial parts of the algorithm only had to be calculated once during the first run.
One of the most common functions in a programing language are loops. *Python* supports more than one looping construction, but *for* statements are applied most frequently. This kind of loop can be replaced by the built-in method *map* [61], which forwards the loop into *C* code.
Besides the mentioned methods, the used modules play an important role. A lot of modules provide similar, or even the same methods and functions but differ in the speed. Some of them provide their code in two different languages e.g. pickle [62] and cPickle [63]. Both are utilized to turn *Python* objects into series of bytes, which can then be stored efficiently. The difference between the two of them is the language of the implementation. *pickle* is written in *Python*, whereas *cPickle* is implemented in the language *C*. Storing variables can benefit from this concept as well. As an example therefore, *lists* as a data structure can be named. The run time of the algorithm can be decreased by using *array* structures from the *numpy* module [64]. *Numpy* is using *C* in the background and therefore it improves the speed of the code. This module can as well be used for basic calculations like square roots [65].
Moving windows play an important role in data processing and data analysis. This technique does not use the full extent of data, but a small part of it. It can be utilized for processing data, which is too large to compute fully, or like in this specific case, to get statistics on a local scale.
In a first step, the user has to define the width and height of the moving window. Furthermore, the

overlapping area A, see figure 15, needs to be classified. It is used to avoid possible false structures at the edges of the window. The height and width do not have to be equal and can differ. Same goes with the vertical and horizontal overlapping area A. The origin of the first window size is the origin of the input image. After the calculation of the first window, it moves on line by line. The origin of the second window can be calculated by the origin of the input image plus the $width$ of the window minus the offset A, for the x-coordinate. In the first row, the y-coordinate does not have to be changed. If the moving window changes the row, the y-coordinate has to be changed, in regard to the origin of the input image, the height of the window size and the size of the overlapping area A. If the filter has reached the bottom right coordinate of the input image after N times, the filter stops. Due to the chosen width, height and the size of A, the size of the moving window has to be adjusted and recalculated at some specific positions. These are the right and the bottom boundary of the image. Figure 15 shows the addressed moving window and its features.

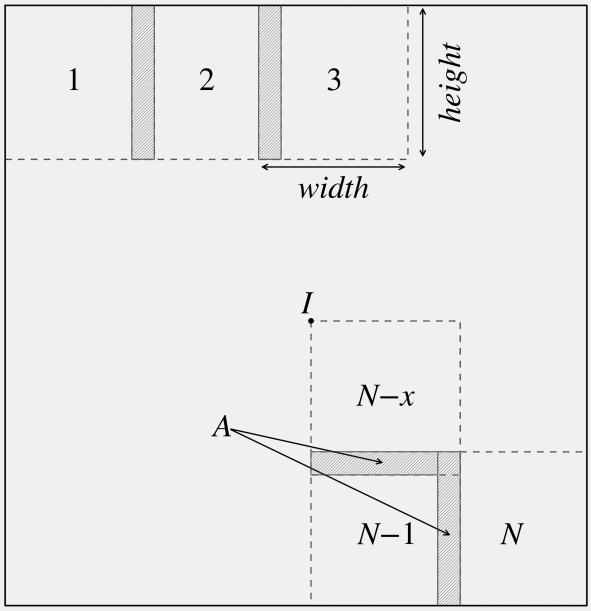


Figure 15. Visual representation of a moving window

Modern computers are equipped with more than one processor. These multi-core processors are capable of computing several tasks at once. This feature can be used to dramatically increase the performance of an algorithm. The so called **multithreading** is the process of spawning multiple threads, which exists in the context of one process. These threads are executing independently, but share the same resources.

In order to use all available cores of the operating machine, the algorithm hast to be split in several tasks. These threads have to be independent from each other. In this specific case, the input image was split into four parts, considered unique and calculated seperately on different processor cores. The most important reason for choosing multithreading is the amount of time, which can be saved. If only one processor is used for the computation, all other available cores are idle. For example, the computer should have four cores the computation time can be reduced by theoretically almost

75 percent. This value will not be reached, one core is already loaded up by tasks of the operating system and of other running programs [66].

*2.8. Development of the Maximum Angle Maximum Distance Vectorization (*MADV*) Method*

A method is needed, which transforms the detected single pixel values from the *MWSD* filter binary into a usable result. Due to the missing of further information, a fully automatic technique seems to be impossible. Therefore, a semiautomatic method has been developed, which is dependent on some location specific input variables and predefined start lines of skid trails.
In order to receive suitable results, some information has to be given to the algorithm. These are the maximum angle, in which the algorithm will search for new nodes of the line feature and the maximum distance between two potential line segments. Furthermore, a road network of forest roads is utilized to validate the resulting features. The most important input feature is the predefined segments of skid trails. They are used as the starting point for the algorithm and have to be defined for every skid trail. Lastly, the binary raster, which is the result from the *MWSD* filter and contains possible nodes, needs to be passed to the algorithm.
In a first step the algorithm calculates the angle for each given segment of a skid trail. To calculate this angle, a second line is needed. For this task, the upper boundary of the image is chosen with its upper left and right coordinate. The given segments consist of two points, P_1 and P_2 the start and end point of the line, thus the following method has to be applied to both sides of the given line. In a next step, raster points are searched within a certain distance and a determined angle. To find these pixels, a circle segment with the origin of either one of the two start points and the user defined maximum distance as the radius of the segment, which has to be perpendicular to the line segment, is calculated. For this task, the calculated angle of the line segment is used. It is needed to compute the angle of the circle segment for each side of the given line. Afterwards, each pixel value with a value equal one is saved for further calculation. The combination for each stored pixel value and the starting point is computed. Subsequently, the length and their angle is calculated, and used to determine the combination with the smallest difference regarding the angle of the given start line. This new skid trail segment, $\overline{P_{1/2}N_1}$ is saved and checked if it is intersecting the road network. If so, the intersection point will be stored and the algorithm will stop. If not, the ending point of the calculated segment $\overline{P_{1/2}N_1}$ will be utilized as the new start point and the algorithm will start again. If all of the given start lines have been run through the algorithm, the results are saved as a shapefile and it can be used for further purposes. Figure 16 shows a graphical visualization of the described *MADV* algorithm. The flowchart of the *MADV* algorithm with the required inputs and the resulting output is displayed in figure 17.

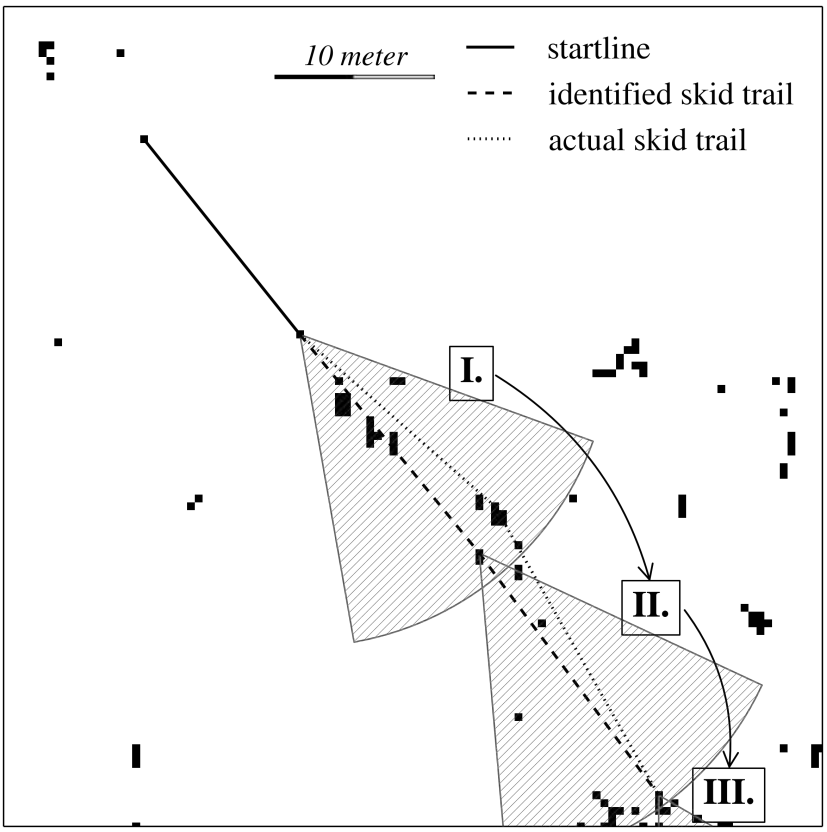


Figure 16. Graphical visualization of the *MADV* method

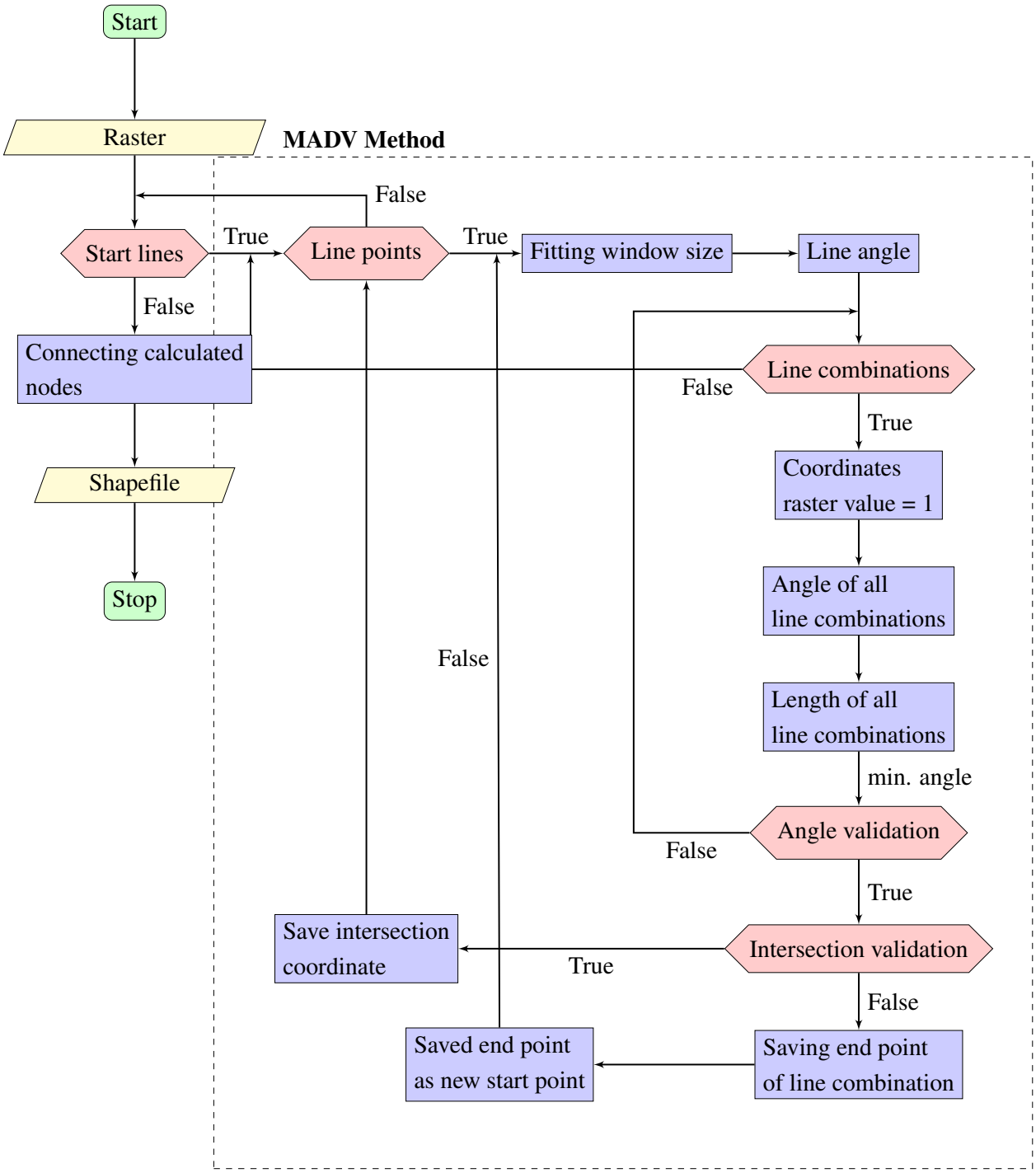


Figure 17. Flowchart of the *MADV* algorithm

Due to the fact, that the *MADV* algorithm uses high-intensity computing, like the calculation of the angle of each possible line combination and the length of each line, the input binary raster is split up into several smaller parts and analyzed separately. The **size of the window** plays an important part in terms of processing speed. If it is chosen too large, the amount of unsuitable line combinations, according to maximum distance and the angle difference, rises and therefore, unnecessary computations are carried out.

In order to calculate the perfect window size, some information is needed. First, the angle α of

the line $\overline{P_1P_2}$, for which the window size should be computed, has to be calculated. As already mentioned above, the orientation of the line is crucial for the angle. If line $\overline{P_1P_2}$ has an angle $\alpha = 40$, the line $\overline{P_2P_1}$, which is defined by the same points P_1 and P_2 as line $\overline{P_1P_2}$, does not have the same angle α, but an angle of $\alpha = 180 - \alpha$. In order not to get confused by the orientation of the lines, a sorting algorithm is used in a first step to "normalize" the line direction. Finally, the angle of the lines can be computed by equation 35. The second information, which is needed to compute the window size, is the maximum distance d between two pixels. It has to be defined by the user and is dependent on the results of the *MWSD* filter. If d is chosen too large, a misclassification can happen by connecting pixels, which do not represent the same part of the skid trail, or even are classified as noise and thus, define no skid trail at all. On the other hand, if the maximum distance is chosen too small, adjacent pixels may not be connected due to a too large distance. The last needed information is the search angle β. It defines the segment in which the algorithm is searching matching binary raster pixels. Like the maximum distance d, angle β is given by the user and depends on the input raster and therefore, on the spatial conditions of the underlaying test area. If the search angle is chosen too narrow, curves of the skid trails stop the algorithm and therefore, only some parts of the trails are vectorized. However, if it is chosen too broadly, then it is very likely that misclassification will occur. In this case, more pixels are laying in the segment and wrong links could be chosen by the algorithm. After the information is acquired, the basic workflow is:

1. Define a circle C with radius = d and origin = P_1/P_2.

2. Compute points I_1 and I_2 on circle c with the angle β_1 and β_2.

3. Calculate the minimum and maximum circle expansion in x and y direction; X_{max}, X_{min} and Y_{max}, Y_{min}.

4. Define the bounding box B of P_1/P_2, C_1 , $C2$, X_{max}, X_{min}, Y_{min} and Y_{max}.

5. Compute the length and height of the bounding box B.

In the first, step circle C is created by the equation of a circle $r^2 = (x-h)^2 + (y-k)^2$ with h and k equal the x and y coordinate of the center and the radius r. The equation with the actual values can be formulated as $d^2 = (x - P_{1x})^2 + (y - P_{1y})^2$. The next step is the computation of points I_1 and I_2, which are laying in the circle. In order to compute these, it is important to understand the basic angle distribution on a circle. Figure 18 shows this distribution.

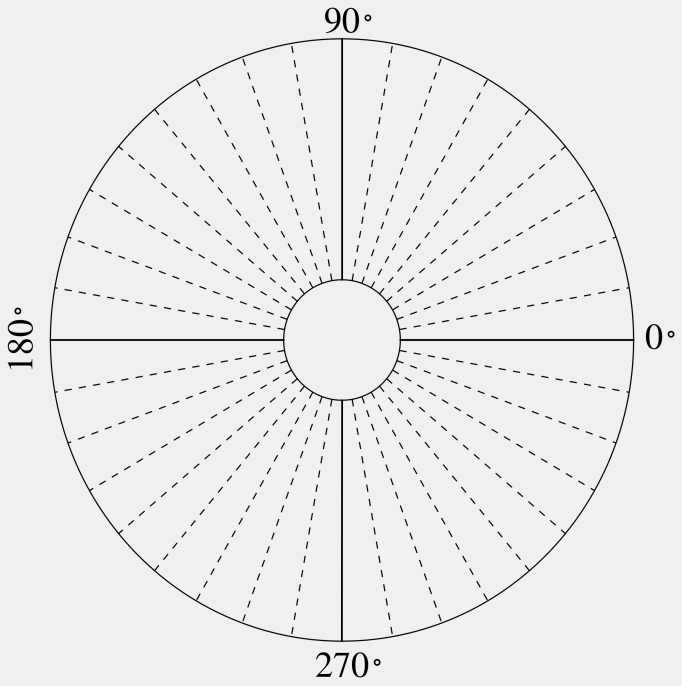


Figure 18. Visual representation of the angle distribution of a circle

As an example, a search angle $\beta = 30°$ and the angle $\alpha = 45°$ of line $\overline{P_1P_2}$ against a horizontal line e.g. upper or lower boundary of the image, is given. In order to calculate the according angles on the circle for the search radius, the angle α has to be converted into "circle angles", which is in this case $\alpha_{circle} = 225°$ for point P_1 and $315°$ for point P_2 (see figure 19). This angle can now be used to determine the angles $\beta_{1circle}$ and $\beta_{2circle}$, which differ of the search angle β (in this case $\beta = 30°$). Thus, the angle to the right and left hand side has to be symmetrical, each angle $\beta_{1circle}$ and $\beta_{2circle}$ has to be the half of the given search angle β and the sum of the angle α_{circle}. For this example, the computed values for angle $\beta_{1circle}$ and $\beta_{2circle}$, for P_1 of line $\overline{P_1P_2}$ is $135° - 15° = 120°$ and $135° + 15° = 150°$ ($45° + 15° = 60°$ and $45° - 15° = 30°$ for P_2). The coordinates of the points I_1 and I_2 were calculated with the following equation:

$$x = x_{origin} + r \cdot \cos(\beta)\ y = y_{origin} + r \cdot \sin(\beta) \tag{34}$$

with the angle β converted into radians.
In the next step, the minimum and maximum extension (in x and y direction) of the circle has to be computed to define the bounding box of the circle. To achieve this task, each point on the line segment for each angle was computed and the maximum (X_{max}, Y_{max}) / minimum (X_{min}, Y_{min}) was chosen. Following this task, the bounding box B of the coordinates P_1/P_2, I_1, I_2 and the coordinates of the circle extension was determined. Finally, the top left coordinate and the height and width of the bounding box was identified.

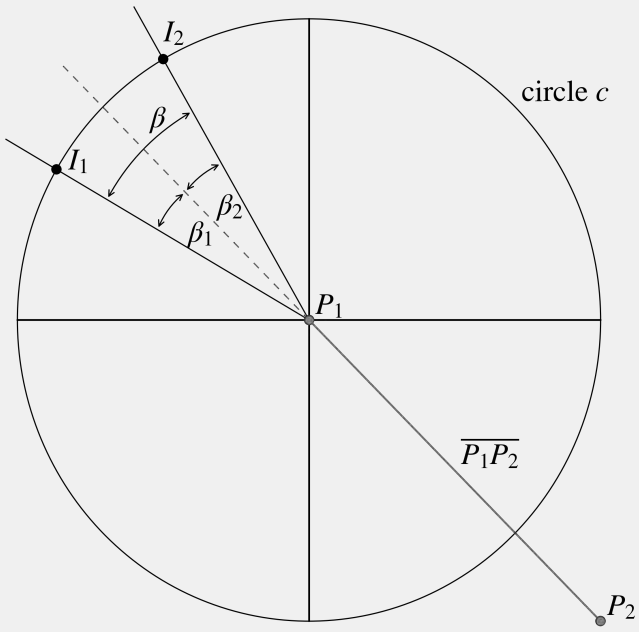


Figure 19. Visual representation of the calculation of the bounding box of a circle segment

The **definition of line features**, with regard to the start and end point, is an essential part before calculating the angle. If the x or y coordinate of both line points are the same, this step is redundant. While using real world coordinate points, it is very unlikely to happen. If the line is not defined, two different angles are calculated when changing the coordinates of the start with end point and therefore, the lines cannot be compared accordingly to their angle. Therefore, the start point of the line is defined as the point of the line with the smallest x and y coordinate. If the x coordinates of the start and end point are identical, then the point with the highest y value is chosen as the start point of the line. On the other side, if both y coordinates are the same, the point with the smallest x coordinate is defined as the start point of the line.

The **angle of the lines** is the most important condition for the vetocization process. In order to calculate the cut angle between two given lines, two steps are needed. First the angles for each line $\overline{P_1P_2}$ are created by using equation 35 which will return the angle in radians.

$$theta = arctan(\frac{dy}{dx}) \tag{35}$$

with

$$dy = P_2y - P_1y \text{ and } dx = P_2x - P_1x$$

Afterwards, the difference of the two angles will be calculated and converted to degrees using the conversion formula $degrees = radians \cdot \frac{180}{\pi}$.

Digital image processing can be used to **increase the visual quality of images** and therefore, improve the results from techniques like edge detection or *active contour models*. Without changing the local structure of the image, the size or the image's structure, these methods are called *point operations*. Therefore, every new pixel $a' = I'(u, v)$ exclusively depends on its previous value $a = I(u, v)$. Pixel a' is located at the same position as pixel a and is not dependent on any surrounding pixel. In order to compute pixel a', the function $a' \leftarrow f(a)$ or $I'(u, v) \leftarrow f(I(u, v))$ is applied to map the original pixel value. This operation is called *global* or *homogeneous*, if function f is not dependent on the coordinates of the image. Typical examples including contrast and brightness modification, gamma correction, color transformations quantizing and applying intensity transformations to the image [45]. After implementing some these techniques on pixels, the resulting pixel values have to be checked afterwards. A 8-bit grayscale image lays in between the range $[0...255]$. In order not to exceed this range, the result has to be clamped in some cases. Every values larger than 255 will be set to 255 and vice versa to zero for values, which are smaller than zero. Some of these techniques will be explained and applied, using the program *Matlab R2014a student version*, to a test image in the following chapter.

One of the easiest examples for a point operation, is **inverting images**. By the multiplication of the input image by -1, the ordering of the pixels is reversed, and a constant is added to map the resulting pixel values to the required range again [45]. The equation for inverting a pixel value $a = I(u, v)$ within the range $[0, a_{max}]$, can be formulated as

$$f_{invert}(a) = -a + a_{max} = a_{max} - a \tag{36}$$

Figure 20 shows the test image and the inverted version. It can be seen, that dark pixel values are translated into bright values and vice versa. The runway on figure 20a, which appears in bright pixel values is dark in figure 20b.

(a) Chosen test image **(b)** Inverted image

Figure 20. Test image and its inverted version

Another relatively easy point operation, is **thresholding**, which can be described as a quantization that classifies pixel values into a certain amount of different classes, depending on the given value $a_{threshold}$. The following threshold function $f_{threshold}(a)$ in equation 37 maps the given input image pixels to one of the two predefined values a_0 or a_1 ($0 < a_{threshold} \leq a_{max}$).

$$f_{threshold}(a) = \begin{cases} a_0, & \text{for } a < a_{threshold} \\ a_1, & \text{for } a \geq a_{threshold} \end{cases} \tag{37}$$

This technique is often used for binarization of an intensity image. Therefore, the values $a_0 = 0$ and $a_1 = 1$ are chosen [45]. Figure 21 shows the result for the thresholding technique for a test image. Values greater than 0.4 were classified as 1 and therefore, colored in white. All other values were set to 0 and are displayed in black.

(a) Chosen test image **(b)** Thresholded image

Figure 21. Test image and its thresholded version

Gamma correction is a fast and easy way to improve the contrast in an image by modifying the histogram of the input image. The technique is obtained by using different values of the parameter γ [67]. The basic equation for such a power-law transformation is

$$s = c \cdot r^{\gamma} \tag{38}$$

where the parameters γ and c (c = 1 in the following test image) as positive constants. Often equation 38 is written as $s = c \cdot (r + \epsilon)^{\gamma}$ to create a measurable result of the formula if the input gray value is zero. Offsets are normally not used in digital image processing, but play an important role of display calibration [54]. If the input gray value lays in between the range $[0, 1]$, then the resulting gray level stays in the same range. Furthermore, regardless of the chosen parameter γ, every function runs through the points $(0, 0)$ and $(1, 1)$. Figure 22 shows some values of the constant γ. As a result of the log transformation, fractional values of γ of power-law curves shift dark input values into a broader range of output gray values. Same holds true for the opposite with higher input gray level values [45].

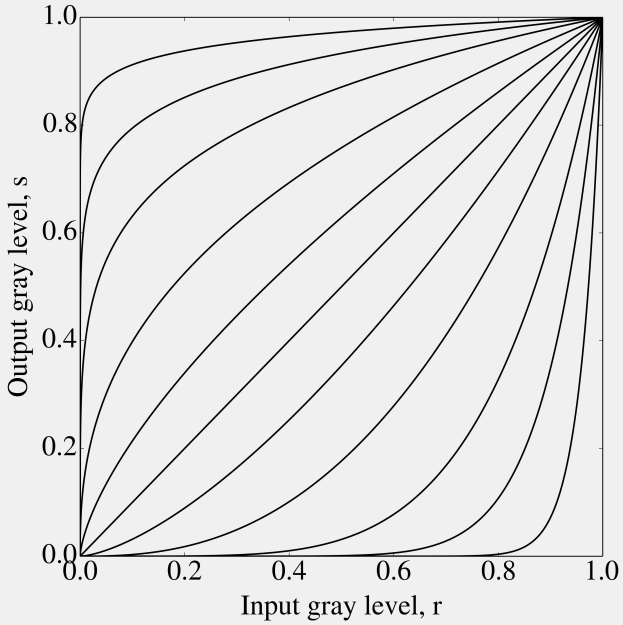


Figure 22. Visual representation of the γ values, from top to bottom, 0.04, 0.1, 0.2, 0.4, 0.67, 1, 1.5, 2.5, 5.0, 10.0 and 25.0

Figure 23 shows the test image and the resulting image after the gamma correction process. The gamma correction was calculated with a γ value of 1.75. A γ value larger than 1 was chosen to darken the relatively bright image.

(a) Chosen test image **(b)** Gamma corrected image

Figure 23. Test image and the gamma corrected version

In computer vision noise often refers to any part of an image, which is not the main part of the computation and is therefore, of no interest. In this specific case, line or edge detection, all

spurious signals of pixel values, which are caused by the the relief, the image acquisition system or other disruptive factors [68]. This subsection will explain two filters, the ***Gauss*** filter and the more complex ***Rudin-Osher-Fatemi (ROF)*** filter [69], which can be used to remove noise from an image. The basic principles of them will be briefly explained and results for both filters will be shown.
The *Gauss* filter

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right) \tag{39}$$

with σ as the standard deviation of the *Gauss* kernel, takes pixels at the center more into account than at the boundaries. Qualitatively it can be said, that the noise is suppressed by enforcing the condition, that pixels should look like surrounding ones. If γ, the standard deviation of the filter, is very small, e.g. less than one pixel, the smoothing effect will be very little, because of the very small weights for all pixels off the center. Secondly, if γ is chosen larger, the surrounding pixels will be weighted more strongly and, at the expense of blurring, the noise of the image will extensively disappear. Finally, if γ is very large, a lot of details in the image will vanish along side with its noise [53]. The result of the smoothing with a *Gaussian* filter can be seen in figure 24. A $6x6$ kernel and a σ value of 6 was chosen for this example.

(a) Chosen test image **(b)** *Gaussian* filtered image

Figure 24. Test image and the *Gaussian* filtered version

The *ROF* model is used for this specific case due to its underlaying principle. While finding a smoother version of the input image, the *Rudin-Osher-Fatemi* filter preserves structures and edges in the image and therefore, can be utilized for noise removal in edge detection [70]. This paper is applying the implementation of the *ROF* model based on the modifications by *Chambolle* [71].
The *total variation* (TV) of an image *I* is called sum of the gradient norm and can be written as

$$J(I) = \int |\nabla I| d\boldsymbol{x}. \tag{40}$$

The total variation can be expressed as

$$J(I) = \sum_{\boldsymbol{x}} |\nabla I| \tag{41}$$

in a discrete setting. The sum is taken from all coordinates $\boldsymbol{x} = [x, y]$ of image *I*.
The objective of the *Chambolle* version of the *ROF* is to find the denoised version *U* of image *I* that minimizes

$$\min_{U} ||I - U||^2 + 2\lambda J(U), \tag{42}$$

The term $||I - U||$ is the norm and calculates the difference between the denoised image *U* and the original image *I*. As a summary the model searches for image variations with jumps, which equals edges, between areas [70].The result of the presented *ROF* method is shown in figure 25.

(a) Chosen test image **(b)** Rudin-Osher-Fatemi filtered image

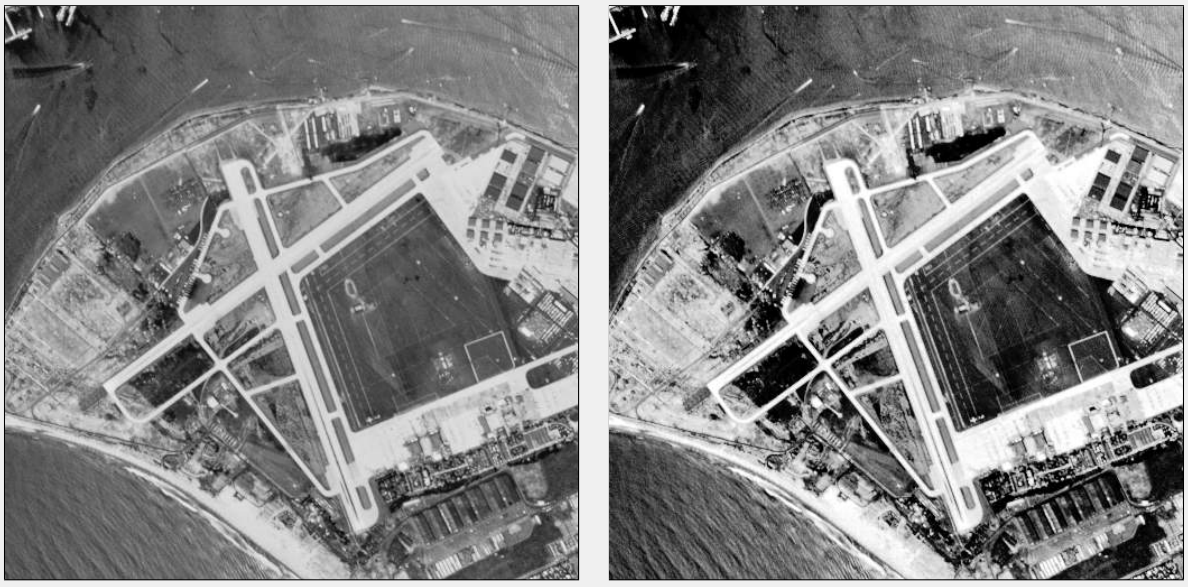
Figure 25. Test image and the filtered version with the Rudin-Osher-Fatemi filter method

Another useful tool and example of a graylevel transformation is the so called **histogram equalization** [70]. The major task of a histogram equalization is to find a point operation, which modifies the histogram of the image and approximates an uniform distribution [45]. Due to the fact, that homogeneous point operations cannot split but merge and shift histogram entries, and histograms are discrete contributions, only an approximation of the solution can be obtained. Consequently there are no possibilities to decrease or even remove individual peaks. Therefore, it is impossible to reach a truly uniform histogram distribution. The image can only be modified by point operations to an extent that the histogram is approximately uniform. The resulting question is how good the approximation will be and which, based on image content, point operation has to be applied to the image. In order to find such a point operation, a function is searched, which shifts the lines of the histogram so that the result, a cumulative histogram, is approximately linear. Equation 43 shows the point function.

$$f_{eq}(a) = \left\lfloor H(a) \cdot \frac{K-1}{MN} \right\rfloor \tag{43}$$

with the size of the image $M\ x\ N$, pixel value a in the range $[0, K-1]$. Function $f_{eq}(a)$ is monotonically increasing due to the positive constants M, N, K and the monotonic properties of the histogram of the original image $H(a)$ [45].
The histogram equalization technique can be classified and split up into two methods: *local* and *global* histogram equalization.

The global method uses the information of the histogram of the whole image as a transformation function. It stretches the contrast of regions with high histogram values and vice versa for low histogram regions. Important objects in the image have a wider and higher histogram region and their contrast is stretched. Contrast in the background of the image, with narrower and lower histogram regions is lost on the other hand. One downside of the global histogram equalization is the use of the global histogram information. Small regions lose the contrast and the background as well. Sometimes, these tradeoffs are desired but most of the time they are not.
The second method, the local histogram equalization or *block-overlapped histogram equalization*, overcomes the limitations of the global variant. In a first step a sub-block (rectangular) of the input image is defined and the histogram of that region is generated [72]. Afterwards, the histogram equalization function is calculated. Subsequent to the previous step, this function is used to equalize the center pixel of the window. The center of the chosen region is then moved to the contiguous pixel and the method is repeated for all pixels in the image. The computation complexity for this method is very high due to the large amount of pixels for even small images, e.g. 307,200 times for a 640 x 480 pixel image [73]. The result of the described method, global histogram equalization, can be seen in figure 26.



(a) Chosen test image **(b)** Histogram equalized image

Figure 26. Test image and the result of the global histogram equalization method

Automatic contrast enhancement, or **auto-contrast**, modifies the pixels, that the available range of pixel value is fully used. This point operation maps the current brightest and darkest values to the highest and lowest available intensity values. The intermediate values are linearly distributed. In order to stretch the image, the highest and lowest values a_{high} and a_{low} in the intensity range $[a_{min}, a_{max}]$ are needed. In a first step the smallest value a_{low} is mapped to zero and the contrast is

increased by the factor $(a_{max} - a_{min})/(a_{high} - a_{low})$. Finally the pixel value is shifted to the desired range by adding a_{min}. The full equation is defined as

$$f_{ac}(a) = a_{min} + (a - a_{low})\frac{a_{max} - a_{min}}{a_{high} - a_{low}}, \tag{44}$$

with $a_{high} \neq a_{low}$, which means, that the highest and lowest values of the image must not be equal. For an 8-bit image, equation 44 can be formulated as

$$f_{ac}(a) = (a - a_{low})\frac{255}{a_{high} - a_{low}}, \tag{45}$$

with $a_{min} = 0$ and $a_{max} = 255$.

Figure 27 shows the result for the automatic contrast enhancement method. Image 27a is ranging from 21 to 245. After applying equation 44 to it, the range is stretched from 0 to 255, see figure 27b. Differences between the two images can especially be seen at the darker values at the runway and at the beach in the southern part of the image.

(a) Chosen test image **(b)** Automatic contrast enhanced image

Figure 27. Test image and the result of the automatic contrast enhancement method

Automatic contrast enhancement (see equation 44) is strongly influenced by few outliers, very high or low pixel values. These pixel values may not be representative of the whole image and therefore cause errors in the contrast enhancement. This effect can be prevented by saturating a range of pixels (s_{low}, s_{high}), which defines the upper and lower ending of the designated intensity range. It is called **modified auto-contrast**. Therefore, two values a'_{low} and a'_{high} are determined. These limiting values have to be chosen in a way, that the user defined quantile q_{low} of the image I, pixel values are smaller than a'_{low}. The second quantile q_{high} has to be greater than a'_{high}. Values a'_{low} and a'_{high} can be calculated from the cumulative histogram $H(i)$ of the image and is fully dependent on its content.

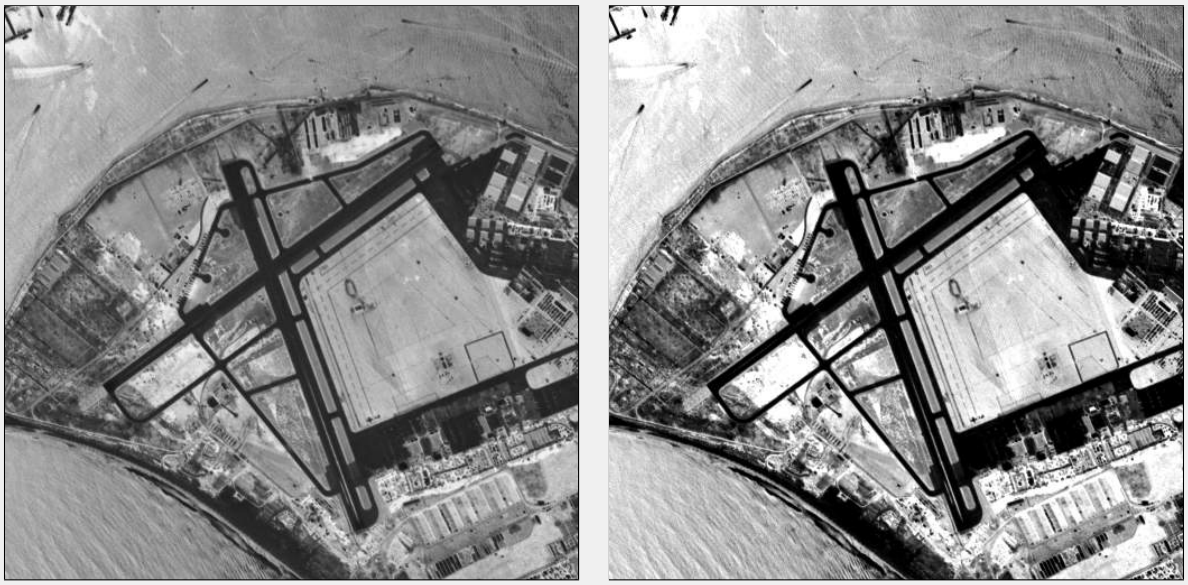
$$a'_{low} = i|H(i) \geq M \cdot N \cdot q_{low} \tag{46}$$

$$a'_{high} = i | H(i) \leq M \cdot N \cdot (1 - q_{high}) \tag{47}$$

with $M \cdot N$ as the amount of pixels in the image, $0 \leq q_{low}, q_{high} \leq 1$ and $q_{low} + q_{high} \leq 1$. All pixel values, which are not in the range $[a'_{low}, a'_{high}]$, are defined as new values a_{min} and a_{max}. In order to map intermediate values, a linear approach to the interval is applied. The mapping function f_{mac} can be seen in equation 48.

$$f_{mac}(a) = \begin{cases} a_{min}, & \text{for } a \leq a'_{low} \\ a_{min} + (a - a'_{low}) \cdot \dfrac{a_{max} - a_{min}}{a'_{high} - a'_{low}}, & \text{for } a'_{low} < a < a'_{high} \\ a_{max}, & \text{for } a \geq a'_{high}. \end{cases} \tag{48}$$

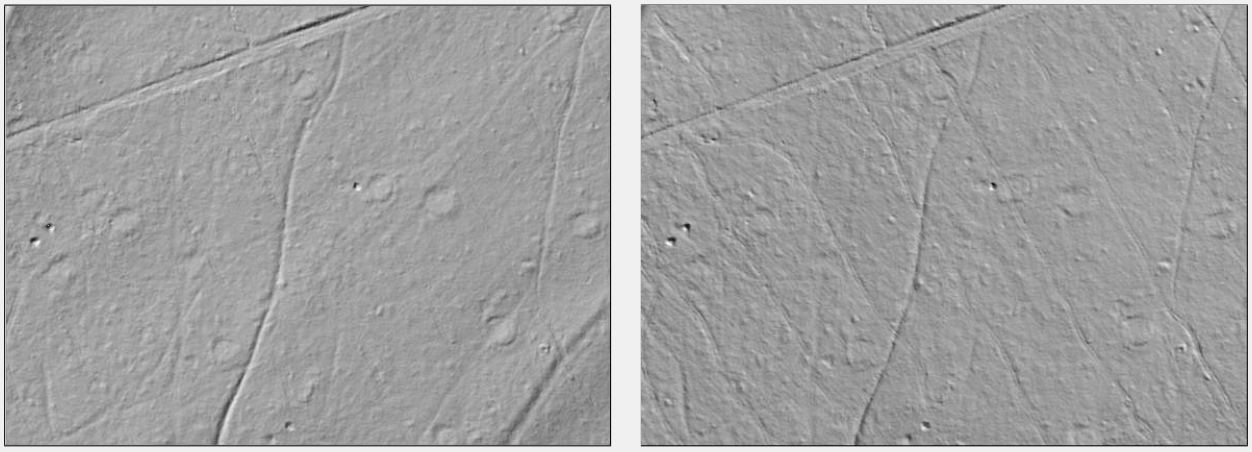
As described above, equation 48 does is not depend on outliers, such as extreme maximum and minimum pixel values. As a tradeoff, the mapping to minimum and maximum intensities is now based on a representative number of pixels. For the upper and lower quantiles, the same values are usually taken. Common values are $q_{low} = q_{high} = q$ with $q = 0.005 \ldots 0.015$ $(0.5 \ldots 1.5\%)$ [45]. The result for the modified automatic contrast enhancement method is displayed in figure 28. It shows the test image, figure 28a, and the result after applying equation 48. It can be seen that, the contrast is highly increased for the runway, the beach in the southern part of the image and the water in the top left corner.



(a) Chosen test image **(b)** Modified automatic contrast enhanced image

Figure 28. Test image and the result of the modified automatic contrast enhancement method with a chosen range from 5 to 95% of the image values

One technique, which reduces noise and dimensionality is to calculate the **average of** several **images**. Images can be averaged by summing them up and divide the sum afterwards by the total amount of images [68,70]. Figure 29b shows the average of all 360 hillshades of the test area and the hillshade itself is shown in figure 29a.



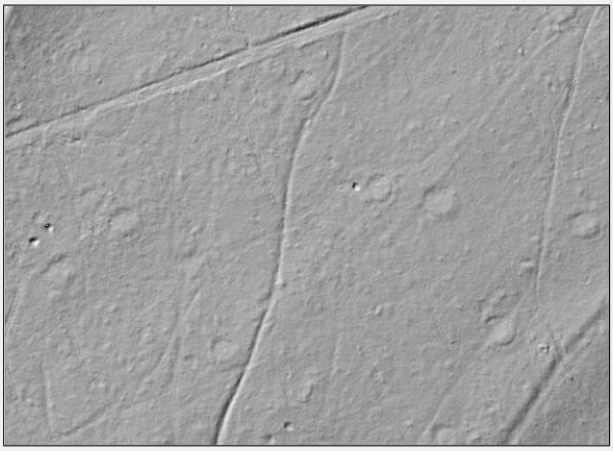
(a) Hillshade of the test area (315 degrees) **(b)** Average image

Figure 29. Hillshade of the test area (315 degrees) and the average image of all 360 hillshades

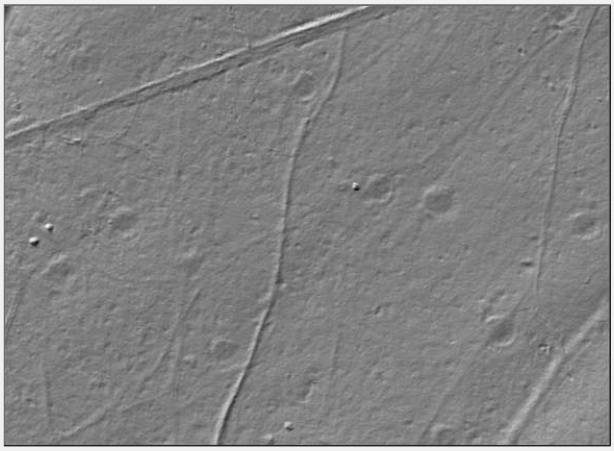
The last technique, which is presented in this chapter, is the calculation of **principle components of** several **images**. *Active contour models* or other level set functions utilize the visual information of the image to detect contours of desired features. Not only the chosen test area, but most of the national park area is covered by forest and therefore, skid trails are not visible on aerial imagery. To make these skid trails visible, hillshades of the digital terrain model can be used to visualize the data by shading the relief with a given, artificial light source in a certain azimuth angle. Depending on the angle, different features in the given digital terrain model are visible. If the artificial light source is perpendicular to the demanded feature, a shadow is visible in the exact opposite directing as the light source. If the angle is parallel to the feature, no shadow is generated and therefore, it is not visible on the hillshade. In order to save time by not having to perform the method 360 times, the principle components of the stack of all hillshades were computed. Principle component analysis is applied to reduce dimensionality and the result contains the variability of the input images with as few dimensions as possible.

In order to calculate the principle components of several images, some steps are needed. In a first step, each image has to be converted into a one-dimensional vector. A grayscale image with a resolution of $512*512$ is converted into 262,144 dimensional space. Afterwards, the one-dimensional image representations are stored in a matrix, with a row for each image. Before the calculation of predominant directions, rows are centered around the mean image. Singular value decomposition is normally used to state the principle components [70].

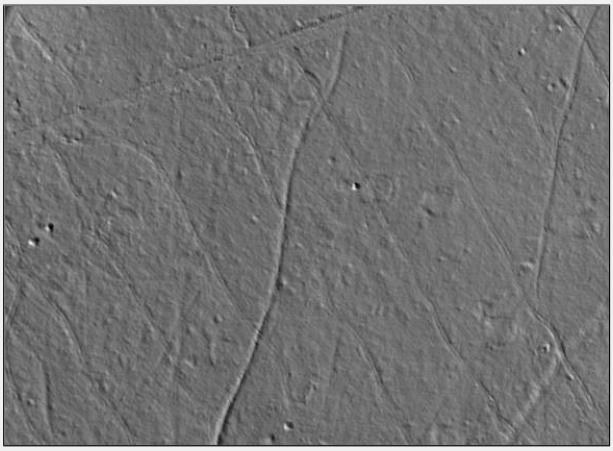
Figure 30 shows the hillshade of the test site with an azimuth angle of 315 degrees and the first three principle components of the hillshade stack with an angle ranging from zero to 359 degrees.



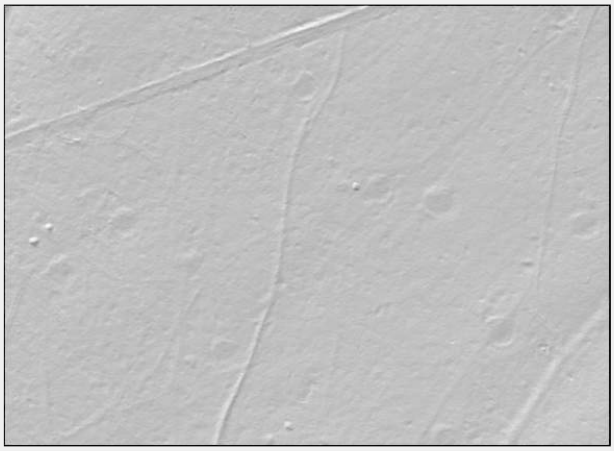
(a) Hillshade of the test area (315 degrees)



(b) First principle component



(c) Second principle component



(d) Third principle component

Figure 30. First three principle components of all hillshades with an azimuth angle ranging from zero to 359 degrees

All of the source code is available at github.com/sTAKKLE5/masterthesis.

3. Results

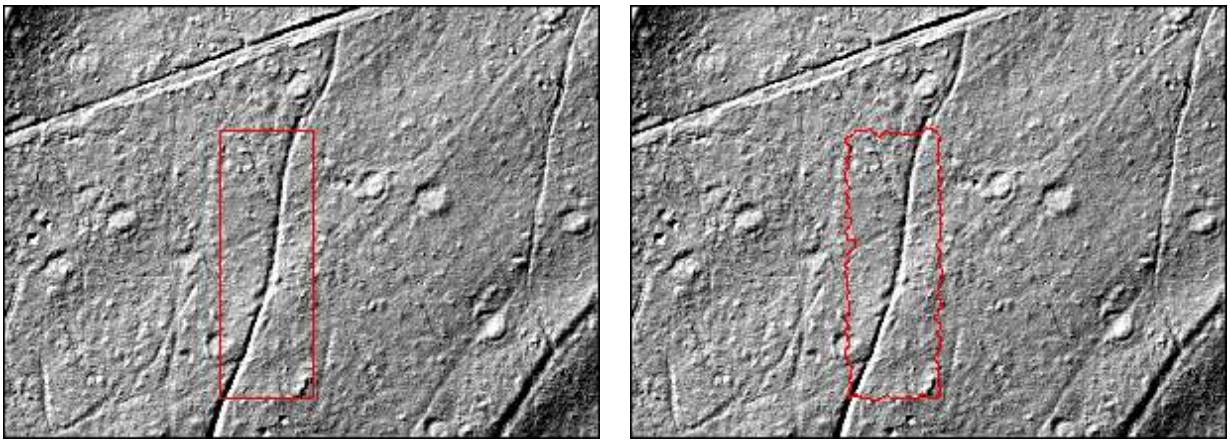
3.1. Active Contour Model derived Feature Accuracy

For the detection of features, such as roads and the desired skid trails, the hillshade of the digital terrain model with an illumination angle of 315 degrees was used. The variables of the four applications utilized by the *DRLSE* method are listed in table 3. Due to the lack of usable results, no further validation and evaluation of the results were carried out, but a visual interpretation and analysis.

Table 3. Used variables for the four applications of the *DRLSE* method

Variable	Value
Sigma σ	5
Timestep	5
Lambda λ	5
Alpha α	$\pm$1.5
Epsilon ϵ	1.5

In a first step, a large starting level set function was used. It was aimed at a detection of the trail in the center of the test cite, which has a strong contrast. Figure 31a shows this feature and the designed starting level set function in red. The level set function is located outside the borders of the trail and therefore, the shrinking version of the *DRSLE* method was chosen.

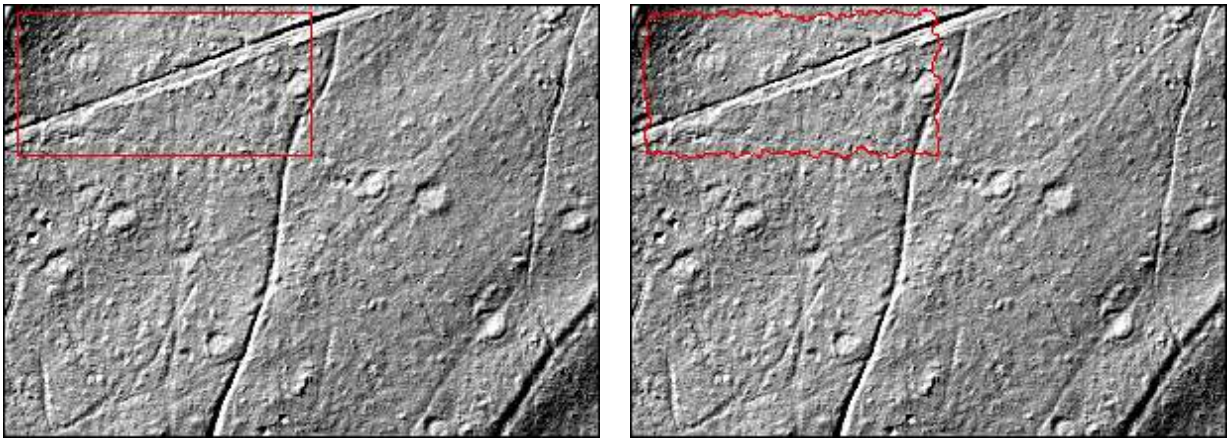


(a) Starting level set function **(b)** Resulting level set function

Figure 31. Attempted detection of a trail using the *DRLSE* method

Figure 31b shows the result of the *DRLSE* method. The iterations were manually stopped, after the shape of the level set function did not significantly change anymore. It can be seen, that the initial shape changed, but did not reach the desired feature.
In a second application of the *DRLSE* method, it was aimed at the detection of the main road in the northern part of the test image. Using a large initial level set function, the detection rate on a large scale was looked at. Furthermore, as in the first application, the level set function surrounds its target and

the shrinking version has been used. The initial shape of the level set function is shown in figure 32a. The desired feature, the main road, has a larger contrast than the surround and the width of the road, approximately seven to eight meters, and is larger than skid trails.



(a) Starting level set function **(b)** Resulting level set function

Figure 32. Attempted detection of the main road using the *DRLSE* method

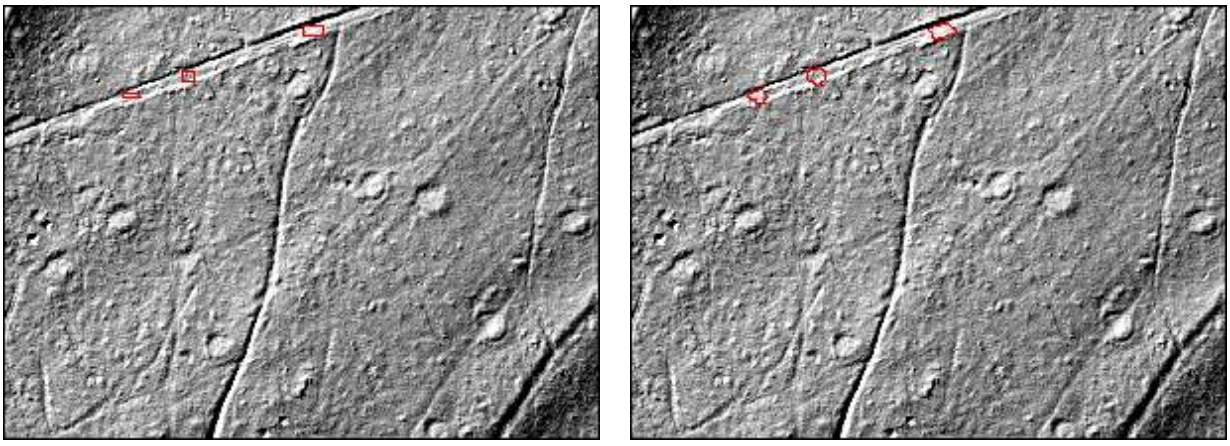
As like in the first use of the method, the iteration process was stopped after the shape of the level set function did not significantly change anymore. The result after the iterations can be seen in 32b. The level set equation was not able to capture the boundaries of the main road. On the eastern boundary, the function adapts to the boundaries of a former coal pile. On the other hand, the main feature, the road was not detected.

The *DRLSE* method does not only allow a shrinking of the level set equation, but an expansion as well. This feature was used to detect the wide and uniform main road. Due to its width, the level set function can be placed inside of it and detect the borders from the inside. In order to achieve this task, not only one equation was applied, but three small ones. It was aimed at an extensive recognition of the feature. Figure 33a displays the initial level set functions in the main road in the northern part of the hillshade.

The resulting level set function, which was again stopped manually after no significant changes occurred anymore, did stop at the boundaries of the main road and did not violate them, see figure 33b. The size of the initial equation did not change significantly and only small parts of the main road were detected. The three level set functions did not recognize any further, related parts of the road. Furthermore, the desired union of the three level set functions did not occur either.

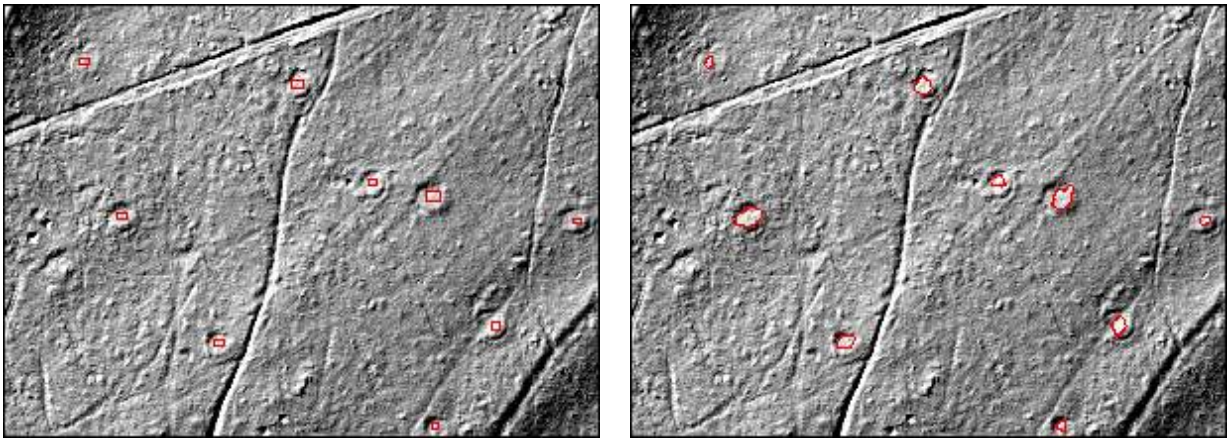
The last test, using *active contour models* for feature detection, is based on the partial detection of the coal pile in figure 32, and is aiming at the detection of it. For this task, multiple initial level set functions were created in the center of the desired coal piles, see figure 34. The homogeneity of the contrast was used for an expanding *DRLSE* method.

In direct comparison to the other three feature detection runs, most of the former coal piles were detected well. None of the resulting level set function did violate the boundaries of it, see figure 34. Furthermore, in four out of nine resulting equations, the whole feature was detected. The resulting five functions did partially recognize their features.



(a) Starting level set function **(b)** Resulting level set function

Figure 33. Attempted detection of the main road using the *DRLSE* method using multiple, expanding initial level set equations



(a) Starting level set function **(b)** Resulting level set function

Figure 34. Detection of coal piles using the *DRLSE* method

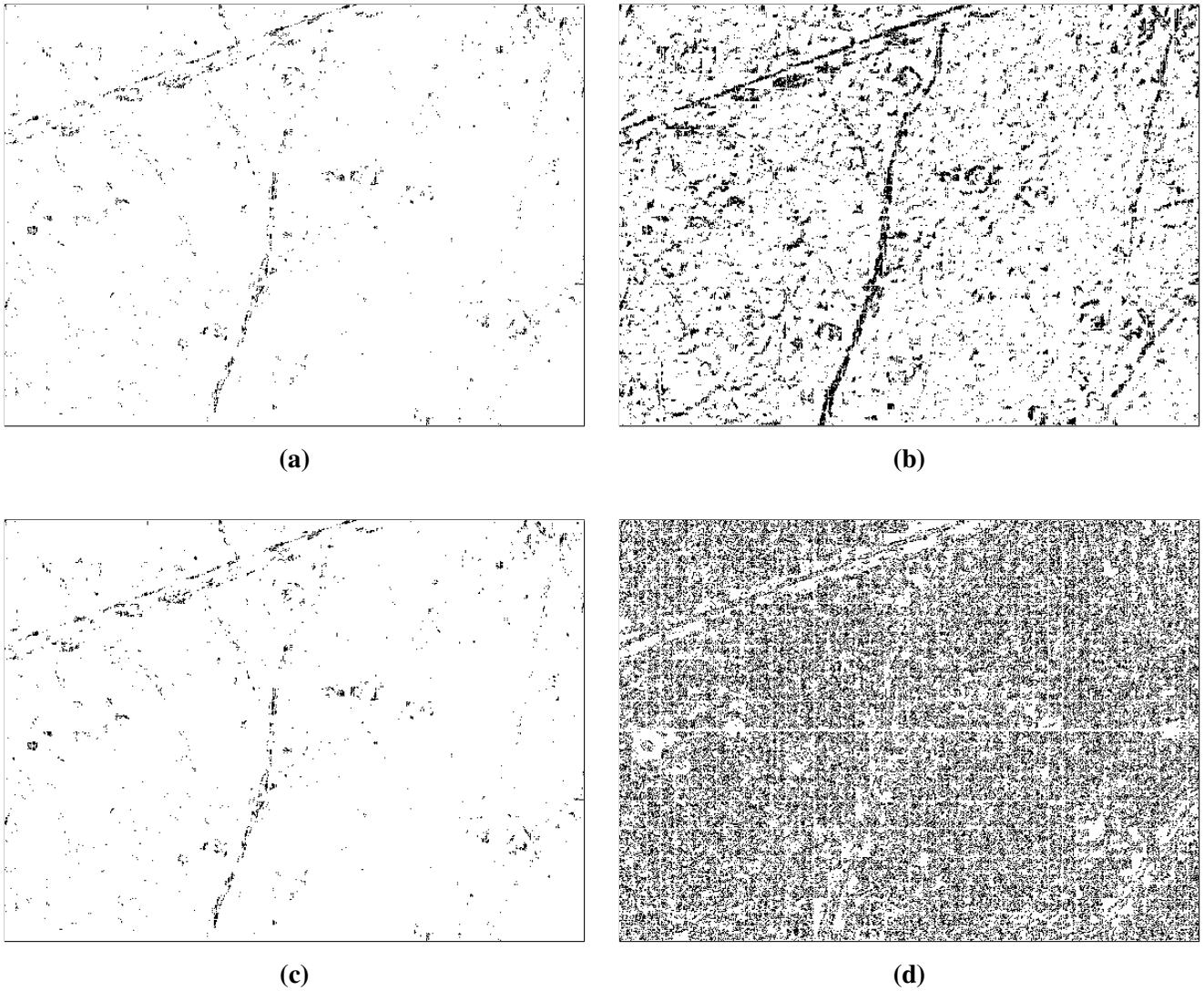
3.2. MWSD *Filter derived Binary Pixel Accuracy*

The *MWSD* filter was carried out for the digital terrain model, using different settings. These settings are displayed in table 4.

Table 4. Used settings for the *MWSD* filter for the chosen test site

Image	**Settings**		
	Type	**Standard deviation range**	**Distance range [m]**
Figure 35a	DTM	[1.0;1.5]	[0.5;1.0]
Figure 35b	DTM	[1.0;1.5]	[0.5;1.5]
Figure 35c	DTM	[1.0;1.75]	[0.5;1.0]
Figure 35d	SLOPE	[1.0;1.5]	[0.5;1.0]

The first column sets the reference to the resulting binary raster shown below. In the next column, the input variables are listed. Four out of five runs used the digital terrain model, DTM. One filter run utilized the slope product, which was computed from the digital terrain model. Lastly, the input values, the range for the standard deviation and the distance are part of the third and fourth column. For the first variable, the standard deviation, an absolute range from one standard deviation to 1.75 times the standard deviation of the moving window has been chosen. The distance is ranging from 0.5 to 1.5 meters. These variables have been visually derived from the perpendicular height profiles of skid trails in the test area, see figure 3.
The results of the different filter settings can be seen in figure 35.



(a) **(b)**

(c) **(d)**

Figure 35. Results from the *MWSD* filter using different variables and input images

For the validation of the resulting binary rasters, the digitized roads, rifts and skid trails were buffered with a width of three meters, which leads to a total width of six meters with a resulting area of $11,524.1517m^2$. In a next step, all pixels with a value of one, which are located inside the buffered features were selected and counted. The results are can be seen in table 5.

Table 5. Number of pixels laying inside and outside the buffered feature area in the test area. Percentage of pixels calculated from the total amount of pixels, 246,186, in the test image

Image	Detected pixels	Detected pixels inside the buffer	Detected pixels outside the buffer
Figure 35a	3,650 (≈ 1.5%)	1,694 (≈ 46.4%)	1,956 (≈ 53.6%)
Figure 35b	42,432 (≈ 17.0%)	14,815 (≈ 34.9%)	27,617 (≈ 65.1%)
Figure 35c	4,036 (≈ 1.6%)	1,939 (≈ 48.0%)	2,097 (≈ 52.0%)
Figure 35d	94,849 (≈ 39.0%)	17,089 (≈ 18.0%)	77,760 (≈ 82.0%)

The results for the different runs of the filter, which are displayed in table 5, strongly vary. The amount of recognized pixels ranges from 1.5 to 39 percent of all available pixels of the image. In the first run, figure 35a, the used variables are listed in table 4, 3,650 out of 246,186 pixels were recognized as parts of the features by the filter. These 1.5 percent can be divided into two parts, pixels inside or outside the buffer of the digitized features. 1,694 pixels, which equals approximately 46.4 percent, are located in the buffered feature area. On the other hand, 1,956 pixels, 53.6 percent, are not within the 6 meter buffer width.

The second result, visually displayed in figure 35b, which was applying an increased distance range, see table 4, shows a large increasing number of detected pixels. 42,432 pixels were classified by the filter as a feature, which is 17 percent of all available pixels. This increasing number of recognition rate leads to an increasing amount of pixels located inside and outside the buffer. In direct comparison to the results from the first run, these two values are not equally balanced, but the amount of pixels, which are not in the buffer is nearly double the size of pixels inside the buffer. 65.1 percent are not located in the buffer and only the resulting 34.9 percent are truly part of the desired feature.

Figure 35c shows the results of the filter using an increasing standard deviation range, see table 4 for the exact values. In the third run, more pixels were classified than in the first, but less than in the second. 1.6 percent, 4,036 pixels, were recognized by the filter with the differing input variables. If you compare the pixels, which are laying inside the buffer width from this run to the first result, it can be seen, that a higher percentage is located inside the buffered area. Like in the first result, the majority of pixels, 52.0 percent, are still falsely classified as features.

All three described filter runs have been using a digital terrain model, as the input image. For the fourth run, figure 35d, the slope, which is derived by the digital terrain model, has been used. In this case, most of the pixels in all four runs have been classified as parts of desired features, like trails and roads. 39 percent of all image pixels, which are 94,849 pixels, were detected. The ratio of pixels inside to outside the buffer has to be named as well. More than 80 percent were laying outside the buffer shape and only the resulting, less than 18 percent, are real parts of the buffer area.

When visually interpreting the four results, figure 35, it can be said, that three of them show the desired features. In figure 35a and 35c the main road in the norther part and, with some further background knowledge about the location of the skid trails, parts of the trails can be seen. Furthermore, the rift, running in the center from north to south, is clearly visible as well. These features can also be seen in figure 35b. In direct comparison to 35a and 35c, they are distinctly more apparent. On the other hand, the smaller features, like the actual skid trails, are not visible due to an increasing amount of noise anymore.

The last result, displayed in figure 35d, does not show features anymore. The absence and shape of classified pixels in the norther part can be used to identify the main road.

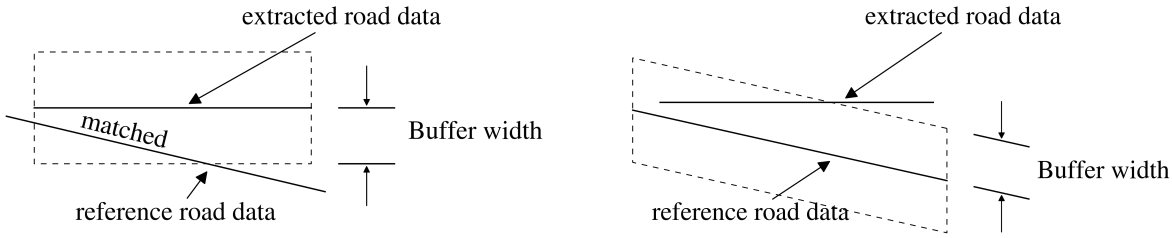
3.3. MADV *Method derived Feature Accuracy*

The vectorization method, which has been described in chapter 2.8, has been performed using the results shown in figure 35a with the variables as seen in table 4 in the first row. It has been chosen due to the high detection rate of pixels inside the buffered area. In order to decrease the running time of the algorithm, it was chosen, even though the detection rate was lower than the result of figure 35c.
The needed input variables were chosen after a visual analysis of the filter result, see figure 35a. A maximum search angle of 30 degrees was selected. The maximum distance between nodes was set to 20 meters.
For the validation of the outcome, the method introduced by Heipke, *et al.* [74] was chosen. This method needs some further information about the reference data set and the outcome. These variables were calculated and are listed in table 6.

Table 6. Required variables for the validation method by Heipke, *et al* [74]

Variable	**Value [m]**
Length of reference network	2,193.68
Length of matched reference	1,469.20
Length of matched extraction	1,394.72
Length of extracted data	2,051.53
Length of unmatched reference	724.50
Length of gaps	416.32
Chosen bufferwidth	3
No. of gaps	8

The two variables, the length of the *matched reference* and *extraction* are explained in figure 36. Both of them are created by buffering the corresponding data with the given buffer size, three meters. For the creation of the *matched reference*, figure 36a, the extracted road data is buffered and clipped with the reference road data. The *matched extraction*, figure 36b, was created by buffering the reference road data and clipping it afterwards with the extracted road data.



(a) Matched reference **(b)** Matched extraction

Figure 36. Matched reference and extraction. Based on [74]

The variables are used to compare different techniques and methods for digitization of roads and cover the *completeness*, *correctness*, *quality*, *redundancy*, *number of gaps per kilometer* and the *mean gap length*. Table 7 shows the results for the calculated variables.

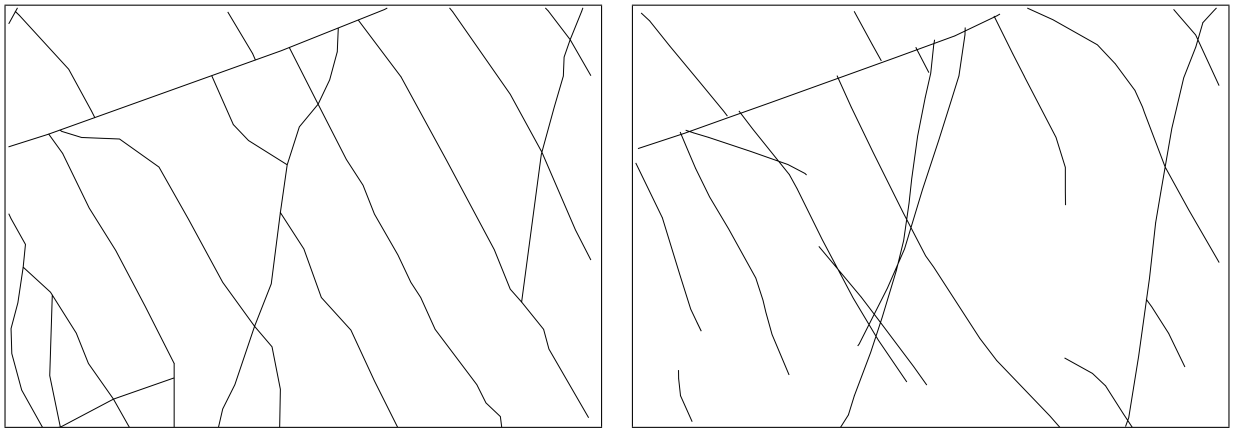
Table 7. Results of the *MADV* method

Measurements	**Value**
Completeness	0.67
Correctness	0.68
Quality	0.50
Redundancy	0.05
No. of gaps per kilometer	3.65
$\mu_{gap\ length}$ [m]	52.04
RMSE [m]	2.7

The *MADV* method reached an overall completeness of 67 percent with a correctness of 68 percent. The quality of the resulting features, which is described by the following equation

$$\text{quality} = \frac{\text{length of matched extraction}}{\text{length of extracted data} + \text{length of unmatched reference}},$$

is the measurement of the overall "goodness" and ranges from zero to one. The redundancy of the *MADV* method, describes the amount of overlapping features in percent, is at five percent.
Worse results are achieved concerning the amount and length of gaps in the resulting features. Statistically, 3.65 gaps occur every kilometer in the resulting features, with a mean length of 52.04 meters. The root mean squared error of the calculated skid trail network has a value of 2.7 meters.
Figure 37a shows the resulting features, derived by the described *MADV* method. The basic structure of the trail network has been detected. Furthermore, the main road on the top part of the test side and the skid trail running from the top to the bottom have been fully vectorized by the algorithm. Problems occurred regarding steep curves and junctions. In the center and on the borders of the result, not only small gaps, but large parts of the network are missing. The data set, which was used for validate the result, is displayed in figure 37b.



(a) Validation data set **(b)** Result of the vectorization method

Figure 37. Validation data set and the resulting network of the vectorization method

4. Discussion

4.1. Active Contour Models

Several studies have been carried out applying *active contour models* to locate roads and other features from aerial imagery, achieving good results. Seppke, *et al.* used a modified *active contour model* to successfully recognize coastlines in a SAR image [36]. Comparing the input data, the SAR imagery and the data for this paper, a digital terrain model, one major difference can be spotted. The contrast ratio between the background and the desired features, the coastlines and the skid trails, strongly differ. Coastlines stand out against the lighter water surface. Furthermore, the noisy water surface can be smoothed by filters, without loosing information about the coastline, due to the very high contrast difference. In this paper, the contrast difference between skid trails and the background of the hillshade is rather low, which decreases the detection rate of desired features. An application of filters, to reduce the noise in the background of the hillshade, is also not possible because of the low contrast values of pixels containing skid trails and non skid trail pixels.

Mayer, *et al.* applied *Ribbon Snakes* to automatically extract rural roads in high resolution aerial imagery [37,39]. Achieving very good results with a correctness of 97 percent, a completeness value of 83 percent and a RMSE of 0.46 meters, the input data and the aimed at extraction features strongly differ from the data in this paper. Not the extraction of smaller skid trails, but wider, rural roads are the center of the study. These roads have a higher contrast against the surrounding area. The bright concrete of the roads stand out against the darker fields and settlements. Furthermore, no strong contrast differences along the road network occur. The width of the roads can be utilized to apply the initial level set equation in the center of the network. As a result of the strong border between the background and the roads, the final level set equation does not violate the boundaries of the driving lanes. Such a continuously high contrast value is not available in the hillshades concerning skid trails. Therefore, and because of their much smaller width, it is not possible to place the initial level set function inside the skid trails.

Active contour models are widely used in medical informatics for segmentation [31,33,35]. In this scope, *active contour models* achieve very good results, which can be explained by the input images. Bredies and Wolinski use microscopy images to localize yeast cells [33]. The overall contrast of these images is not very high. One of the main reason for the very good detection rate of the cells is the delimitation among the cells by a darker boundary. An initial level set equation, which is placed in this case in the center of each cell, cannot violate the continuous boundary of the cell. It is possible to apply an expanding level set equation to extract the spatial expansion of each cell. Applying this technique to hillshades of the test area, comparable results cannot be achieved due to a missing, distinct border. Magnetic resonance imagery is used for segmentation of bone in clinical knee [33]. As like in previous applications of *active contour models*, this paper relies on imagery with a high contrast and clearly defined borders.

Concerning the results, the desired features were not detected. Unlike as in previous studies, which detected roads and structures from high resolution satellite imagery [16,18–20,36,37,39], it was not possible for the method to visually differentiate between the noisy background and the skid trails. The resulting level set equation was not able to adapt to the boundaries of the skid trails, but stopped after an insignificant distance at the wrong place. The only features, which were recognized by the *DRLSE* application, were coal piles.

When discussing the application of *active contour models*, and in this specific case the *DRLSE* method, for the detection of skid trails in hillshades, two major difficulties have to be addressed, the method itself and the results. Starting with the application, further knowledge is needed. It does not only include information about the test site and the data, but understanding about digital image processing, too. *Active contour models* and the *DRLSE* method need to be tuned by different variables to lead to suitable results. As an example, the kernel size of the *Gaussian* filter or the utilized time step can be named. If the required information is available, the use of the program is fairly easy by manually creating initial level set functions around or inside the desired features. The creation of the underlaying algorithm can lead to problems, too. *Active contour models*, depending on the chosen variation, can result in numerical errors and reach incorrect outcomes.

The use of *active contour models* for the detection of skid trails leads to a couple of other problems. First, for every skid trail an initial level set equation has to be manually defined by an operator. With an increasing size of a test area, the amount of skid trails and the amount of required initial level set equations ascends. Due to the fact, that the initial level set equation has to be placed fairly close to the skid trails in order to detect them, the user has to work precisely to gain good results. This functioning is very time consuming. The second problem is the basic principle of the *DRLSE*, which is the detection on a visual level. *Active contour models* use contrast differences to recognize feature in the image. In this case, the data, which is able to detect skid trails, are digital terrain models. This kind of data does not have high contrast differences and therefore, hillshades have to be created out of them. When calculating hillshades, an illumination angle has to be defined. That can be described as the direction of an artificial light sources. Depending on the chosen angle, different features are visible. In order to detect all skid trails, several runs of the *DRLSE* application need to be carried out with varying hillshades. These runs will directly lead to an increasing operation time.

The input data, the hillshades, can be named as the main reason for the poor localization rate of the

desired skid trails. Not enough contrast between the background and the features is available for the the method to detect skid trails. Furthermore, skid trails are not wide and equally enough to use expanding initial level set functions. Looking at the coal piles in the hillshades and at the good detection rate of them, the homogeneous contrast ratio can be named as the main reason. It was also possible to avoid the noisy background by using an expanding initial level set equation due to the large spatial resolution.
In order to improve the quality of the results, input data with a higher spatial resolution could be used. With an increasing spatial resolution of the digital terrain model, the quality of the computed hillshades will enhance, perhaps less noise will occur in the background and therefore, the *DRLSE* could be able to detect skid trails. In some cases the current available data can visually be enhanced by applying edge preserving filters, which reduce the noise in the background or increase the contrast. These techniques have been carried out for the provided data, but did not lead to results.

4.2. Moving Window Standard Deviation Maximum Distance Filter

Just as in the previous chapter, the discussion of the *active contour model*, two major topics of the *moving window standard deviation maximum distance filter* have to be named. First, the method itself will be addressed. The underlaying principle, comparing the height difference and the distance between every possible pixel combination of the image is resulting in a long running time. This long computation time makes it nearly impossible to apply the method on a larger area. For the chosen test image, with a pixel resolution of 582×423 pixels and a total amount of 246,186 pixels, the application took about an hour. The computation was carried out at a MacBook Pro with 2.9 GHz Intel Core i7 and 16 GB memory capacity.
Secondly, the resulting binary image of the *MWSD* filter needs to be discussed. On a first look at the results, linear structures are visible. Pixels laying in every desired skid trail were localized. Subsequently, these pixels can be used to connect linking nodes and therefore, can be utilized to detect the full feature. On the one hand, parts of the desired features were recognized. But on the other hand, the distance between connecting pixels is in some cases too large. Not enough pixels were detected in some parts of the skid trails to set up a satisfying distance between nodes. Should this distance be too large, it might not be possible to connect these pixels in a subsequent step, the *MADV* method, and therefore, gaps in the resulting skid trail network will occure.
Looking at the possibles causes of the results, two reasons can be named: The input data and the applied variables of the *MWSD* filter. This paper relies on a digital terrain model with a spatial resolution of 0.5×0.5 meters from the *Landesamt für Vermessung und Geobasisinformation Rheinland Pfalz*. The height distance of skid trails to the surrounding forest floor lies in the twenty to thirty centimeter level. Due to the fact, that the input digital terrain model is interpolated from airborne LiDAR data, information can be lost. For a lot of scenarios, which are not dealing with these small height differences, these losses do not matter. But in this case, the loss of information means, that skid trails might not be detected. Input data with a higher spatial resolution could prevent these losses and might lead to better results. The recording date of the terrain model has to be addressed, too. During a filed campaign, several skid trails were measured. This data was ought to be used for validation. While validating the results, the

surveyed skid trails were not recognized in the digital terrain model. The data was created in the year 2008. In order to detect all skid trails, up-to-date data is needed.
To address the running time problem, the method could be translated into a faster and more storage efficient programming language like *C++* or *C#*. To improve the overall quality of the results, the input terrain model could be split up into several smaller parts. This technique allows the use of different variables, the maximum height difference and the distance between pixels. In some parts of the test image good results were achieved, whereas in other regions these values did not lead to satisfactory results. By using smaller parts of the input image, the needed variables could be tuned, which might lead to better results. The major disadvantage of this technique is the increasing running and preparation time.

4.3. Maximum Angle Maximum Distance Vectorization Method

After the run of the *maximum angle maximum distance vectorization method*, the method itself and the results need to be discussed. On a first view of the resulting skid trail network, good results were achieved. The method was able to detect and connect linking nodes. In order to operate, the function needs start lines for every skid trail, which have to be vectorized. These parts of skid trails need to be digitized by an operator in a first step. To gain these trails, further work, like an analysis and evaluation of hillshades of the test site, has to be carried out. Depending on the size and complexity of the chosen area, this task can be complex and very time consuming. In general, the method showed that it is possible to use the binary image, resulting from the *MWSD* filter, to connect linking nodes and vectorize the skid trail network.
Looking at the quality of the result of the *MADV* method, a differentiation has to be carried out. Nodes, which are belonging together and constitute a straight line are not error-prone. On the other hand, the *MADV* method does not handle curved parts of the skid trail network well. They are defined by corners or hard junctions. Due to the given maximum angle, the search angle for the function, curves in the network can be neglected by the algorithm. In such a case, line combinations with a smaller difference, with regard to the angle, are more likely to be chosen and falsely used by the method as a part of the skid trail result. On account of the underlaying principle, which is the connection of nodes with the smallest angle difference, curves and hard junctions are the main weaknesses of the *MADV* algorithm. Furthermore, the distance between related nodes has to be addressed, too. In some cases, the defined maximum distance, is not enough to connect nodes. Gaps in the resulting skid trail network will be the result. It is caused by the previous run of the *MWSD* filter and the missing of detected pixels. The search distance can be maximized, but it directly leads to an increasing, falsely vectorization. With a larger distance, more pixel combinations are analyzed and the risk of linking wrong nodes grows.
The *MADV* method could be improved by utilizing smaller parts of the input binary image. With this technique, the two necessary variables, the search angle and maximum distance between nodes, can be tweaked and optimized. It allows to enhance the program not only for the full image, but also takes local anomalies into account. When applying this method, two issues occur. First, for every subset of the input image, the perfect variables need to be defined by calculating the maximum distance between nodes and by measuring the maximum angle. The second problem are the required start lines for each

skid trail. The amount of start lines will increase. For every subset, the user has to vectorize parts of the skid trails in a first step. For a small area, this technique can be performed without any further problems. But as soon as the area grows, the amount of time, needed for vectorization, will increase and will not be in proportion to the better results anymore.

The created **MADV algorithm can be improved** by including some directives and assumptions on skid trails. The layout of the skid trails plays an important role for the practicability. The shortest distance between skid trails, machine ways and forest roads, is aspired. Therefore, most of the time, a straight alignment is used if the terrain allows it. Connections on both sides of the trails to machine ways or forest roads, which can result in a faster transportation of logs, are often created. The slope has to be considered as well. With a slope larger than 15 percent, the skid trails have to be created in the same direction as the slope. If the trails are created parallel to the slope, the risk for heavy machinery of tipping over is too high. The second variable, that has to be considered, is the cross slope. In order to reduce damages in the forest, the risk of tipping over and to decrease the width of skid trails, the cross slope must not be larger than five percent. Junctions of skid trails with forest roads should be made in an angle between 45 to 60 degrees [75]. The distance between skid trails also can be used. Under normal conditions, these trails are build every 20 to 60 meters [7,43,75]. This regularity can be utilized to avoid miss classifications, such as natural sinks and drainage systems. Furthermore, most of the time, if the terrain is suitable, the trails are parallel to each other, compare figure 5. Skid trails do not intersect the forest road network. Such an existing road network, e.g. ATKIS® from the *Landesamt für Vermessung und Geobasisinformation Rheinland Pfalz* [44], can be applied to define the end of the localized feature. If the detected skid trail intersects the network, the intersection point should be used as the ending point of the skid trail, and the algorithm has to stop and continue with the next startline.
Not only man made structural variables of skid trails can be applied to improve the algorithm, but natural conditions as well. To exclude rifts and natural drainage systems, the depth of a detected line feature plays an important role. Thresholds could be used to remove the deeper rifts from the searched skid trails. Terrain profiles, such as figure 3 or on-site measurements, can be used to identify suiting threshold values. Thus, it would be possible to compare the mean depth of a detected line feature within a certain length with the user defined values. Not only the depth, but the width of the detected feature can be used for a differentiation between natural structures and artificial trails. Utilizing heavy machinery for logging needs a certain width of the trails. This width ranges between four to five meters. If the width is too small, than the machines cannot reach the destination, whereas a too large width is avoided by foresters and forest workers [75].

5. Conclusions

This paper describes a simple, and yet effective method for the detection of skid trails in airborne LiDAR data, using a standard deviation moving window filter and a vectorization method for the resulting binary image. The accuracy of the developed method has been compared to *active contour models*, which could not lead to any results in the chosen test area. On the other hand, the combination of the two developed

methods detected 67 percent of the road and skid trail features with a correctness of 68 percent and a rmse value of 2.7 meters. Due to the very high computation time and the use of predefined parts of skid trails, the method has to be classified as experimental, and is not yet suitable for an operational use. The imposed requirements of the two algorithms were partly met. It was proven, that it is possible to recognize and detect skid trails in the available digital terrain model. Problems occurred regarding the requirements of an automatic and a fast method, which can be used on a large scale. Nevertheless, the underlaying principles can be used for an improved version with an enhanced running time and a more automatic method.

Acknowledgments

This research was supported by Dr. Ernst Segatz of the Research Institute for Forest Ecology and Forestry of Rhineland-Palatinate. Furthermore, the support and further background information by Michael Schimper of the Expertise Center Forestry Technology of Rhineland-Palatinate, has been much appreciated.

Conflicts of Interest

The author declares no conflict of interest. As not differently stated, royalty free images have been used in this paper. The aerial images of the test area are provided by the *Landesamt für Vermessung und Geobasisinformation Rheinland Pfalz* [44] and were acquired in the year 2013.

References

1. Timberjack Inc.. TIMBERJACK HARVESTER 1270D. http://www.fagro.edu.uy/~forestal/cursos/tecmadera/Gustavo/1270D-Brochure.pdf, 2008. retrieved: 2015-15-4.
2. John Deere. John Deere 1270E IT4, 1470E IT4. http://www.deere.de/de_DE/docs/html/brochures/publication.html?id=36869068#1, 2014. retrieved: 2015-15-4.
3. Pinard, M.; Barker, M.; Tay, J. Soil disturbance and post-logging forest recovery on bulldozer paths in Sabah, Malaysia. *Forest Ecology and Management* **2000**, *130*, 213–225.
4. Pierzchała, M.; Talbot, B.; Astrup, R. Estimating Soil Displacement from Timber Extraction Trails in Steep Terrain: Application of an Unmanned Aircraft for 3D Modelling. *Forests* **2014**, *5*, 1212–1223.
5. Montgomery, D.R. Road surface drainage, channel initiation, and slope instability. *Water Resources Research* **1994**, *30*, 1925–1932.
6. Bundesministerium für Ernährung und Landwirtschaft (BMEL). Der Wald in Deutschland - Ausgewählte Ergebnisse der dritten Bundeswaldinventur. http://www.bmel.de/SharedDocs/Downloads/Broschueren/Bundeswaldinventur3.html, 2014. retrieved: 2015-15-4.
7. Deutschland eV, PEFC. PEFC-Standards für nachhaltige Waldbewirtschaftung, 2014.
8. Forest Stewardship Council, FSC Deutschland. Deutscher FSC-Standard - Deutsche übersetzte Fassung. www.fsc-deutschland.de/download.fsc-waldstandard.21.pdf, 2012. retrieved: 2015-15-4.

9. Zheng, J.; Wang, Y.; Nihan, N.L. Quantitative evaluation of GPS performance under forest canopies. Networking, Sensing and Control, 2005. Proceedings. 2005 IEEE. IEEE, 2005, pp. 777–782.
10. Baltsavias, E.P. Airborne laser scanning: basic relations and formulas. *ISPRS Journal of Photogrammetry and Remote Sensing* **1999**, *54*, 199–214.
11. Krogstad, F.; Schiess, P. The allure and pitfalls of using LiDAR topography in harvesting and road design. Joint Conference of IUFRO. Forest Operations under Mountainous Conditions and the 12th International Mountain Logging Conference, 2004.
12. Crow, P.; Benham, S.; Devereux, B.; Amable, G. Woodland vegetation and its implications for archaeological survey using LiDAR. *Forestry* **2007**, *80*, 241–252.
13. Hudak, A.T.; Evans, J.S.; Stuart Smith, A.M. LiDAR utility for natural resource managers. *Remote Sensing* **2009**, *1*, 934–951.
14. White, R.A.; Dietterick, B.C.; Mastin, T.; Strohman, R. Forest roads mapped using LiDAR in steep forested terrain. *Remote Sensing* **2010**, *2*, 1120–1141.
15. Clode, S.; Kootsookos, P.J.; Rottensteiner, F. The automatic extraction of roads from LIDAR data. ISPRS 2004, 2004.
16. Clode, S.; Rottensteiner, F.; Kootsookos, P.; Zelniker, E. Detection and vectorization of roads from lidar data. *Photogrammetric Engineering & Remote Sensing* **2007**, *73*, 517–535.
17. Rieger, W.; Kerschner, M.; Reiter, T.; Rottensteiner, F. Roads and buildings from laser scanner data within a forest enterprise. *International Archives of Photogrammetry and Remote Sensing* **1999**, *32*, W14.
18. Mangala, T.R.; Bhirud, S. A new Automatic Road Extraction technique using Gradient Operation and skeletal ray formation. *International Journal of Computer Applications (0975–8887) Volume* **2011**.
19. Doucette, P.; Agouris, P.; Stefanidis, A. Automated road extraction from high resolution multispectral imagery. *Photogrammetric Engineering & Remote Sensing* **2004**, *70*, 1405–1416.
20. Hu, X.; Tao, V. Automatic extraction of main road centerlines from high resolution satellite imagery using hierarchical grouping. *Photogrammetric Engineering and Remote Sensing* **2007**, *73*, 1049.
21. Zhao, H.; Kumagai, J.; Nakagawa, M.; Shibasaki, R. Semi-automatic road extraction from high-resolution satellite image. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences* **2002**, *34*, 406–411.
22. Mena, J.B.; Malpica, J.A. An automatic method for road extraction in rural and semi-urban areas starting from high resolution satellite imagery. *Pattern Recognition Letters* **2005**, *26*, 1201–1220.
23. Cohen, L.D. On active contour models and balloons. *CVGIP: Image understanding* **1991**, *53*, 211–218.
24. Bresson, X.; Esedoglu, S.; Vandergheynst, P.; Thiran, J.P.; Osher, S. Fast global minimization of the active contour/snake model. *Journal of Mathematical Imaging and vision* **2007**, *28*, 151–167.
25. Paragios, N.; Deriche, R. Geodesic active contours and level sets for the detection and tracking of moving objects. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **2000**, *22*, 266–280.

26. Paragios, N.; Deriche, R. Geodesic active regions: A new framework to deal with frame partition problems in computer vision. *Journal of Visual Communication and Image Representation* **2002**, *13*, 249–268.
27. Paragios, N.; Mellina-Gottardo, O.; Ramesh, V. Gradient vector flow fast geometric active contours. Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on. IEEE, 2001, Vol. 1, pp. 67–73.
28. Goldenberg, R.; Kimmel, R.; Rivlin, E.; Rudzsky, M. Fast geodesic active contours. *Image Processing, IEEE Transactions on* **2001**, *10*, 1467–1475.
29. Caselles, V.; Kimmel, R.; Sapiro, G. Geodesic active contours. *International journal of computer vision* **1997**, *22*, 61–79.
30. Chopra, A.; Dandu, B.R. Image Segmentation Using Active Contour Model. *Editorial Board*, p. 819.
31. Bredies, K.; Wolinski, H. An active-contour based algorithm for the automated segmentation of dense yeast populations on transmission microscopy images. *Computing and Visualization in Science* **2011**, *14*, 341–352.
32. Osher, S.; Sethian, J.A. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. *Journal of computational physics* **1988**, *79*, 12–49.
33. Lorigo, L.M.; Faugeras, O.; Grimson, W.E.L.; Keriven, R.; Kikinis, R. Segmentation of bone in clinical knee MRI using texture-based geodesic active contours. In *Medical Image Computing and Computer-Assisted Intervention*; Springer, 1998; pp. 1195–1204.
34. Leventon, M.E.; Grimson, W.E.L.; Faugeras, O. Statistical shape influence in geodesic active contours. Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on. IEEE, 2000, Vol. 1, pp. 316–323.
35. McInerney, T.; Terzopoulos, D. Deformable models in medical image analysis: a survey. *Medical image analysis* **1996**, *1*, 91–108.
36. Seppke, B.; Dreschler-Fischer, L.; Brauer, M. The Use of Active Contours for the Detection of Coastlines in SAR images: A Modular Knowledge-based Framework. *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* **2011**, *3819*, 297–301.
37. Mayer, H.; Laptev, I.; Baumgartner, A. Multi-scale and snakes for automatic road extraction. In *Computer Vision ECCV 98*; Springer, 1998; pp. 720–733.
38. Dell'Acqua, F.; Gamba, P.; Lisini, G. Road extraction aided by adaptive directional filtering and template matching. *Proc. of URBAN* **2005**.
39. Mayer, H.; Laptev, I.; Baumgartner, A.; Steger, C. Automatic road extraction based on multi-scale modeling, context, and snakes. *International Archives of Photogrammetry and Remote Sensing* **1997**, *32*, 106–113.
40. Gruen, A.; Li, H. Semi-automatic linear feature extraction by dynamic programming and LSB-snakes. *Photogrammetric engineering and remote sensing* **1997**, *63*, 985–994.
41. Rheinland-Pfalz. Ministerium für Umwelt, Landwirtschaft, Ernährung, Weinbau und Forsten. JETZT UNSERE CHANCE NUTZEN! - FÜR DEN NATIONALPARK HUNSRÜCK. FÜR

DAS LAND. http://mulewf.rlp.de/fileadmin/mufv/publikationen/Jetzt_unsere_Chance_nutzen_Broschuere_140214.pdf, 2014. retrieved: 2015-15-4.

42. Gauer, J.; Aldinger, E. Waldökologische Naturräume Deutschlands. *Mitteilungen des Vereins für Forstliche Standortskunde und Forstpflanzenzüchtung.* **2005**, *43*.
43. Ebrecht, L.; Schmidt, W. Bedeutung der Bodensamenbank und des Diasporentransports durch Forstmaschinen für die Entwicklung der Vegetation auf Rückegassen. *Forstarchiv* **2008**, *79*, 91–105.
44. Landesamt für Vermessung und Geobasisinformation Rheinland Pfalz. Landesamt für Vermessung und Geobasisinformation Rheinland Pfalz. http://www.lvermgeo.rlp.de/. retrieved: 2015-15-4.
45. Burger, W.; Burge, M.J.; Burge, M.J.; Burge, M.J. *Principles of Digital Image Processing*; Springer, 2009.
46. Roushdy, M. Comparative study of edge detection algorithms applying on the grayscale noisy image using morphological filter. *GVIP journal* **2006**, *6*, 17–23.
47. Verri, A.; Trucco, E. Introductory Techniques for 3-D computer vision, 1998.
48. Gao, W.; Zhang, X.; Yang, L.; Liu, H. An improved Sobel edge detection. Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on. IEEE, 2010, Vol. 5, pp. 67–71.
49. Vincent, O.; Folorunso, O. A descriptive algorithm for sobel image edge detection. Proceedings of Informing Science & IT Education Conference (InSITE), 2009, Vol. 40, pp. 97–107.
50. Canny, J. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **1986**, pp. 679–698.
51. Efford, N. *Digital image processing: a practical introduction using java (with CD-ROM)*; Addison-Wesley Longman Publishing Co., Inc., 2000.
52. Marr, D.; Hildreth, E. Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences* **1980**, *207*, 187–217.
53. Forsyth, D.A.; Ponce, J. Computer Vision A Modern Approach. *Computer Vision: A Modern Approach* **2003**.
54. Gonzalez, R.C.; Woods, R.E. Digital image processing second edition. *Beijing: Publishing House of Electronics Industry* **2002**.
55. Li, C.; Xu, C.; Gui, C.; Fox, M.D. Distance regularized level set evolution and its application to image segmentation. *Image Processing, IEEE Transactions on* **2010**, *19*, 3243–3254.
56. Li, C. Distance Regularized Level Set Evolution (DRLSE). http://www.imagecomputing.org/~cmli/DRLSE/. retrieved: 2015-28-5.
57. Caselles, V.; Catté, F.; Coll, T.; Dibos, F. A geometric model for active contours in image processing. *Numerische mathematik* **1993**, *66*, 1–31.
58. Peng, D.; Merriman, B.; Osher, S.; Zhao, H.; Kang, M. A PDE-based fast local level set method. *Journal of Computational Physics* **1999**, *155*, 410–438.
59. Barles, G.; Soner, H.M.; Souganidis, P.E. Front propagation and phase field theory. *SIAM Journal on Control and Optimization* **1993**, *31*, 439–469.

60. Gomes, J.; Faugeras, O. Reconciling distance functions and level sets. In *Scale-Space Theories in Computer Vision*; Springer, 1999; pp. 70–81.
61. Python Software Foundation. Built-in Functions. https://docs.python.org/2/library/functions.html#map. retrieved: 2015-15-4.
62. Python Software Foundation. pickle - Python object serialization. https://docs.python.org/2/library/pickle.html. retrieved: 2015-15-4.
63. Python Software Foundation. cPickle - A faster pickle. https://docs.python.org/2/library/pickle.html#module-cPickle. retrieved: 2015-15-4.
64. Numpy developers. NumPy. http://www.numpy.org/. retrieved: 2015-15-4.
65. Python Software Foundation. PythonSpeed/PerformanceTips. https://wiki.python.org/moin/PythonSpeed/PerformanceTips, 2014. retrieved: 2015-15-4.
66. Masini, S.; Bientinesi, P. High-performance parallel computations using python as high-level language. Euro-Par 2010 Parallel Processing Workshops. Springer, 2011, pp. 541–548.
67. Huang, S.C.; Cheng, F.C.; Chiu, Y.S. Efficient contrast enhancement using adaptive gamma correction with weighting distribution. *Image Processing, IEEE Transactions on* **2013**, *22*, 1032–1041.
68. Trucco, E.; Verri, A. *Introductory techniques for 3-D computer vision*; Vol. 201, Prentice Hall Englewood Cliffs, 1998.
69. Rudin, L.I.; Osher, S.; Fatemi, E. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena* **1992**, *60*, 259–268.
70. Solem, J.E. *Programming Computer Vision with Python: Tools and algorithms for analyzing images*; " O'Reilly Media, Inc.", 2012.
71. Chambolle, A. Total variation minimization and a class of binary MRF models. Energy minimization methods in computer vision and pattern recognition. Springer, 2005, pp. 136–152.
72. Stark, J.A. Adaptive image contrast enhancement using generalizations of histogram equalization. *Image Processing, IEEE Transactions on* **2000**, *9*, 889–896.
73. Kim, J.Y.; Kim, L.S.; Hwang, S.H. An advanced contrast enhancement using partially overlapped sub-block histogram equalization. *Circuits and Systems for Video Technology, IEEE Transactions on* **2001**, *11*, 475–484.
74. Heipke, C.; Mayer, H.; Wiedemann, C.; Jamet, O. Evaluation of automatic road extraction. *International Archives of Photogrammetry and Remote Sensing* **1997**, *32*, 151–160.
75. Rheinland-Pfalz Ministerium für Landwirtschaft, Weinbau und Forsten. *Empfehlungen Waldwegebau, Teil 2: Empfehlungen zur Feinerschließung der Bestände im Staatswald des Landes Rheinland-Pfalz*; 2003.

Declaration of Authorship

With this statement I, Thomas Rommel, matriculation number 1103878, declare, that I have independently completed the master thesis entitled with:

"Automatic Detection of Line Structures in Airborne LiDAR Data"

The thoughts taken directly or indirectly from external sources are properly marked as such. This thesis was not previously submitted to another academic institution and has also not yet been published.

_____

Thomas Rommel, Trier, July 27, 2015