Royal Military Academy
173rd Promotion Polytechnics
Lieutenant Aviator Jan Olieslagers

Generation of a Road Vector Layer

SE511: Strategic Military Sensors

Officer-Student:
Beerten Pieter

Department CISS
Under supervision of Lt. Boland O.

Academic Year 2022-2023

Contents

List of Figures **iv**

1 Introduction **1**

2 Creation of the satellite images **2**
2.1 Generation of the Sentinel-2 images 2
2.1.1 Definition of the Area-of-Interest 2
2.1.2 Choice of the image date 3
2.1.3 Sentinel-2 Image 3
2.2 Generation of the Sentinel-1 images 4

3 Exploitation of road detection algorithms **6**
3.1 Generation of the ground truth 6
3.2 Preprocessing of the satellite images 7
3.2.1 Top-hat filtering 7
3.2.2 Unsharp masking 8
3.3 Contour Finding 9
3.3.1 Finding a value for c 9
3.3.2 Loop detection 9
3.3.3 Results for the Sentinel-2 NIR and Sentinel-1 images 10
3.3.4 Comparison with the ground truth 11
3.4 Edge Detection 13
3.4.1 Theoretical description of the edge detection algorithms 13
3.4.2 Edge detection on the Sentinel-2 RGB image 15
3.4.3 Edge detection on the Sentinel-2 NIR image 24
3.4.4 Edge detection on the Sentinel-1 VV image 25
3.4.5 Edge detection based on the Canny filter 26
3.5 Ridge Operators 30
3.5.1 Results for the Sentinel-2 RGB Image 30
3.5.2 Results for the Sentinel-2 NIR Image 32
3.5.3 Comparison with the ground truth 35

4 Creation of the algorithm **36**

5 Validation of the algorithm on another AOI **38**
5.1 AOI with one main road 38
5.2 AOI with water alongside the road 40

6 Conclusion **42**

A Evaluation of the Edge Detection algorithms **43**

B Contour Finding **45**

C Validation **48**

D Code **50**

References **56**

List of Figures

1.1 Example of available Terrascope imagery 1
2.1 Area-of-Interest in Google Earth 3
2.2 Sentinel-2 Images 4
2.3 Sentinel-1 Images 5
3.1 Ground truth 6
3.2 White top-hat filtering 7
3.3 Unsharp masking 8
3.4 Loop detection for the S2 RGB image 10
3.5 Loop detection for the S2 NIR image 10
3.6 Loop detection for the S1 VV image 11
3.7 Contour Finding compared to Ground Truth for S2 RGB 12
3.8 Contour Finding compared to Ground Truth for S2 NIR 12
3.9 Edge operators for the S2 RGB image 15
3.10 Reversed edges for the S2 RGB image 16
3.11 Skeletonized edges for the S2 RGB image 17
3.12 Number of detected noise points for the S2 RGB image 19
3.13 Skeletonized edges for the S2 RGB image after filtering 20
3.14 Number of detected noise points after filtering for the S2 RGB image . . . 21
3.15 Edge detection compared to Ground Truth for S2 RGB 22
3.16 Optimal pixel threshold value for the Sobel detector 23
3.17 Edge operators for the S2 NIR image 24
3.18 Skeletonized edges for the S2 NIR image after filtering 24
3.19 Edge operators for the S1 VV image 25
3.20 Skeletonized edges for the S1 VV image after filtering 25
3.21 Edge operators for the Gaussian filtered S1 VV image 26
3.22 Edge operators for the Gaussian filtered S1 VV image 26
3.23 Determining the best value of σ for the S2 RGB image 27
3.24 Determining the best value of t_{low} and t_{high} for the S2 RGB image 28
3.25 Canny filter compared to Ground Truth for S2 RGB 28
3.26 Canny filter compared to S2 RGB 29
3.27 Ridge Filters applied to S2 RGB Image 30
3.28 Influence of the scale range for the S2 RGB Image 31
3.29 Influence of the scale range for the S2 RGB Image 32
3.30 Ridge Filters applied to S2 RGB Image 33
3.31 Influence of the scale range for the S2 NIR Image 34
3.32 Influence of the scale range for the S2 RGB Image 34
3.33 Edge detection compared to Ground Truth for S2 RGB 35
4.1 Block scheme of the road detection algorithm 36
4.2 Output of the algorithm for the considered 'test' AOI 37
5.1 Result on AOI with one main road 38

5.2 Result on AOI with one main road 39
5.3 Result on AOI with one main road and water areas 40
5.4 Result of contour finding on AOI with one main road and water areas . . . 41
A.1 Optimal pixel threshold value for the Roberts detector 43
A.2 Optimal pixel threshold value for the Scharr detector 43
A.3 Optimal pixel threshold value for the Prewitt detector 44
B.1 Finding c for the S2 RGB image 45
B.2 Finding c for the S2 NIR image 46
B.3 Finding c for the S1 VV image 47
C.1 Finding c for the first S2 RGB validation image 48
C.2 Finding c for the second S2 RGB validation image 49

1 Introduction

In this report, an algorithm will be designed that is able to detect and extract the roads present in satellite images. This extracted Road Layer, which is in fact a Geographic Information System, can then be used in tools like Google Maps or navigation applications such as Waze to accurately model the roads in an area. All of this helps them in offering more precise navigation aid, which has nowadays become an integral part of our lives in this globalized world.

The report is divided in multiple sections. In a first section, the Sentinel-1 and Sentinel-2 images will be created for a certain specified Area-of-Interest. In a second part, several detection algorithms will be exploited and tested on the images of this area, in order to create a precise prediction model. In a final section, the proposed models will be tested on a new area and possible shortcomings of the model will be discussed.

The Sentinel-1 and Sentinel-2 images will be created and treated by using the free open-source service *Terrascope*, which offers imagery of several European satellites.

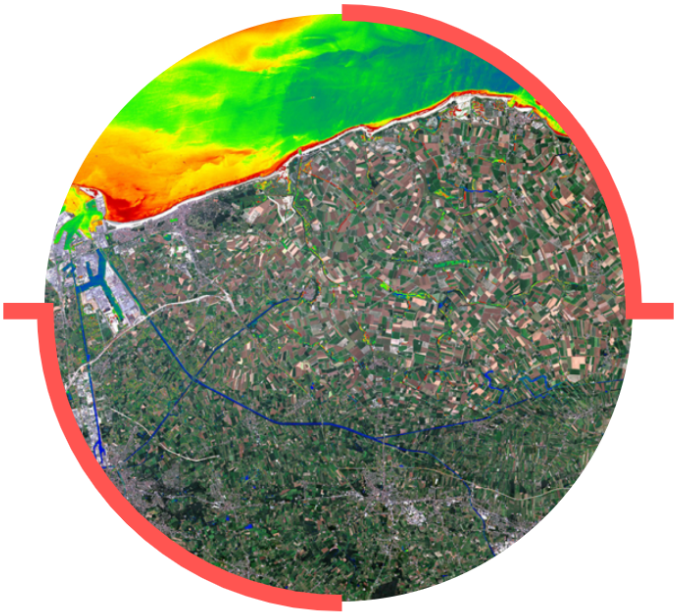

Figure 1.1: Example of available Terrascope imagery

2 Creation of the satellite images

In this section, both the Sentinel-1 and the Sentinel-2 image will be created by using *Terrascope*. In annex D of this report, only the most pertinent lines of code will be given. For further details, the reader is referred to the Jupyter notebook[1].

2.1 Generation of the Sentinel-2 images

The Sentinel-2 (S2) constellation consists of two polar-orbiting satellites that are placed in the same sun-synchronous orbit, but at a phase difference of 180°. The aim of this constellation is to model the changes of the Earth's surface. In order to do so, the satellites have a high swath width of about 290 kilometers, and the constellation has a high revisit time of 5 days[2] at the equator.

Because roads are often only several meters wide, a high spatial resolution is required to be able to clearly distinguish the roads in an S2 image. Therefore, only the high resolution images with a resolution of 10 meters will be considered. Moreover, as a second constraint, the images should contain as few clouds as possible in order to have a clear and workable image.

2.1.1 Definition of the Area-of-Interest

A zone with a clear separation between roads and vegetation or other artefacts like buildings is required to develop the algorithm. Therefore, the following area is considered for the development and testing of the algorithm:

- Lower left limit: (4.4480 °, 50.7946 °)
- Upper right limit: (4.4840 °, 50.8200 °)

This zone is represented as a Google Earth image in figure 2.1.

1 *SE511_RVL.ipynb*; available on GitHub: https://github.com/PB173/Road-Vector-Layer.git
2 Revisit time at the equator of 5 days when both satellites are in use and under cloud-free conditions.

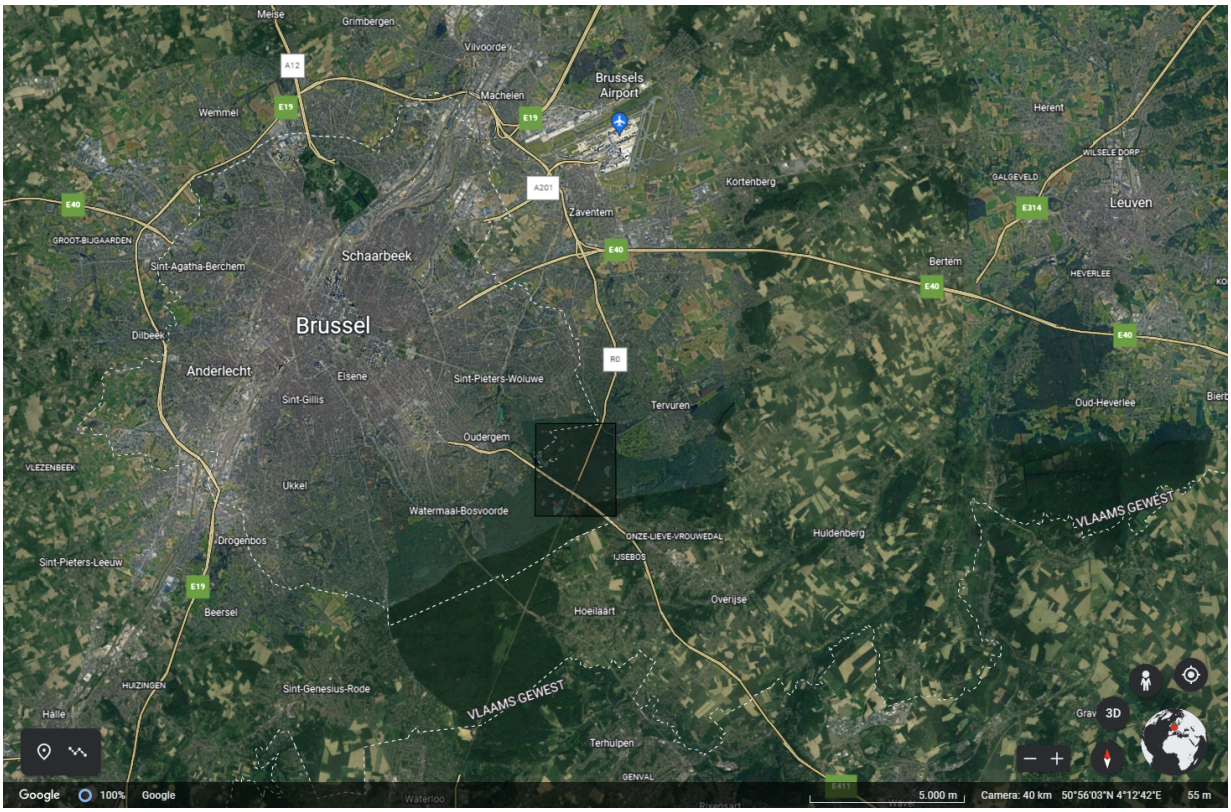

Figure 2.1: Area-of-Interest in Google Earth

It is clear that this zone contains 2 major sets of crossing roads in a wooded area. Moreover, there is a smaller road running from the top left to the top right part of the area. Finally, a curved road is present. These are indeed ideal circumstances to develop a road detection and extraction tool.

2.1.2 Choice of the image date

As said before, the Sentinel-2 image cannot contain too many clouds, as this would have a negative effect on the performance of the algorithm. Luckily, Terrascope has developed a tool to automatically list all dates on which an image with a limited cloud coverage was taken. By going through this list, July 31, 2020 was finally retained as date.

2.1.3 Sentinel-2 Image

Finally, the RGB and Near-Infrared image are shown in figure 2.2. From the cloudmask, it is indeed confirmed that these images contain nearly no clouds whatsoever .

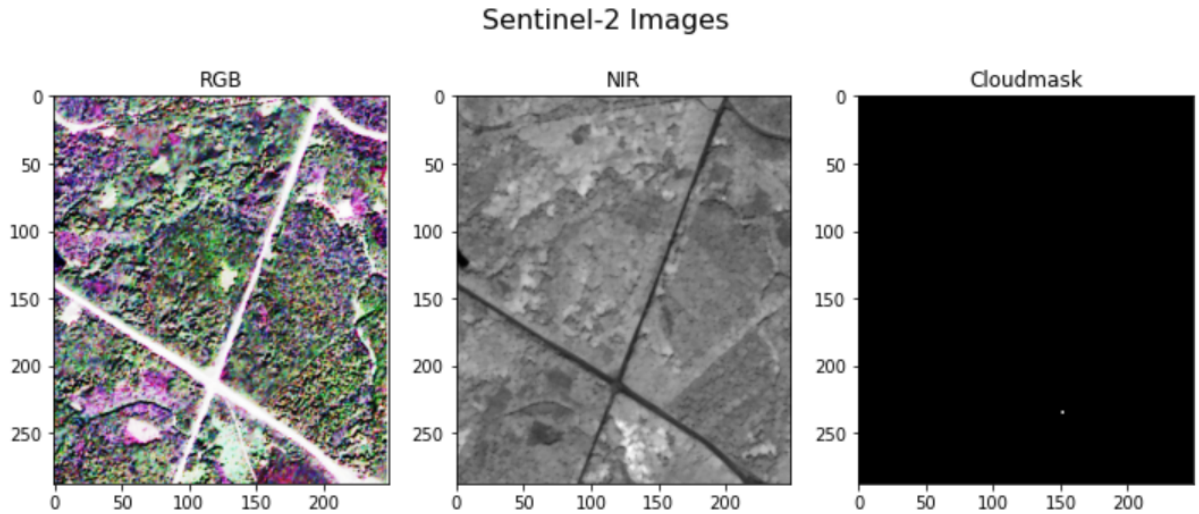

Figure 2.2: Sentinel-2 Images

Note that the top-hat transform has been applied to the NIR image, in order to extract and remove small elements from the image. This yields a much clearer image. More details about this process can be found in section 3.2.1.

2.2 Generation of the Sentinel-1 images

The Sentinel-1 constellation also consists of two polar-orbiting satellites that create synthetic aperture radar images. These quality of these so-called SAR images is independent of the weather conditions.

For this project, only the *Ground Range Detected* (GRD) σ_0 images are considered. These GRD images are derived from the *Single Look Complex* (SLC) images and consist of focused SAR observations that have been detected, multi-looked, and projected to ground range using an Earth ellipsoid model. Because of the multi-look processing, the resulting images suffer less speckle.

Moreover, the GRD σ_0 data are subject to several additional processing steps. Orbital correction, radiometric and thermal noise removal and radiometric calibration are applied to the data, so that it actually contains the radar backscatter intensity.
More information on how Terrascope preprocesses the Sentinel-1 images can be found in [1].

The Sentinel-1 image of course needs to represent the same Area-of-Interest as the Sentinel-2 image, which is defined in section 2.1.1. Also, the S1 and S2 images should be close in time. As roads and vegetation generally do not change within the hour, a time difference of a few days is acceptable.

The resulting images are shown in figure 2.3. The time difference between the S1 and S2 images is only 4 hours 48 minutes.

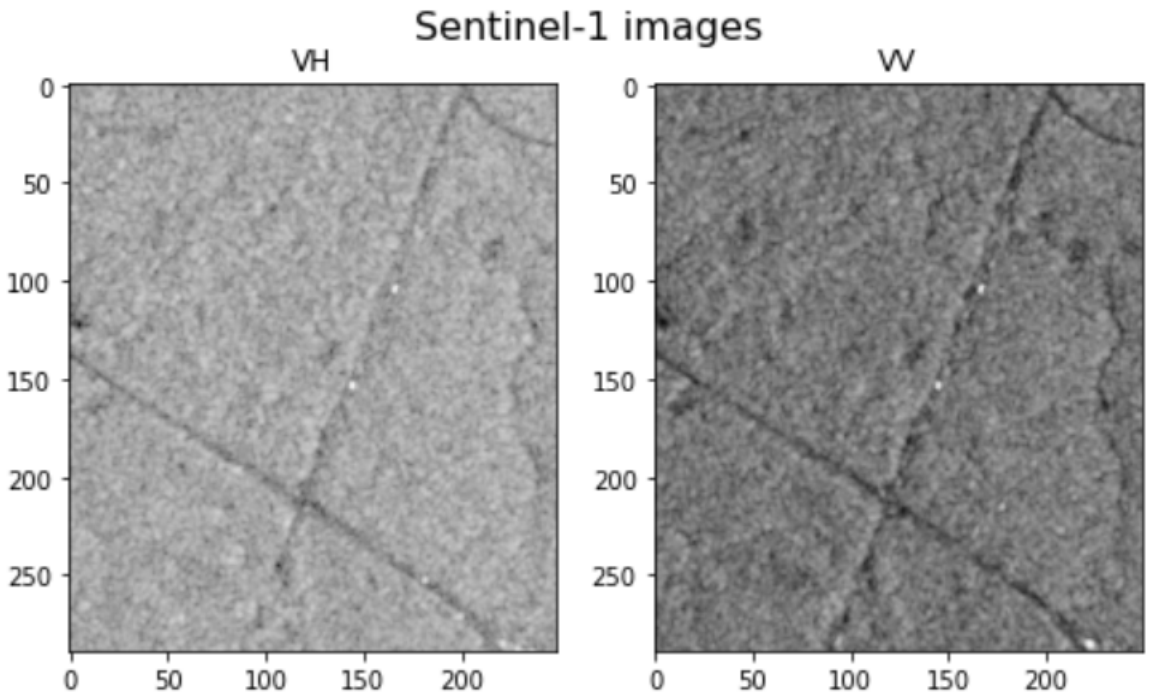

Figure 2.3: Sentinel-1 Images

One immediately notices the reduced visual contrast between the main roads and the vegetation around them. For the remainder of this report, only the *Vertical-Vertical (VV)* image will be fed to the algorithm.

Note that both the *Vertical-Vertical* and the *Vertical-Horizontal* polarised image are displayed in decibels. This generally provides a higher visual contrast, which makes the image more apt to human observers. The conversion to decibels is easily done by applying equation 1:

$$\sigma_{0,dB} = 10 \cdot \log_{10}(\sigma_0) \tag{1}$$

3 Exploitation of road detection algorithms

In this section, several detection algorithms from the *scikit-image* library [2] will be exploited and tested on the images generated in section 2. As well edge detection as contour detection algorithms and ridge operators are tested and discussed.

3.1 Generation of the ground truth

Before the quality and accuracy of the proposed algorithms can be assessed, a ground truth needs to be generated, to which the output of the different algorithms will be compared. In order to do this, a binary data structure is created, based on the S2 image. After converting the RGB image to greyscale, every pixel with a value below or equal to 0.95 is put to 0, while all other pixels are set to 1. The results of this binary mapping are shown in figure 3.1.

On the leftmost figure, it is clear that this approach does not eliminate all noise, which results in some falsely detected road pixels. However, the most interesting areas, being the curved road at the top right of the image and the intersection of the two main roads, are detected correctly. These two regions will be used to assess the different algorithms.

Note that the smaller roads that run through this AOI are not (correctly) represented in the ground truth. Unfortunately, the resolution of the Sentinel-1 and Sentinel-2 images is not good enough to detect these. The assessment of the different algorithms will thus only take the larger sets of roads into account.

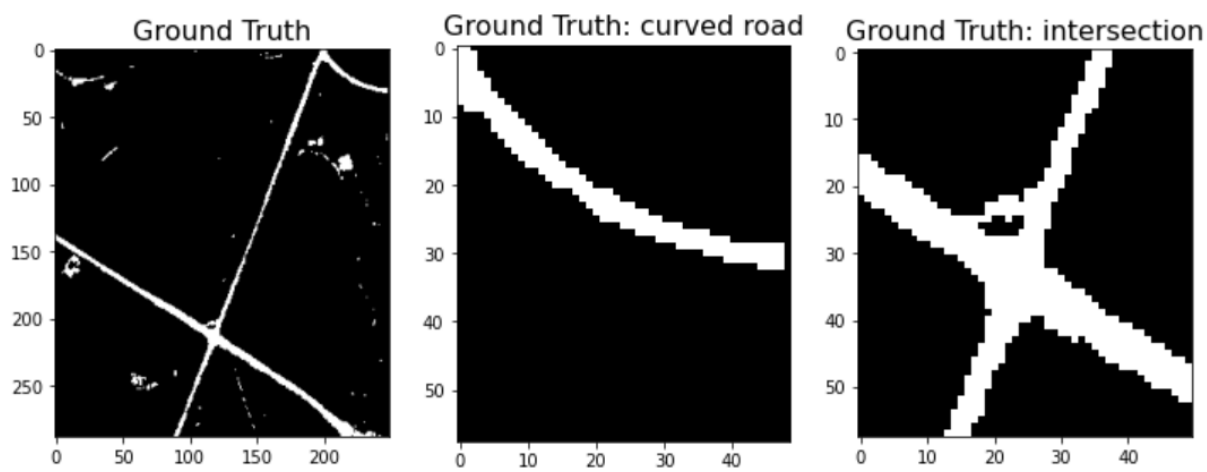

Figure 3.1: Ground truth

3.2 Preprocessing of the satellite images

As a final step before applying the different detection algorithms, the acquired satellite images will be enhanced. This is done in two different steps.

3.2.1 Top-hat filtering

As explained in section 2.1.3, applying a top-hat filter removes small objects from the image. In this report, a white top-hat filter is used, which extracts the bright spots that are smaller than the structuring element [3]. For this application, the radius is set to 1. The results are shown in figure 3.2.

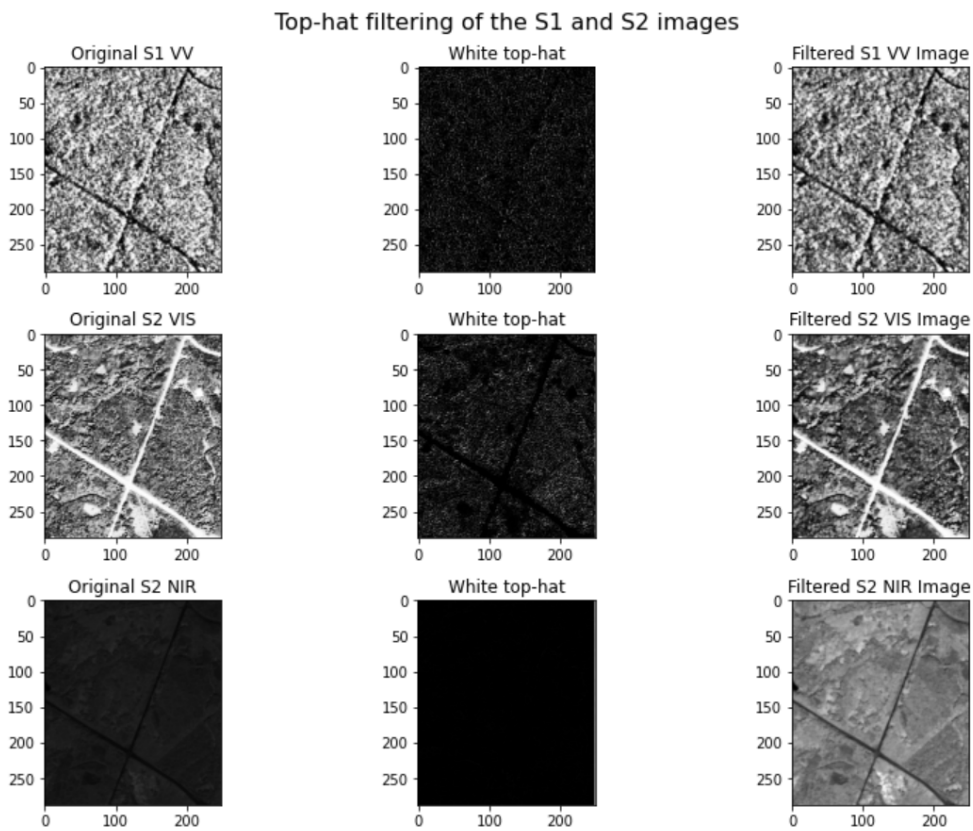

Figure 3.2: White top-hat filtering

Especially for the Sentinel-2 RGB image, the filter removes a lot of the noise around the roads. Moreover, the Sentinel-2 NIR image has become a lot brighter, displaying much more details for the human observer.

3.2.2 Unsharp masking

Secondly, *unsharp masking* is applied to the different images. This image enhancement technique identifies sharp details as a difference between the original image and a blurred version of the same image. The detected details are then amplified and re-added to the original image, yielding stronger edges [4]. This technique should increase the contrast between the roads and the vegetation alongside, which facilitates the detection of the roads.

For this application, the radius for the blurring is set to 1 and the amplification factor to 1.5. The results are shown in figure 3.3.

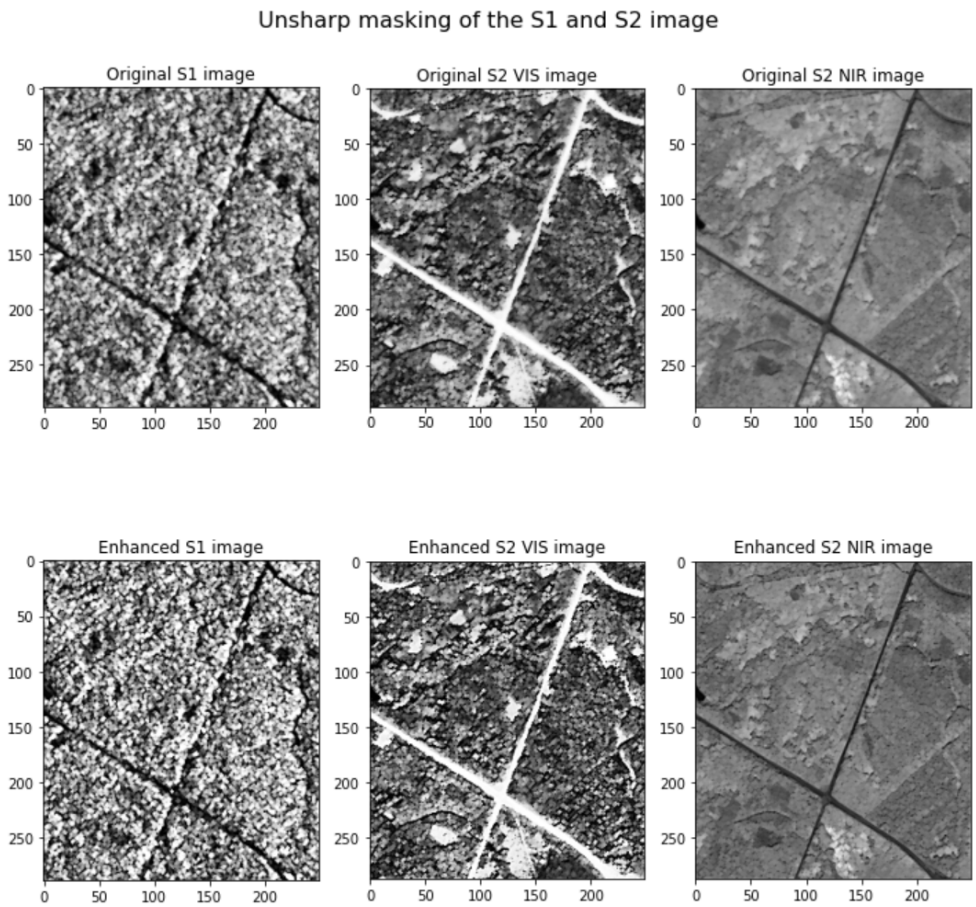

Figure 3.3: Unsharp masking

Especially for the Sentinel-2 NIR image does this filter yield significantly sharper visual edges.

3.3 Contour Finding

The first detection algorithm that will be tested on the enhanced satellite images is the so-called *contour finding*. This algorithm uses a marching squares method to find constant value contours in images [5]. Since roads are mostly all made of the same material, one can easily detect them, as long as the correct constant value is sought for.

Remark: Note that after several trails, it was found that for the Sentinel-2 RGB image, the algorithm performs best without the *unsharp masking*. T herefore, this filtering technique will only be performed on the S2 NIR and the S1 image.

3.3.1 Finding a value for c

In order to know which constant value c the algorithm should look for, several values of c between 0 and 1 are tried. This is displayed in figure B.1 in Appendix B, where the contour finding algorithm is applied to the Sentinel-2 RGB image for 12 different values of c.

It is clear that for too low values of c, the algorithm only finds contours in the woods. However, as from $c \approx 0.80$, the roads are detected quite well. A value of $c = 0.90$ is retained to further exploit this detection technique.

3.3.2 Loop detection

As can be seen in figure B.1, the algorithm often finds small loops in the woods. Of course, these areas do not represent roads and need to be filtered out. Therefore, a loop detecting algorithm is used, that checks whether the detected area is a loop.

An important remark to make here is that the algorithm will always find loops, except when the detected areas intersect with the image edges. To avoid that all detected areas are discarded, a second criterion is verified before deciding on whether a detected area can be classified as a road. The algorithm will, for all detected areas, check whether the ratio of the width and the height (or the dimensions along the principal axis and the axis perpendicular to this one) is larger than a certain user-specified threshold. A high ratio means that the area is very slender, which is an indication that the detected area could in fact be a road.

Figure 3.4 shows the results for the S2 RGB image, both for no loop detection as for loop detection performed with a threshold ratio of respectively 1 and 20. It is clear that by tweaking this parameter, very accurate results can be found. Note however the inaccurate modelling of the purple and brown main roads, where bright spots int he image, next to the road, are incorrectly detected as being part of the road. These bright spots in reality correspond to open spots in the wooded area.

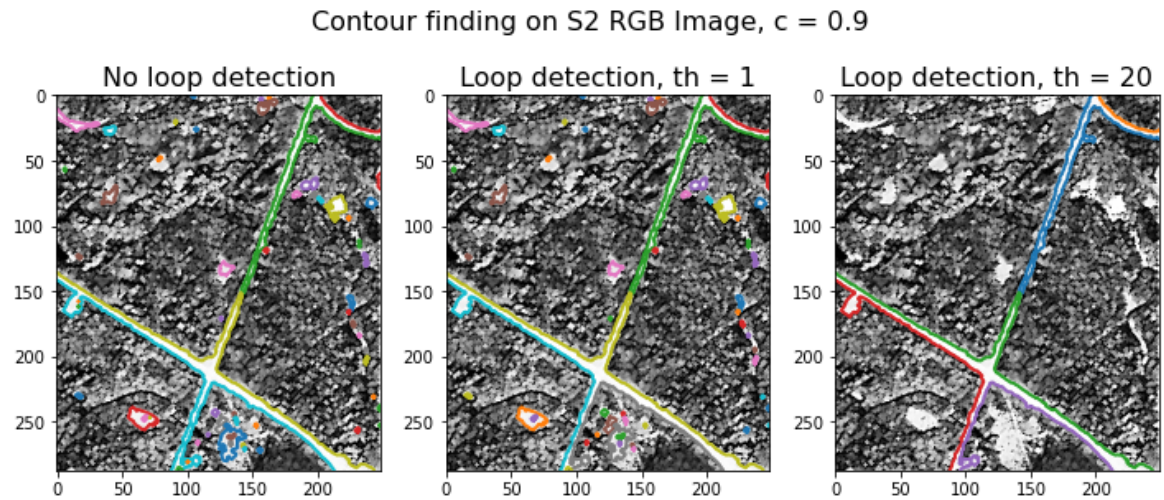

Figure 3.4: Loop detection for the S2 RGB image

3.3.3 Results for the Sentinel-2 NIR and Sentinel-1 images

The same algorithm, with and without loop detection, will now be applied to the Sentinel-2 NIR image. As is clear from figure B.2 in Appendix B, the algorithm can only find contours at rather low constant values. Therefore, a value of $c = 0.07$ is chosen.

In figure 3.5, the results of the algorithm, both with and without loop detection, are shown. The algorithm still performs reasonably accurate. The curved road on the top right of the image is detected with more noise, but now the purple and brown main roads are detected without the large bright spots, as opposed to the S2 RGB image.

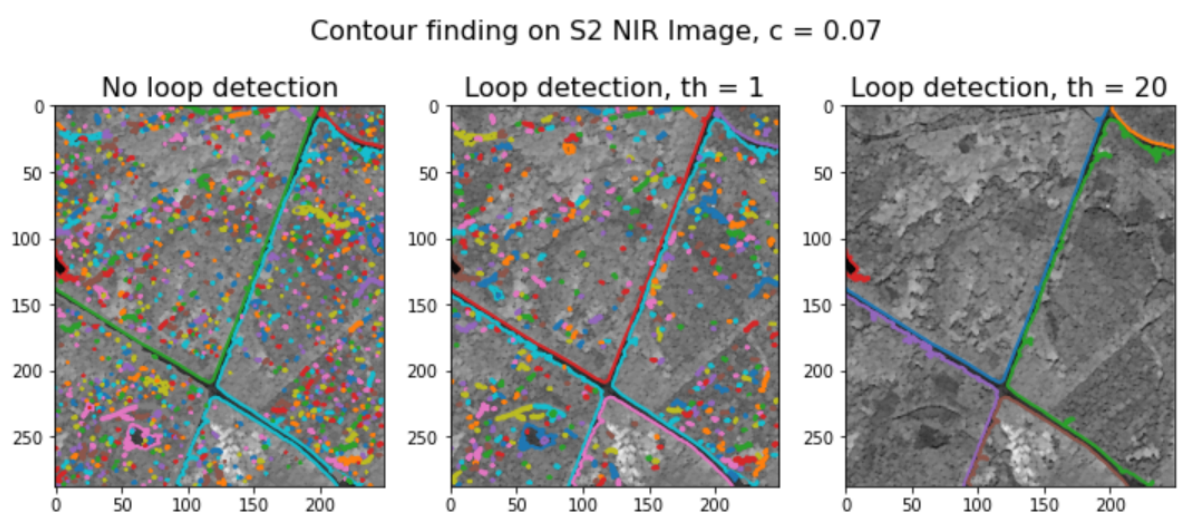

Figure 3.5: Loop detection for the S2 NIR image

Finally, the algorithm is applied to the Sentinel-1 VV polarized image. Figure B.3 in Appendix B shows that the results for any value of c are far from ideal. The algorithm detects a lot of noise points for all values of c, while the actual roads are inaccurately detected, as can be clearly seen for a value of $c = 0.18$ for instance.

Finally, the results for $c = 0.21$ are given in figure 3.6. As said before, only parts of the road are detected, and the parts that are detected suffer from a lot of noise.

In conclusion, this algorithm works well for Sentinel-2 images, but not quite for the Sentinel-1 images. Therefore, only the Sentinel-2 images will be further considered in the analysis of this algorithm.

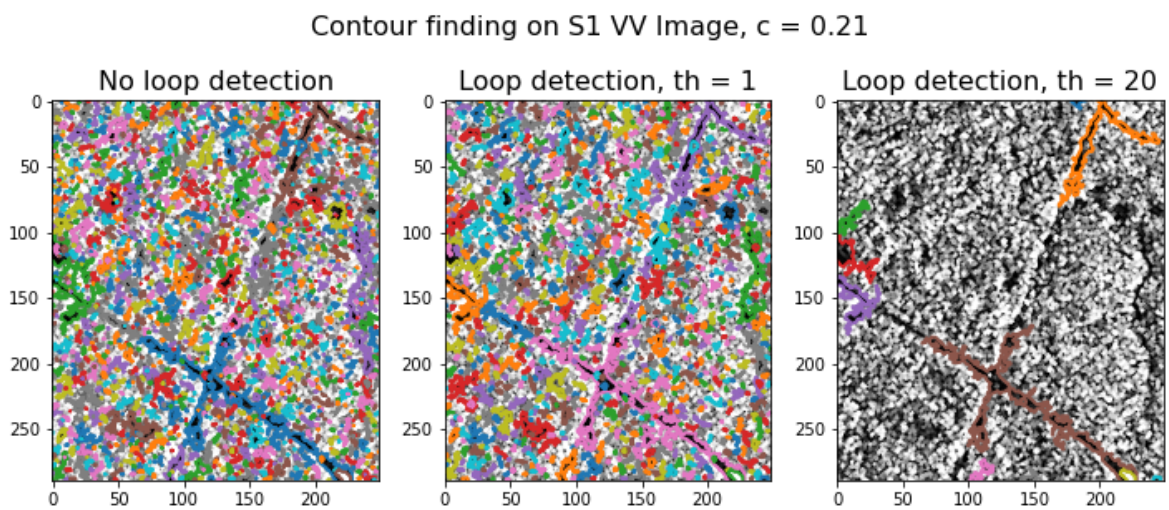

Figure 3.6: Loop detection for the S1 VV image

3.3.4 Comparison with the ground truth

To wrap up the discussion about the contour finding algorithm, the results for the Sentinel-2 images will be compared to the ground truth established in section 3.1.
In figure 3.7, the contours found for the Sentinel-2 RGB image are plotted on top of the ground truth. It can be seen that the found contours, though not perfectly coinciding with the ground truth, follow the general outline of the roads quite well.

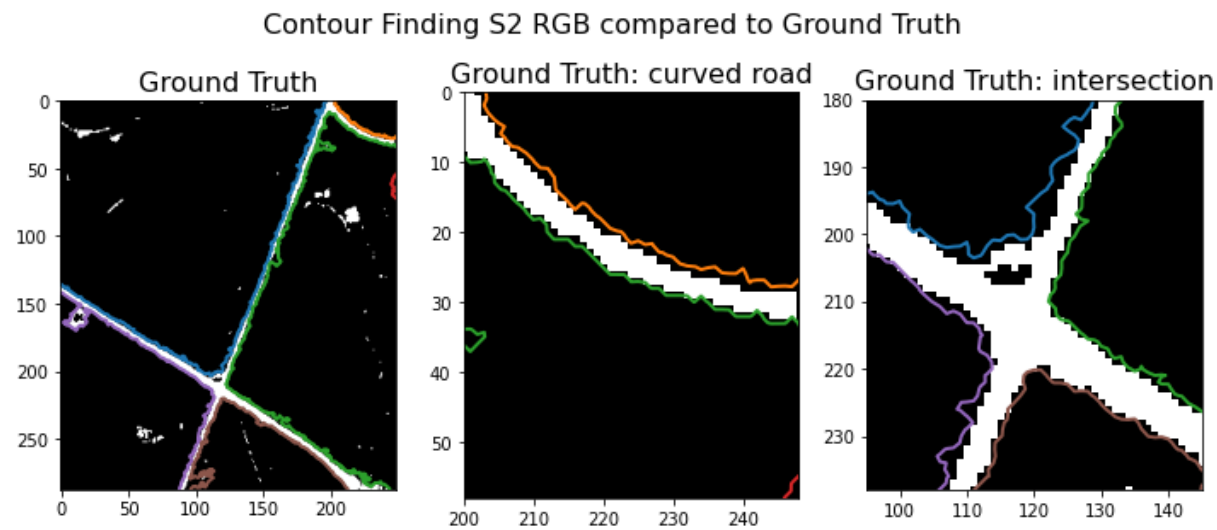

Figure 3.7: Contour Finding compared to Ground Truth for S2 RGB

In figure 3.8, the contours found for the Sentinel-2 NIR image are compared to the ground truth. One can see that the curved road is not very accurately detected. However, the general outline of the roads is still properly found.

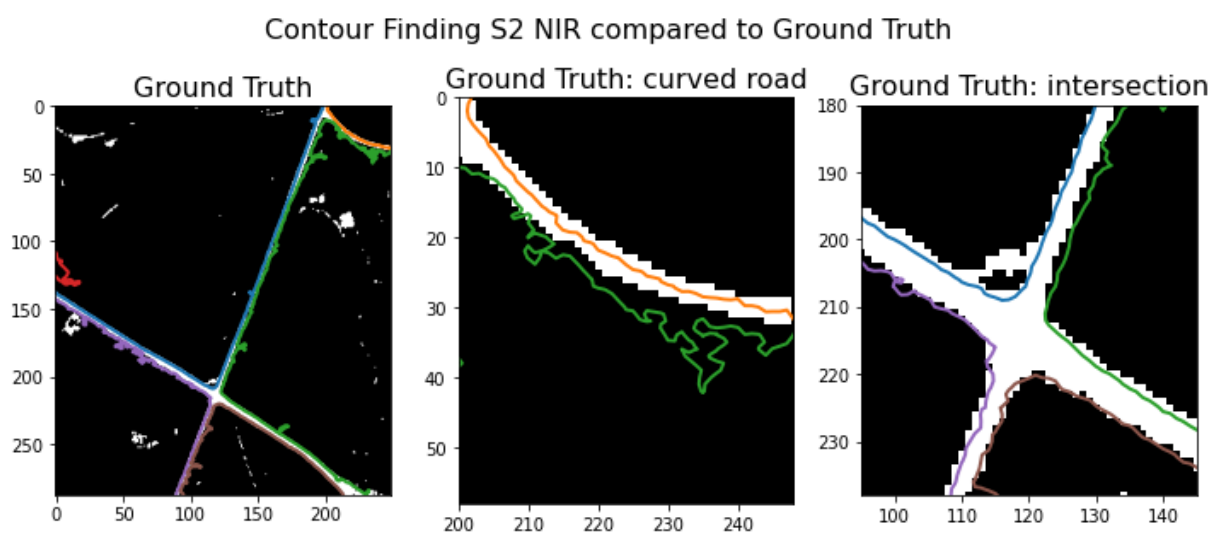

Figure 3.8: Contour Finding compared to Ground Truth for S2 NIR

Note however that, despite the reasonably accurate results for the Sentinel-2 images, it will be hard to actually use this algorithm to create a Road Vector Layer. This algorithm in fact detects the contours of the roads, instead of the middle, which is much more practical when creating a GIS. In practice, this algorithm will thus not be preferred to extract roads from a satellite image. In the following sections, other road detection algorithms will be exploited that are capable of detecting the middle of the roads.

3.4 Edge Detection

Next, several edge detection algorithms will be tested. All of the following algorithms use discrete differentiation to compute an approximation of the image intensity gradient. The difference between the algorithms lays in the way this gradient is approximated [6].

First, a short theoretical description of each algorithm will be given. Later, the results will be discussed for both Sentinel-2 images and the Sentinel-1 VV image. In this section, the edge detection algorithms of Roberts, Sobel, Scharr and Prewitt will be exploited.

3.4.1 Theoretical description of the edge detection algorithms

As said in the introduction of this section, all edge detection algorithms are based on a discrete differentiation in order to approximate the image intensity gradient. However, there are some (minor) differences in the way this approximation is executed. Thsese differences will be explained in this section.

3.4.1.1 Roberts edge operator

The Roberts edge operator [7], often referred to as the Roberts cross, approximates the gradient by computing the sum of the squared differences between diagonally adjacent pixels. To achieve this, Roberts proposed the following equations:

$$\begin{cases} y_{i,j} = \sqrt{x_{i,j}} \\ z_{i,j} = \sqrt{(y_{i,j} - y_{i+1,j+1})^2 + (y_{i+1,j} - y_{i,j+1})^2} \end{cases} \tag{2}$$

Here, x represents the initial intensity. z is the computed derivative. These formulas are then translated to a convolution of the image with the following two 2×2 kernels, one for the horizontal and one for the vertical direction:

$$\begin{cases} K_1 = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} \\ K_2 = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix} \end{cases} \tag{3}$$

The gradient for every point $I(x, y)$ in the image is then computed as follows:

$$\nabla I(x, y) = \sqrt{G_x^2 + G_y^2} \tag{4}$$

where $G_x(x, y)$ represents a point in the image created by convolving the original image with K_1 and $G_y(x, y)$ represents a point in the image created by a convolution with K_2.

At each image point, this gradient is a 2D-vector filled with the derivatives in both the horizontal and vertical directions. By definition, it points in the direction of the largest

intensity increase. This means that, for a certain zone of constant intensity, the gradient corresponds to a 1×2 zero vector. For an edge, the gradient points towards the side with the highest intensity. By exploiting this basic property of the gradient, the different algorithms detect and localize edges in the image.

In the early days of computer vision, the Roberts cross method was favored due to its simplicity. By using smaller kernels that only contain integers, the computational cost of this method is much lower than that of its counterparts. The price to pay however is an increased sensitivity to noise. Nowadays, computational power has significantly increased, which makes the other operators described in section 3.4.1.2 more attractive.

3.4.1.2 Prewitt, Sobel and Scharr edge operators

These operators work in a similar way to the Roberts cross, except by using a larger 3×3 kernel instead of the 2×2 kernel used by Roberts. This yields more accurate results, however at a higher computational cost. The difference between the algorithms lays in the kernels used to approximate the gradient [8, 9].

The Prewitt operator uses the following kernels:

$$\begin{cases} K_1 = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} \\ K_2 = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \end{cases} \tag{5}$$

The gradient is calculated according to the same formula as the Roberts cross, given in equation 4.

The Sobel operator uses the following kernels:

$$\begin{cases} K_1 = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \\ K_2 = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \end{cases} \tag{6}$$

Finally, the kernels for the Scharr operator are the result of a minimization of the weighted mean squared angular error in the Fourier domain. The resulting kernels are given in equation 7:

$$
\begin{cases}
K_1 = \begin{bmatrix} +47 & 0 & -47 \\ +162 & 0 & -162 \\ +47 & 0 & -47 \end{bmatrix} \\
K_2 = \begin{bmatrix} +47 & +162 & +47 \\ 0 & 0 & 0 \\ -47 & -162 & -47 \end{bmatrix}
\end{cases}
\tag{7}
$$

3.4.2 Edge detection on the Sentinel-2 RGB image

3.4.2.1 Results for the Sentinel-2 RGB image

In figure 3.9, the results for the different edge detection operators are shown. All operators are applied to the enhanced Sentinel-2 RGB image, as explained in section 3.2.

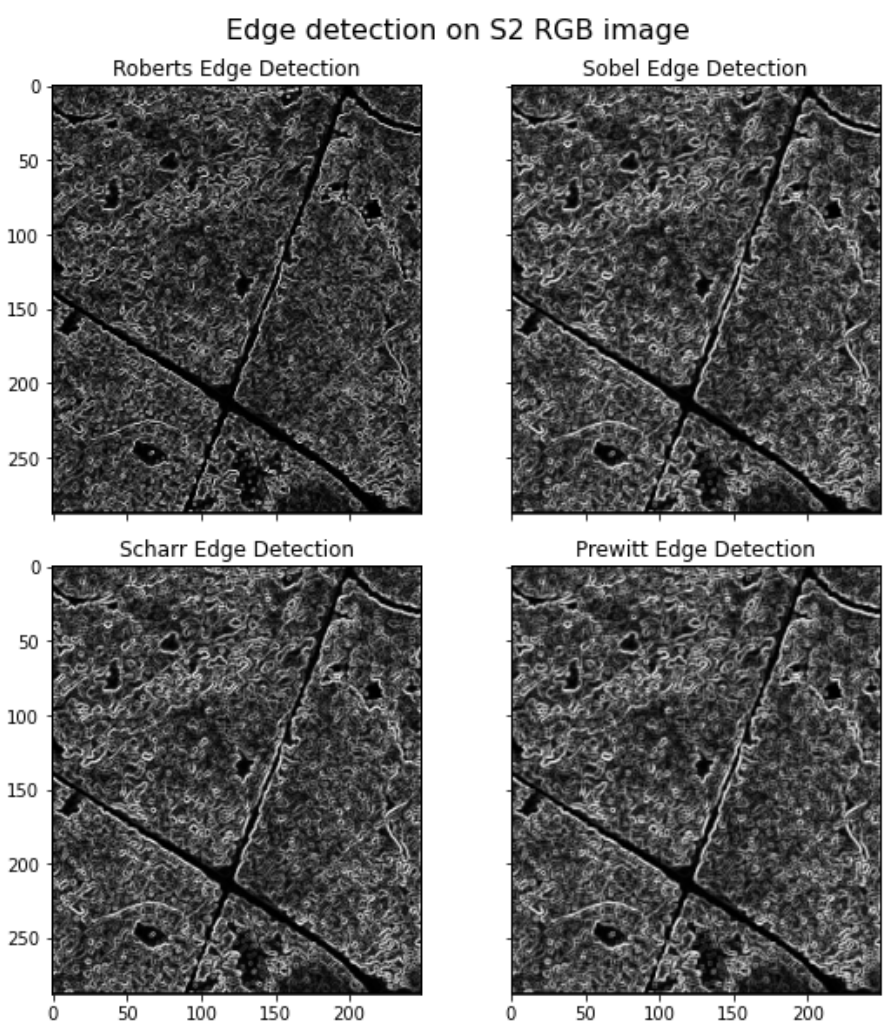

Figure 3.9: Edge operators for the S2 RGB image

It may be clear that all four of the algorithms detect a lot of edges, both those belonging to the roads as well as edges of the vegetation. Keeping the final remark in section 3.3.4 in mind, it is not ideal to highlight the edges of the road instead of the roads themselves. Therefore, a pixel reversing operation will be applied, where all pixel values below a certain threshold are set to 1 and all pixel values above this threshold are set to 0. For the Robert and Sobel edge detectors, the threshold is set to 0.0252. For the two other edge detectors, the threshold is set to 0.0273. How this threshold was chosen is explained later in section 3.4.2.4.

As can be seen in figure 3.10, this operation will highlight the roads.

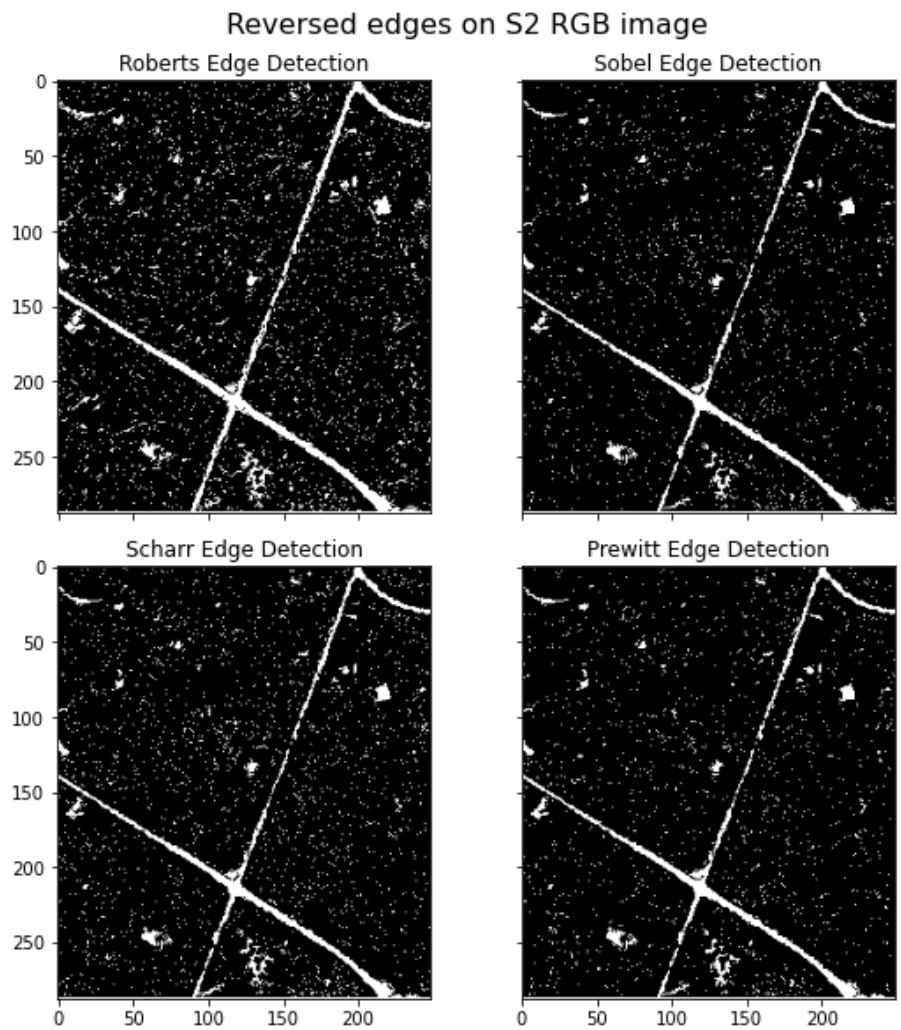

Figure 3.10: Reversed edges for the S2 RGB image

Despite the highlighted roads, these binary images are still not suited to extract a Road

Vector Layer. The roads are several pixels wide, which would lead to sloppy results when extracting and converting them to a Road Vector Layer. To solve this, skeletonization can be applied. This process reduces objects to 1 pixel wide representations by looping over the image multiple times. At every pass, the border pixels of detected objects are removed, yielding only very thin edges. Of course, this operation cannot break the continuity of an object's edge [10].

The results are given in figure 3.11.

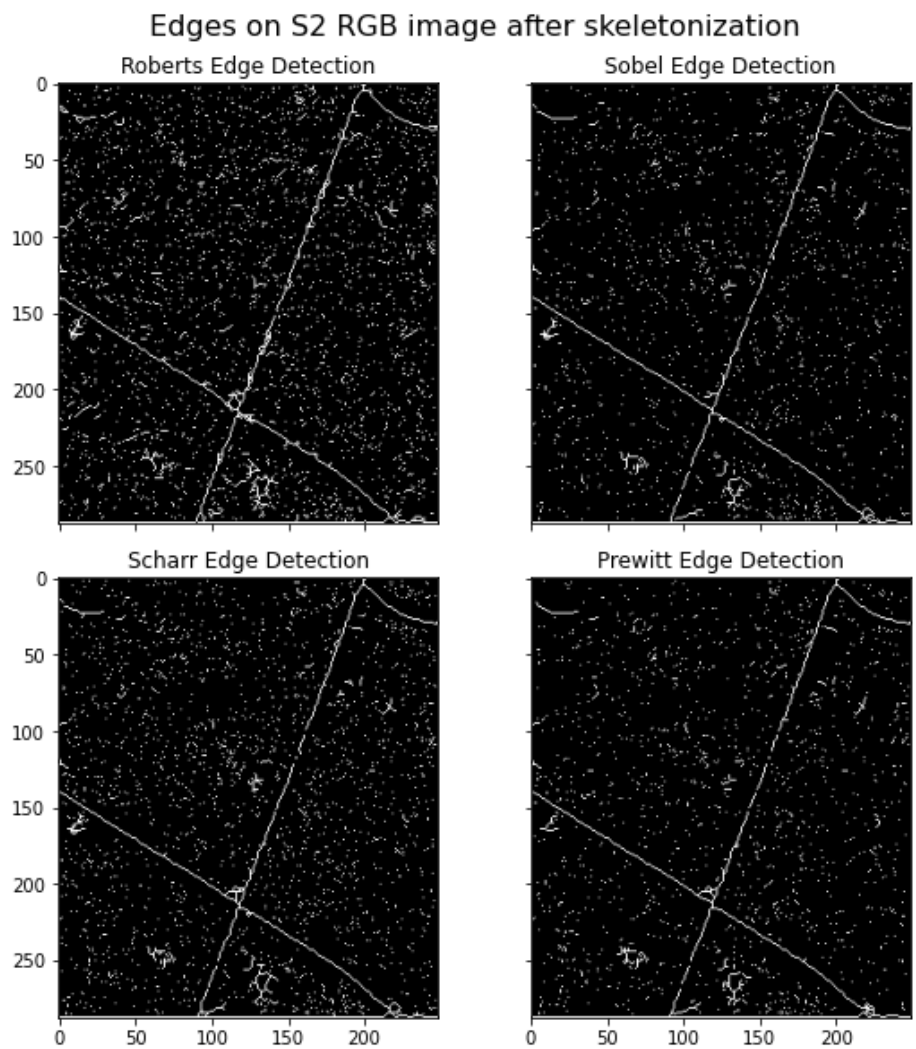

Figure 3.11: Skeletonized edges for the S2 RGB image

All four detection algorithms now clearly display the roads present in the area. However, all of them suffer from a lot of noise. The quantization and reduction of this noise will be the topic of the following section.

3.4.2.2 Noise quantization and reduction

Before one can determine the best edge detection algorithm, one has to define how the quality of the algorithms will be assessed. In this report, the different edge detectors will be ranked according to the number of 'false positives'; the pixels that are falsely detected as being part of a road. To do this, the detected pixels are compared to the ground truth. Pixels that are detected by the algorithm but that do not belong to the ground truth will be classified as noise point.

The resulting noise assessment is displayed as a bar chart in figure 3.12. The numerical values are given in table 1. It is clear that the Roberts edge detection algorithm results in most noise point in the designated area. As explained in section 3.4.1.1, this can be expected due to its smaller kernel. Furthermore, one can conclude that there is no significant difference between the Sobel and Prewitt algorithms for this area of interest.

Method	Number of noise points
Roberts	4220
Sobel	2741
Scharr	3340
Prewitt	2714

Table 1: Number of detected noise points

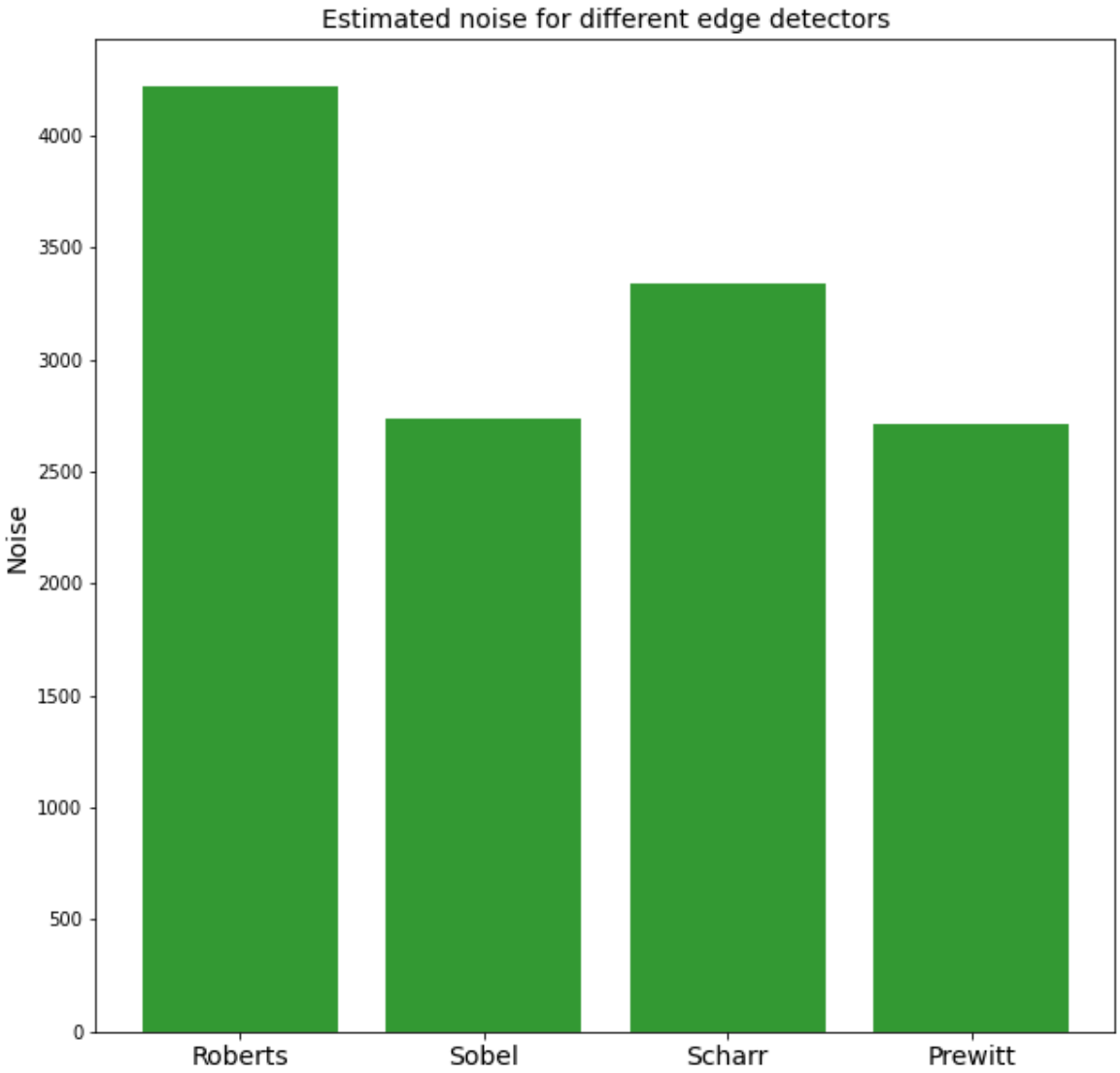

Figure 3.12: Number of detected noise points for the S2 RGB image

In order to reduce these noise values, an algorithm is implemented that identifies and removes isolated noise points. For every detected pixel, the algorithm computes the number of neighboring detected pixels. Since roads before skeletonization are several pixels wide, road pixels usually have multiple neighbors. By filtering the detected pixels that have less neighbors than a certain threshold, isolated noise points can be removed.

The results after applying this filter are shown in figure 3.13.

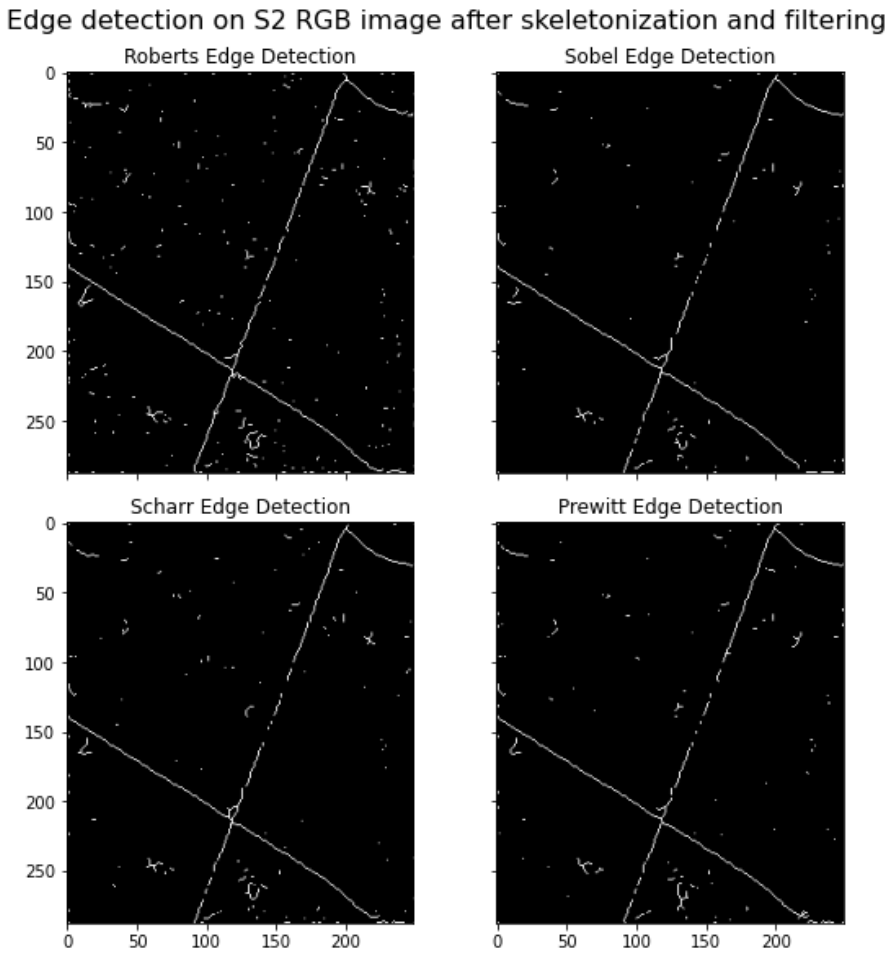

Figure 3.13: Skeletonized edges for the S2 RGB image after filtering

One immediately notices that the noise has reduced as compared to figure 3.11. However, this noise reduction comes at a cost; the roads suffer from some inconsistencies, where the filter incorrectly identifies an actual road pixel as a noise point. Despite this small error, the general results are still very acceptable.

The results of the noise assessment after applying the filter are listed in table 2 and displayed in figure 3.14. The reduction factor corresponds to the ratio of the number of noise points before and after filtering the isolated noise points.

Method	Number of noise points	Reduction factor
Roberts	469	8.998
Sobel	198	13.843
Scharr	252	13.254
Prewitt	244	11.123

Table 2: Number of detected noise points after filtering

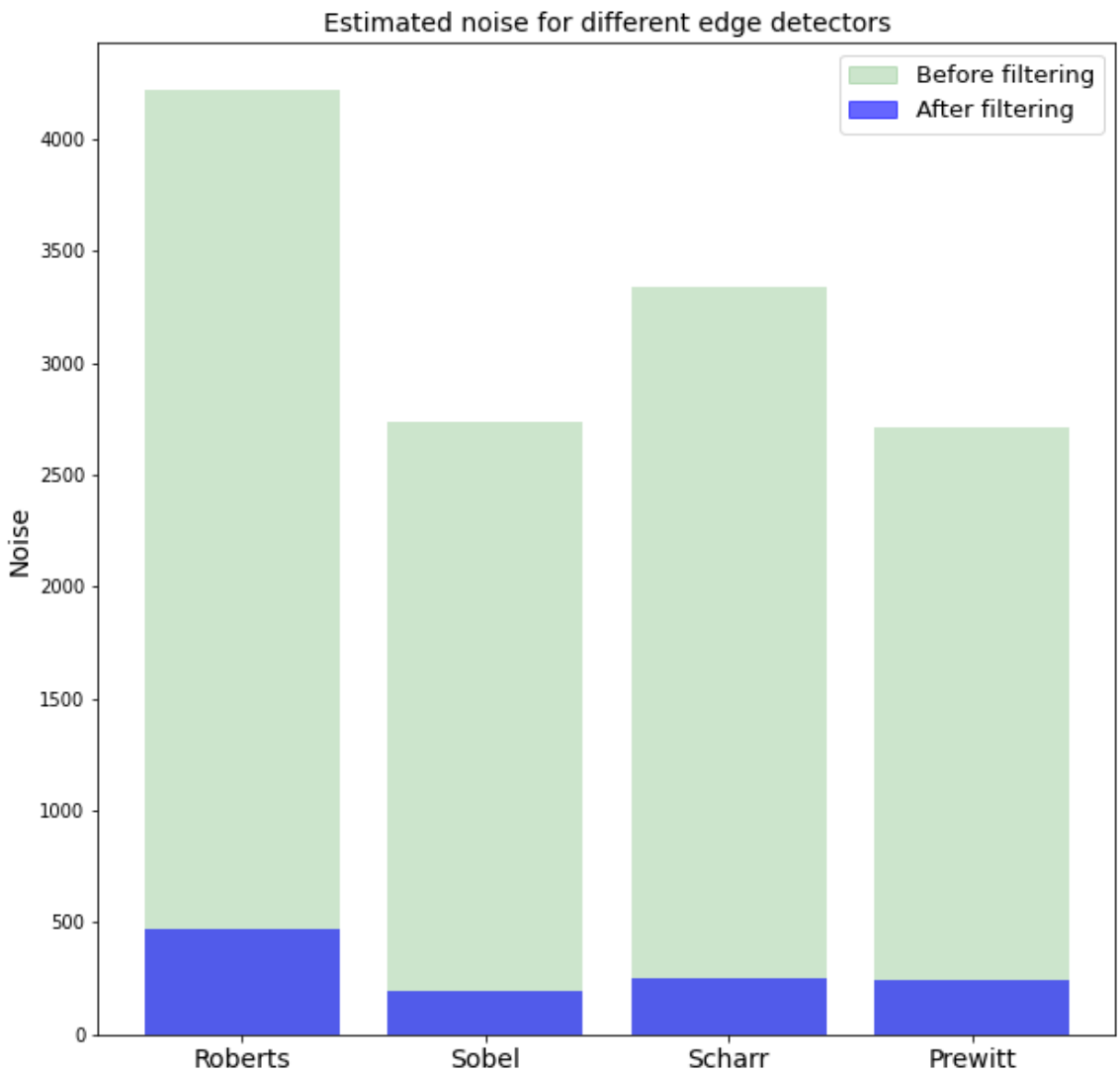

Figure 3.14: Number of detected noise points after filtering for the S2 RGB image

From table 2 and the above bar chart, it is clear that this isolated point filter works very well in terms of reducing the number of noise points.

3.4.2.3 Comparison with the ground truth

Of course, only considering the falsely detected road pixels is not the correct way to go. It is only logical that if an algorithm classifies no pixels at all as being part of a road, there will also be no noise points. Therefore, the accuracy of the detected roads will also be considered when determining the quality of the different edge detection algorithms. This is done by comparing the detected roads with the ground truth for the previously defined zones of interest, being the curved road and the intersection.

Despite not having the lowest noise value, the Sobel edge detector is retained for the remainder of this project. The roads are better preserved for this detector than for the Prewitt detector, so a trade-off has been considered.

The comparison of the Sobel detector with the ground truth is shown in figure 3.15. From the figure, it is clear that, despite the few noise points, the roads are detected quite accurately.

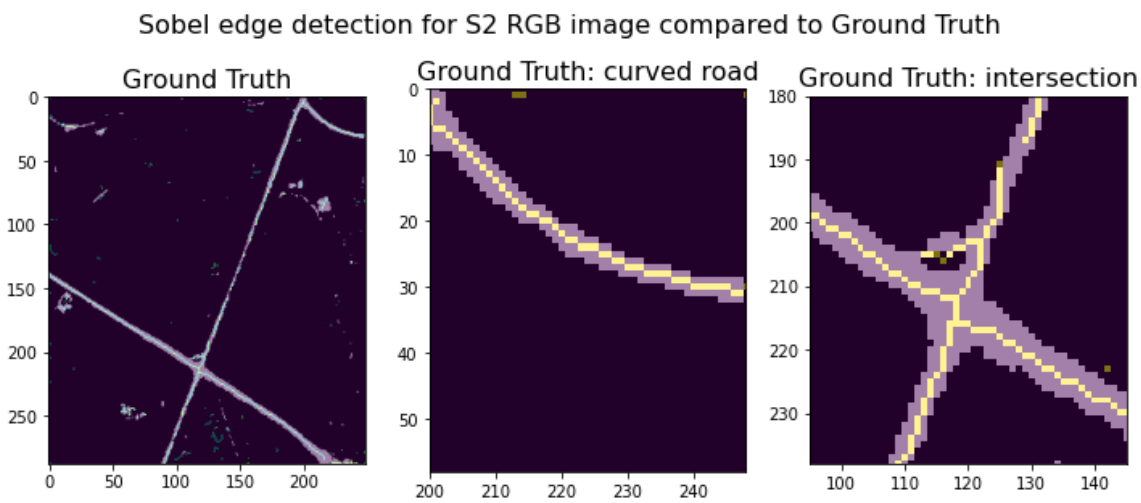

Figure 3.15: Edge detection compared to Ground Truth for S2 RGB

3.4.2.4 Determination of the pixel threshold value

Finally, the optimal threshold value for the pixel reverse operation, as explained in section 3.4.2.1, is determined. As stated before, both the amount of noise points and the accuracy of the detected roads are considered as valuable metrics to determine the best edge detection algorithm. Therefore, in figure 3.16, these two metrics are plotted as a function of the pixel reverse threshold for the Sobel detector. Note that both metrics have been normalized on this graph. The accuracy of the detected roads is measured by counting how many of the pixels belonging to the ground truth are actually detected.

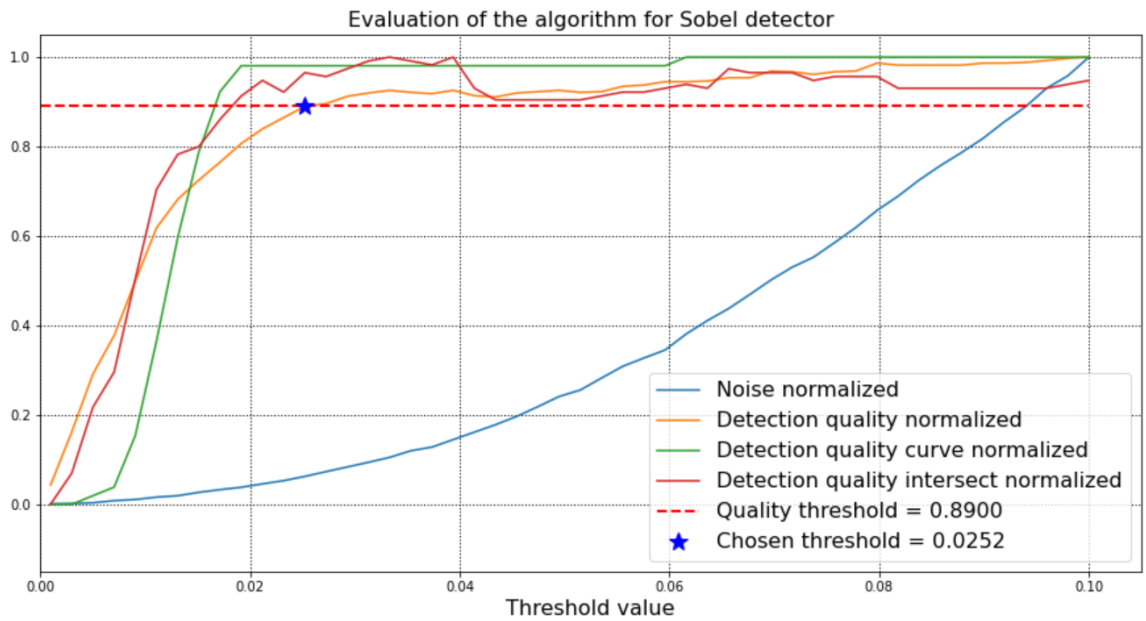

Figure 3.16: Optimal pixel threshold value for the Sobel detector

From the graph, it is clear that the noise keeps increasing for increasing values of the pixel threshold. The curve of the quality of the detected roads on the other hand shows an asymptotic behavior. By choosing the pixel threshold as the value for which this curves starts behaving asymptotically, an optimal trade-off between low noise and good detection of roads can be made.

In table 3, the absolute quality of the detected roads and the number of noise points are listed fir the different edge detectors. Note that one has to critically interpret the results for the accuracy. These values are calculated by dividing the total number of correctly detected road pixels by the number of pixels belonging to the ground truth. Since the roads in the ground truth are multiple pixels wider than the detected roads after skeletonization, it is logical the obtained results are well below 1/3. Hence, one should not focus on the absolute values, rather interpret them in a relative way to compare the different algorithms.

Method	Number of noise points	Overall Accuracy
Roberts	469	22.43%
Sobel	198	19.08%
Scharr	252	19.59%
Prewitt	244	19.15%

Table 3: Evaluation of the different edge detection algorithm

The evaluation of the other three algorithms is plotted on the figures in Appendix A.

3.4.3 Edge detection on the Sentinel-2 NIR image

Now, the same principles will be applied to the Sentinel-2 NIR and the Sentinel-1 images. In figure 3.17, the results for the different edge detection operators applied to the Sentinel-2 NIR image are shown.

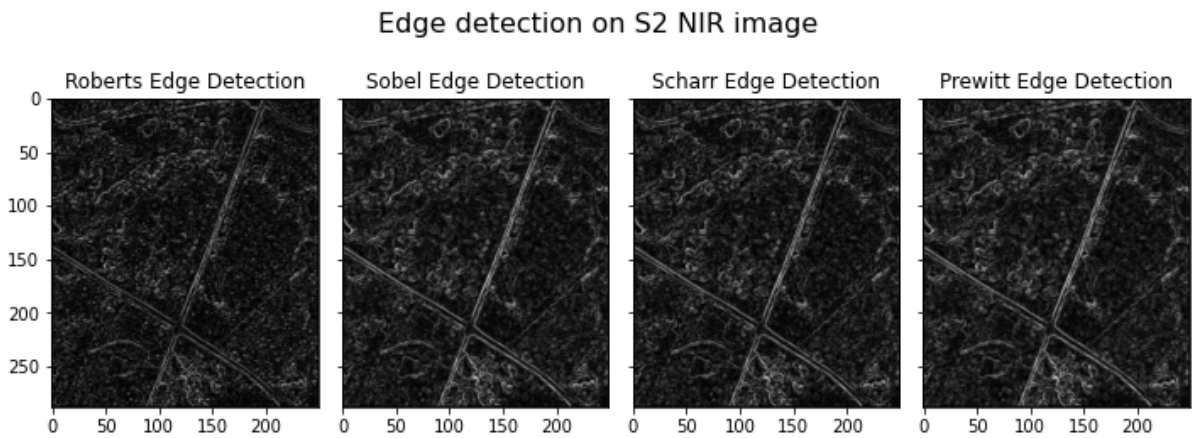

Figure 3.17: Edge operators for the S2 NIR image

One clearly sees a difference with the Sentinel-2 RGB images, as displayed in figure 3.9. However, at first sight, one can still recover the roads from these NIR images.

The results from following the same procedure as for the Sentinel-2 RGB image are shown in figure 3.18. It is clear that, even though the main roads are still visible, this method does not yield good results at all. A grid of noise points exist, and the curved road is not detected.

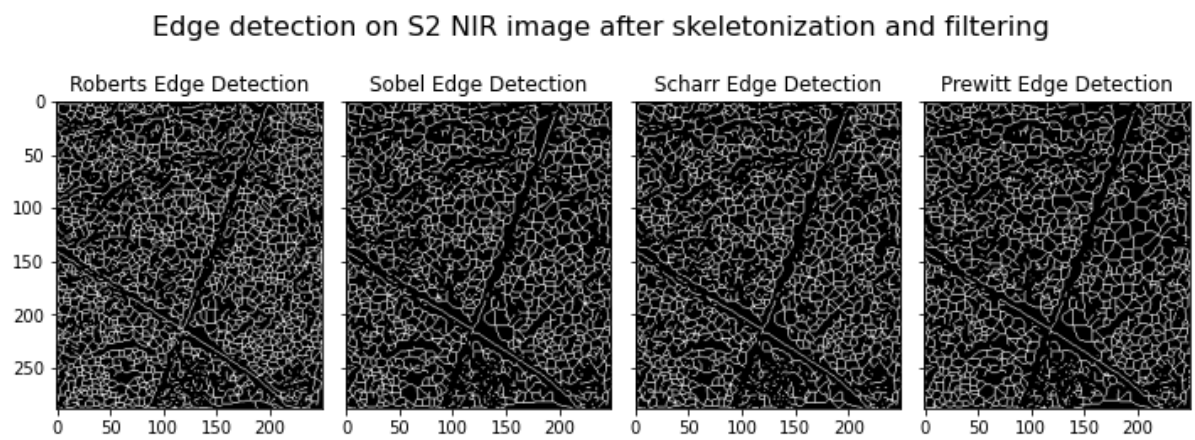

Figure 3.18: Skeletonized edges for the S2 NIR image after filtering

3.4.4 Edge detection on the Sentinel-1 VV image

Finally, the algorithm will be applied to the Sentinel-1 VV image. The results after applying the different edge detectors are shown in figure 3.19.

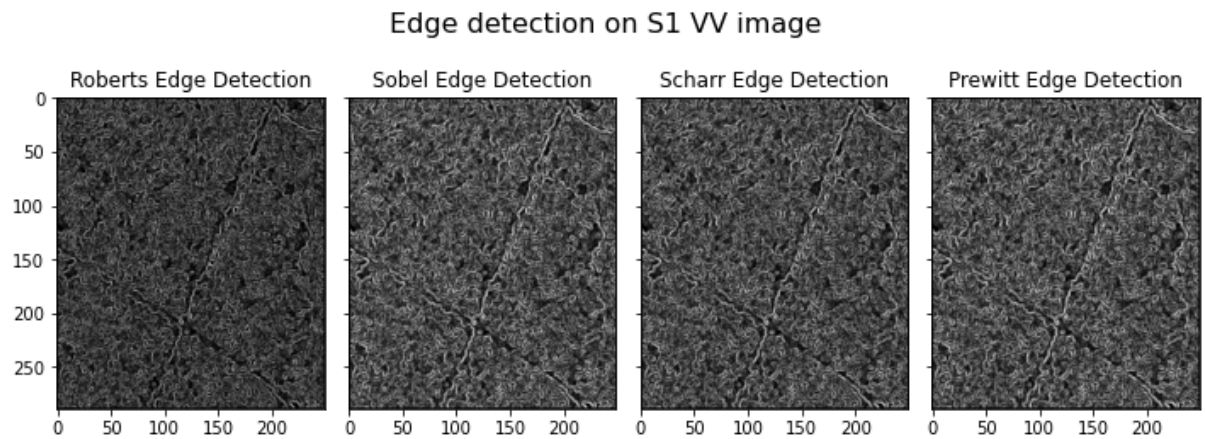

Figure 3.19: Edge operators for the S1 VV image

One can immediately see that these images are not nearly as clear as the Sentinel-2 images. After applying the remaining steps of the algorithm, filtering and skeletonization, the resulting images contain almost no detected pixels. This is because Sentinel-1 SAR images suffer from a lot of noise. The results are shown in figure 3.20.

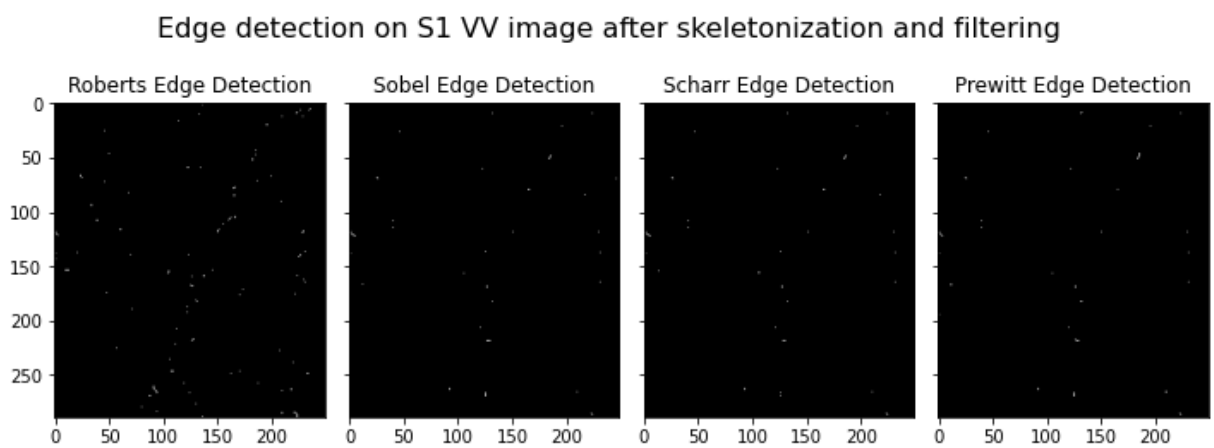

Figure 3.20: Skeletonized edges for the S1 VV image after filtering

This problem can be partially solved by applying a Gaussian filter to the Sentinel-1 images. This filter is essentially a low-pass filter that should reduce the noise present in the image. Applying a filter with $\sigma = 2.5$ and $k = 11$ yields the results shown in figure 3.21.

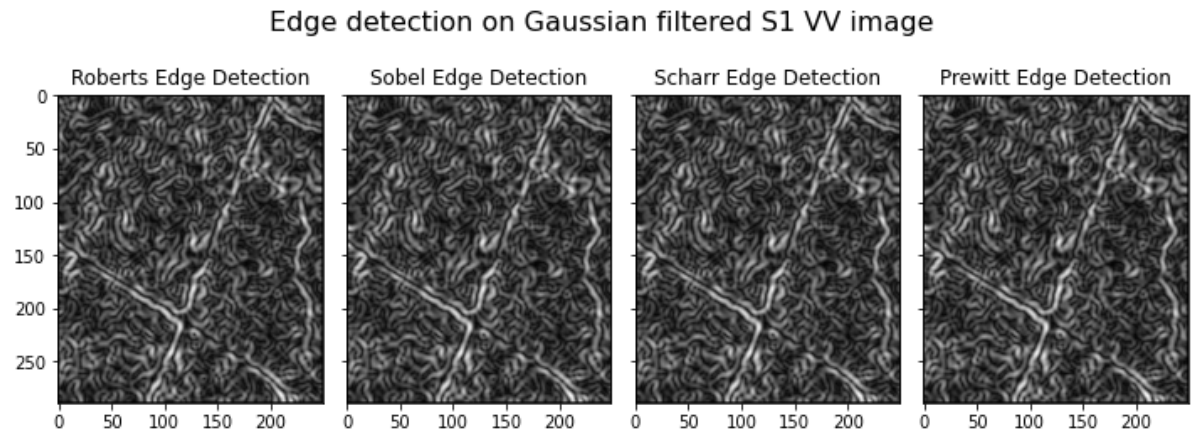

Figure 3.21: Edge operators for the Gaussian filtered S1 VV image

Applying the algorithm to this filtered version of the Sentinel-1 image yields better results, as can be seen in figure 3.22, but they are still far from optimal. Not only does the Gaussian filter remove noise, it also diminishes the contrast in the image, leading to a bad performance of the edge detecting algorithms. As a result, one can conclude that the described algorithm only performs well for the Sentinel-2 visible (RGB) images.

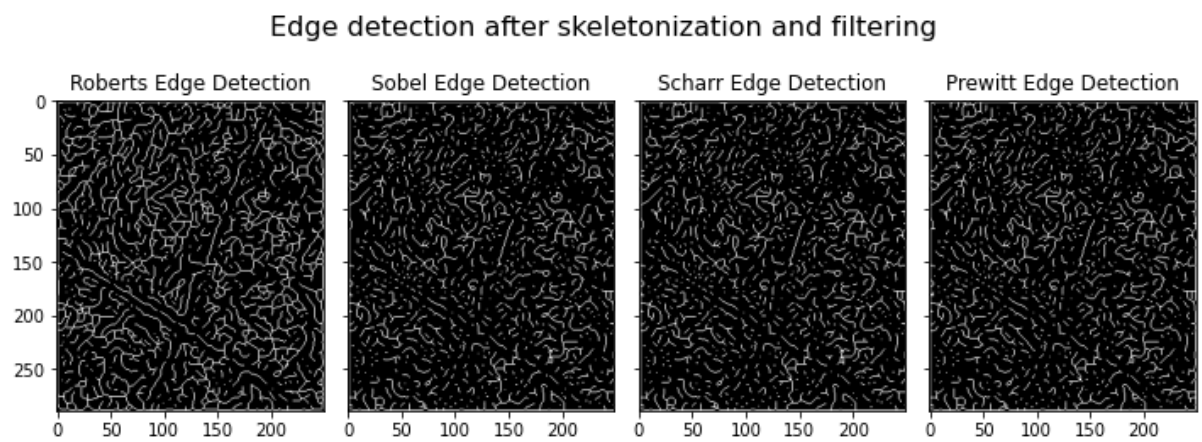

Figure 3.22: Edge operators for the Gaussian filtered S1 VV image

3.4.5 Edge detection based on the Canny filter

As a final edge detector, the Canny filter will be applied to the Sentinel-2 visible image. The Canny filter is a multi-stage edge detecting algorithm that reduces the effect of noise by first applying a Gaussian filter to the image. The rest of the process can be summarized in four major steps [11]:

1. Finding the intensity gradients of the image by using one of the four previously mentioned edge detection algorithms.

2. Gradient magnitude thresholding to obtain thinner edges.

3. Double threshold to identify potential edges. Edges with a pixel value higher than the high threshold are marked as strong edge pixels. Pixel values below the high threshold but above the low threshold are marked as weak edge pixels. Pixel values below the low threshold are suppressed.

4. Edge tracking by hysteresis. The possible connection of weak edges to strong edges is tracked by using blob analysis.

The Canny filter takes 3 values as input; the standard deviation σ of the Gaussian filter, the high threshold and the low threshold. It is not always evident to know which value of σ yield the best results. Therefore, several values are tried. The results are given in figure 3.23.

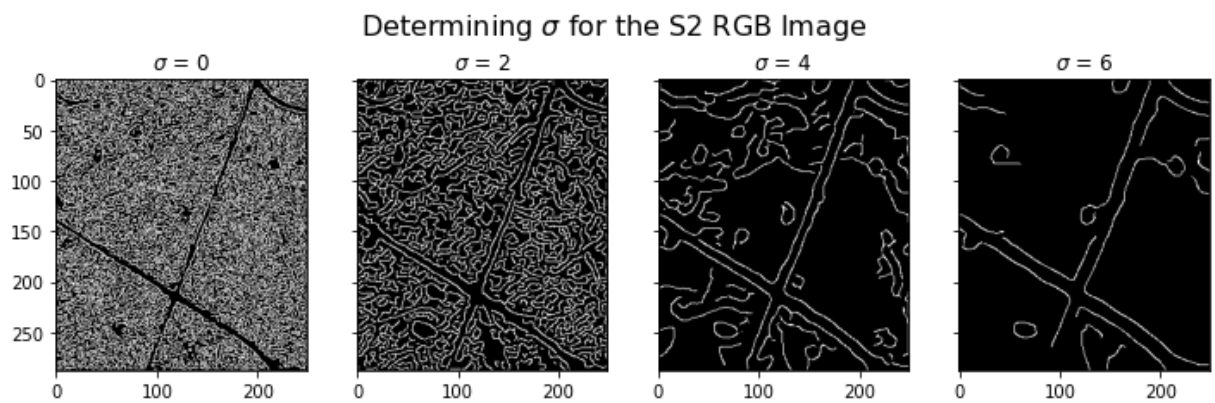

Figure 3.23: Determining the best value of σ for the S2 RGB image

If the standard deviation is set to a too low value, the Gaussian filter does not operate properly. Conversely, a too high value for σ yields an unacceptable loss of information. From the above figure, a value of $\sigma \approx 4$ seems like the best choice.

The next step is to select the lower and upper threshold values. Again, multiple options are tried. The results can be found in figure 3.24. From this figure, a value of $t_{low} = 0.15$ and $t_{high} = 0.3$ seems like a valid choice.

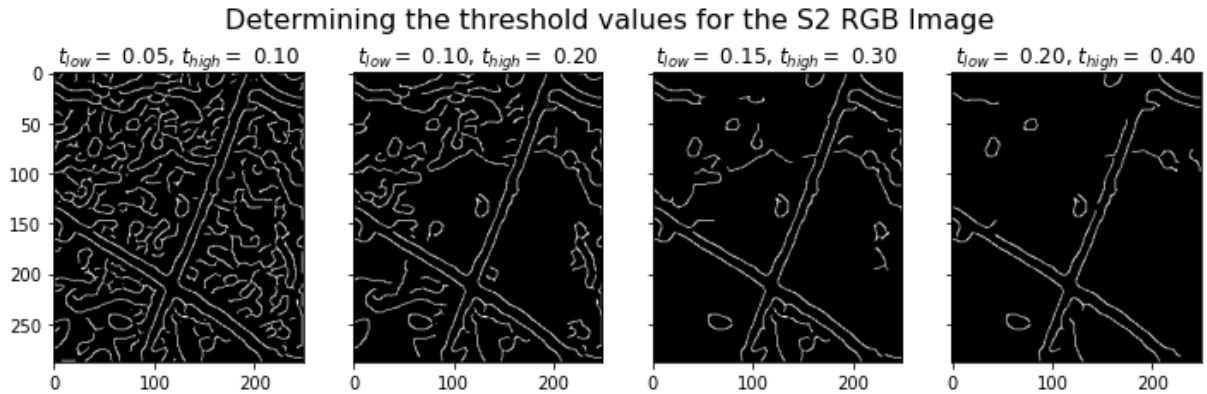

Figure 3.24: Determining the best value of t_{low} and t_{high} for the S2 RGB image

The extracted road pixels are finally compared to the ground truth, as indicated in figure 3.25. Except for the remaining noise points, that could be partially filtered out, another major inconvenience should catch the reader's attention. As was the case for the contour finding algorithm, the Canny edge detector only finds the edges of the roads instead of the middle of the road. This makes the extraction of a Road vector Layer tedious.

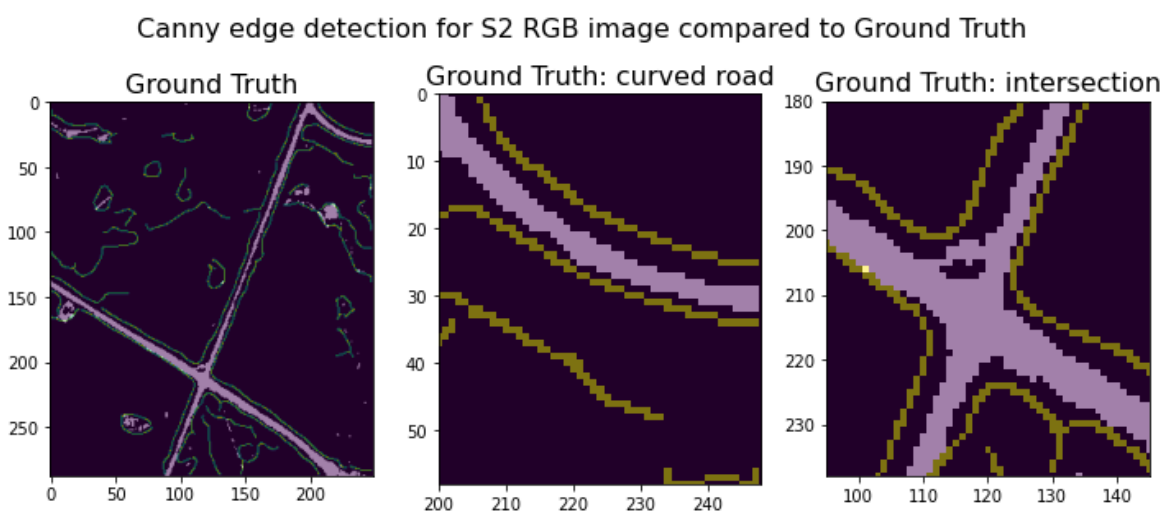

Figure 3.25: Canny filter compared to Ground Truth for S2 RGB

Note that the detected noise edges all correspond to bright spots in the Sentinel-2 image, as shown in figure 3.26. These bright spots correspond to either open spots in the woods, or to partially detected smaller roads in the Sentinel images. Getting rid of the noise is thus not a straightforward task.

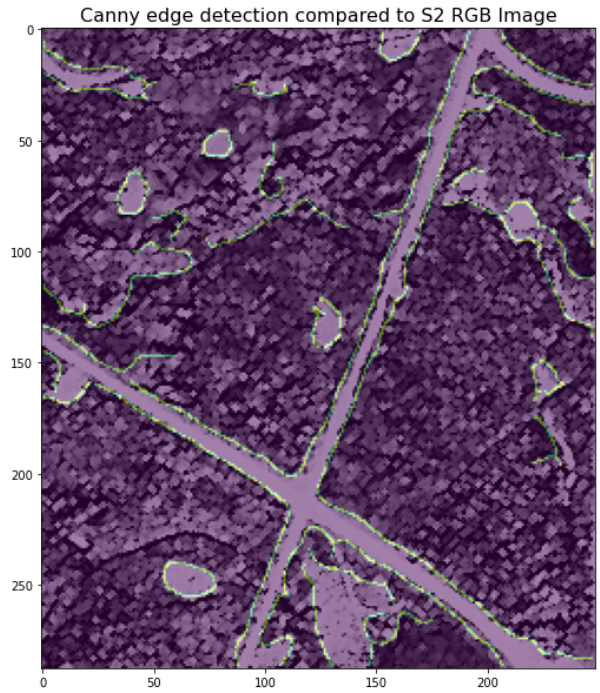

Figure 3.26: Canny filter compared to S2 RGB

As a conclusion to this section, the Sobel edge detector applied to the Sentinel-2 RGB image yields the best results. This algorithm will be further exploited in section 5 to determine the overall best road extraction algorithm.

3.5 Ridge Operators

As a final algorithm, the ridge operators will be exploited. As the name states, these filters can be used to detect ridge-like structures like neurites, tubes or vessels. The ridge filters used in this report all rely on the eigenvalues of the Hessian matrix of the image intensity to detect ridge structures where the intensity changes perpendicular to but not along the structure [12].

These ridge filters will be applied to both Sentinel-2 images.

3.5.1 Results for the Sentinel-2 RGB Image

In figure 3.27, the results for the different ridge operators with a scale factor $\sigma = 1$ are given.

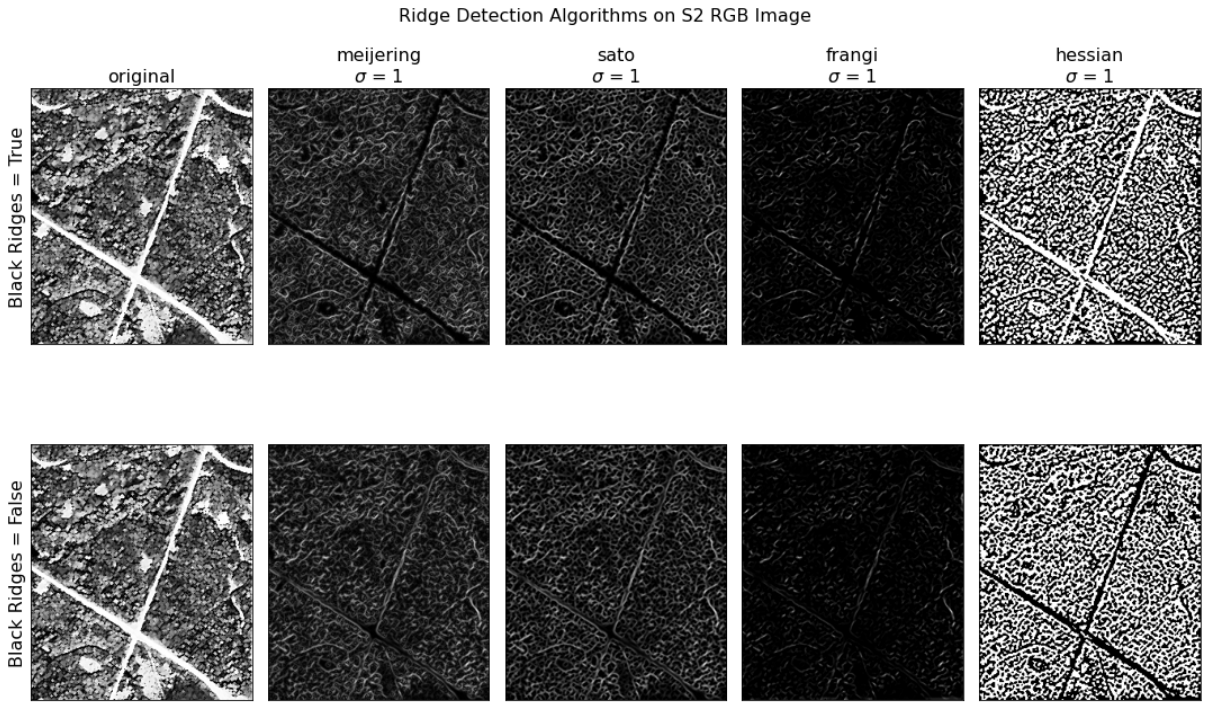

Figure 3.27: Ridge Filters applied to S2 RGB Image

From this figure, it is clear that the Hessian ridge filter with *Black Ridges = True* will most accurately detect the roads in the S2 RGB image. However, for $\sigma = 1$, this filter picks up a very large amount of noise.
The next step will thus be to determine the most suitable scale range. A larger scale range will filter out more noise, but will also less accurately detect the roads. As always, a trade-off has to be made. As in section 3.4, both the noise level and the accuracy of the

detected roads will be considered to determine the most appropriate scale range.

In figure 3.28, the influence of the scale range for the Hessian Ridge Detector is illustrated. This figure is generated by considering the scale range as $\sigma \in [1, \sigma_{max}]$ and by varying the value of the parameter σ_{max}. From this figure, one can indeed see that for $\sigma_{max} = 12$, the noise is lowest, but the curved road is not detected at all.

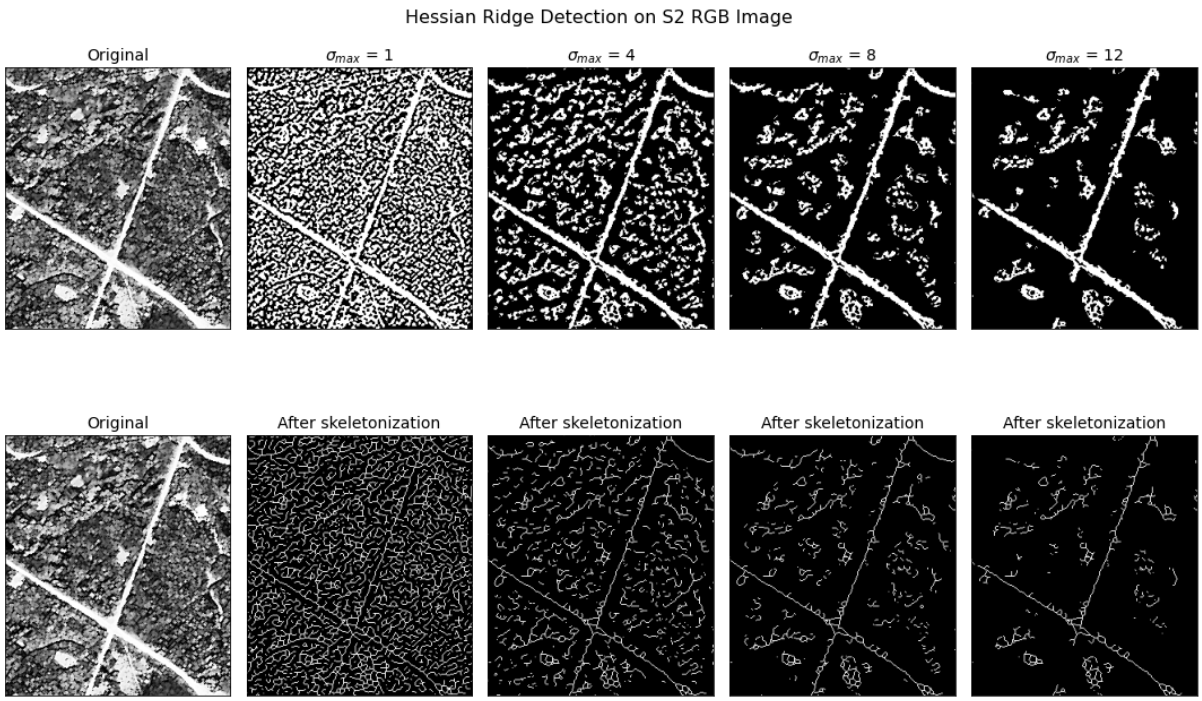

Figure 3.28: Influence of the scale range for the S2 RGB Image

At first sight, the optimal value of σ_{max} seems to be somewhere between 6 and 10. To make a reasoned choice for the value of the most suited σ_{max}, the noise will be calculated in the exact same way as described earlier in section 3.4.2.2 for different values of σ_{max}. The results are shown in figure 3.29.

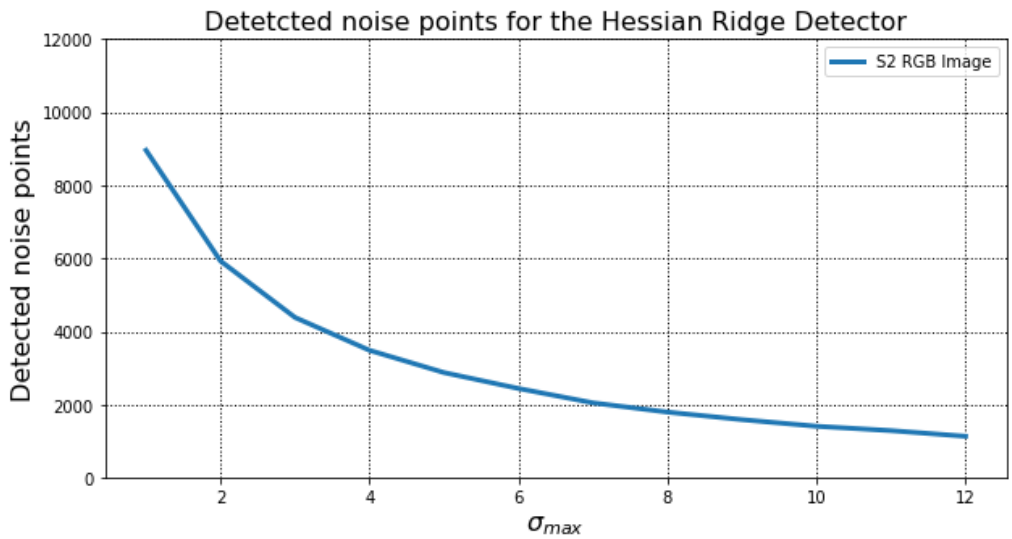

Figure 3.29: Influence of the scale range for the S2 RGB Image

The noise decreases rapidly for smaller values of σ_{max}. As from a value of $\sigma_{max} \approx 5$, this decrease is less rapid. The difference in detected noise points is not obvious for values of σ_{max} between 8 and 10. All things considered, a value of $\sigma_{max} = 9$ is chosen.

3.5.2 Results for the Sentinel-2 NIR Image

In this section, the same procedure will be repeated for the Sentinel-2 NIR image. In figure 3.30, the results of the four different ridge filters with scale factor $\sigma = 1$ are shown. Again, the Hessian filter performs best, but this time for *Black Ridges = False*.

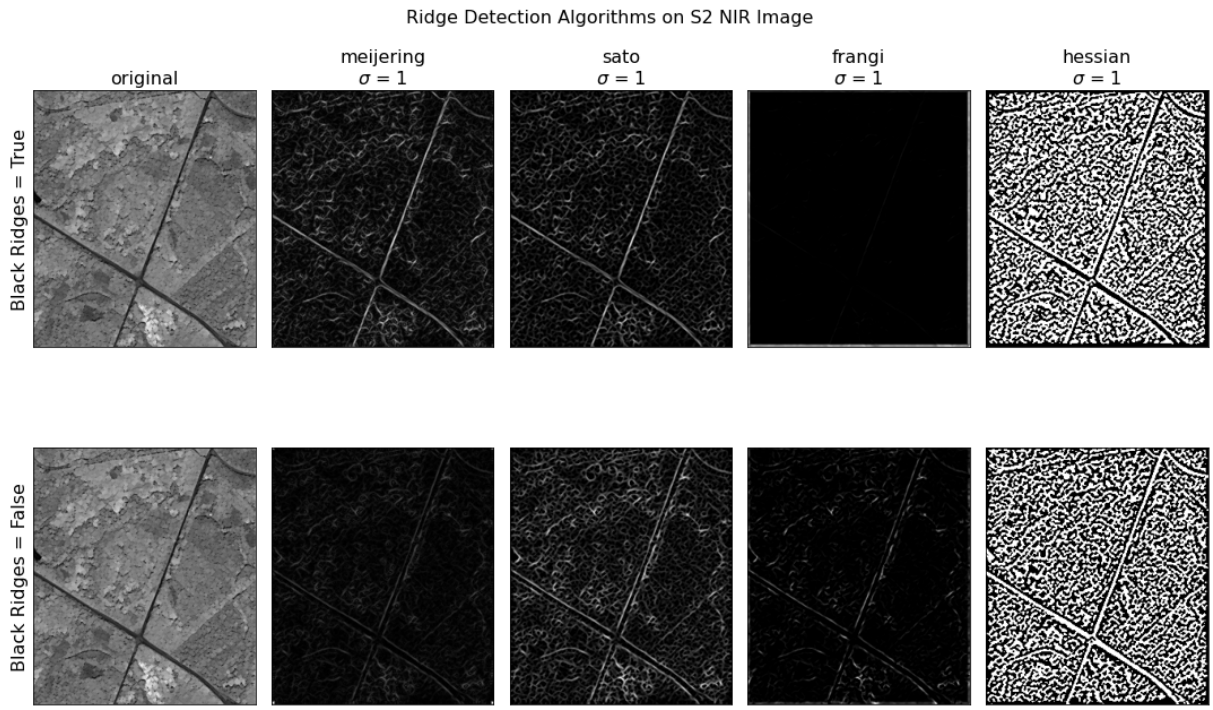

Figure 3.30: Ridge Filters applied to S2 RGB Image

Varying the value of σ_{max} now yields the results as displayed in figure 3.31. One can see that the optimal value of σ_{max} will now be around 4.

Finally, in figure 3.32, the noise for different values of σ_{max} is given for both the Sentinel-2 RGB and NIR images. For the respective optimal values of σ_{max}, the Hessian filter performs better in terms of noise for the RGB image. Moreover, in this figure, the noise level of the Sobel filter for the Sentinel-2 RGB is indicated. The Sobel filter picks up considerably less noise points (198 compared to 1589), making it a much more attractive choice in terms of noise.

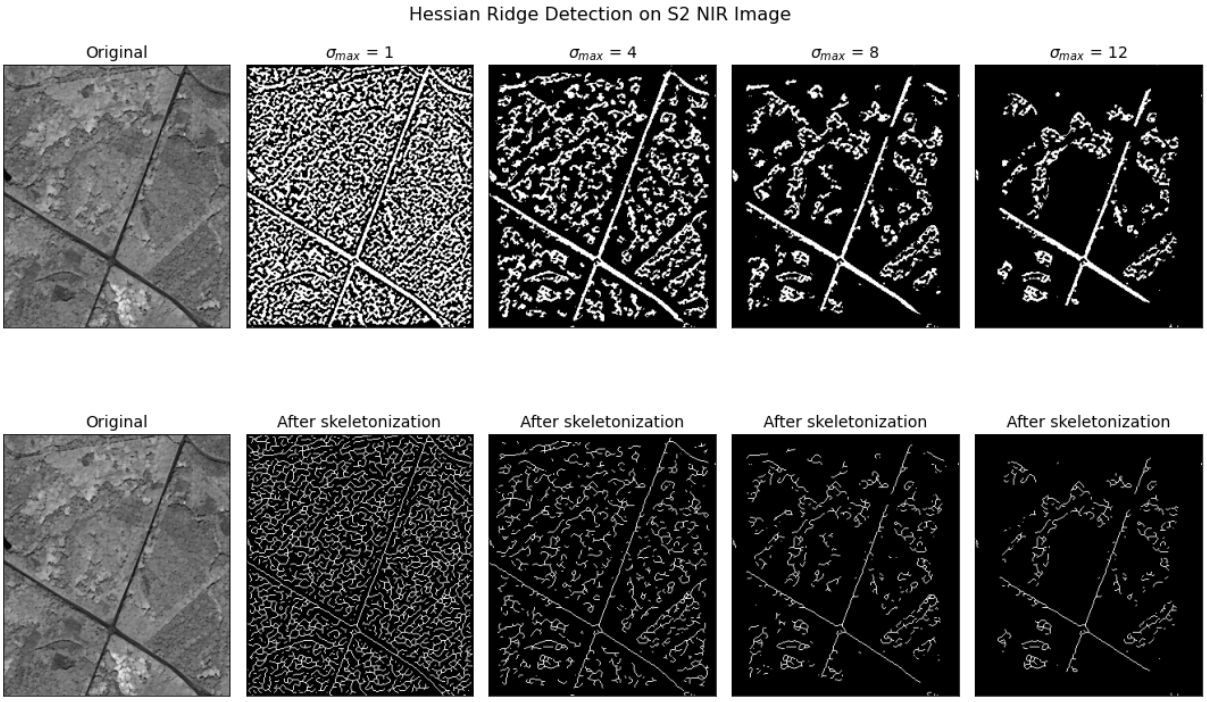

Figure 3.31: Influence of the scale range for the S2 NIR Image

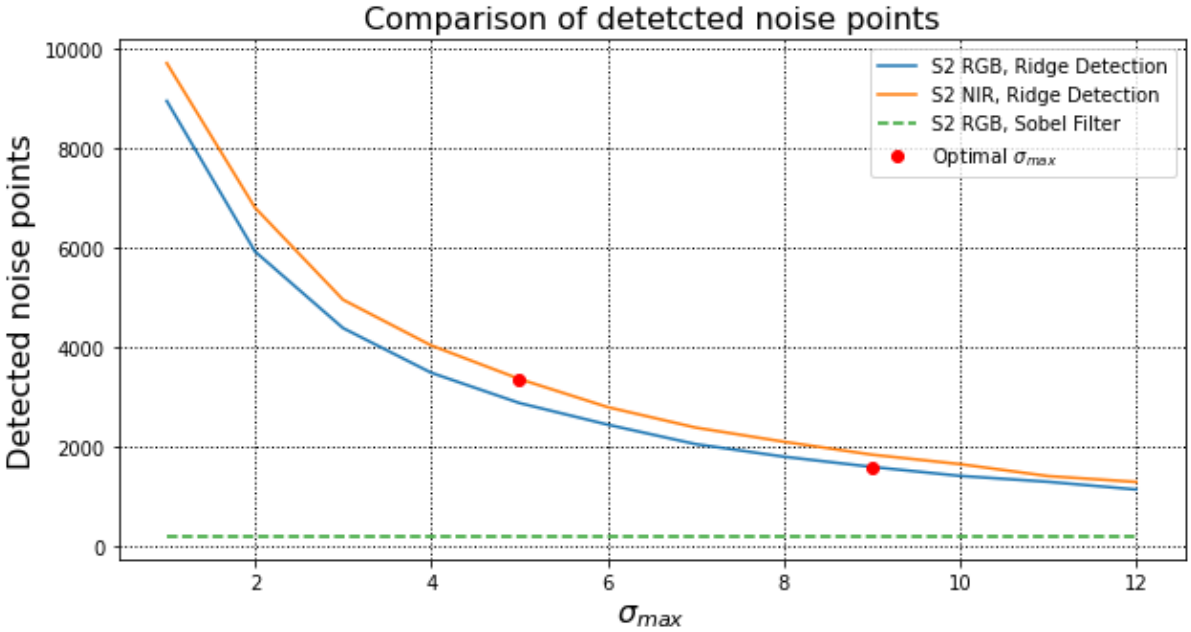

Figure 3.32: Influence of the scale range for the S2 RGB Image

3.5.3 Comparison with the ground truth

To conclude this section, the results of the Hessian Ridge Detector will be compared to the ground truth, to assess the quality of the detected roads. This comparison is visualized in figure 3.33. From the figure, it can be seen that the roads are indeed detected quite well. Calculating the accuracy in the same as explained before, one obtains an overall accuracy of 26.82%. However, due to the heavily increased noise level when using this Hessian ridge operator, the Sobel filter remains the preferred road detection algorithm, at least in this area of interest.

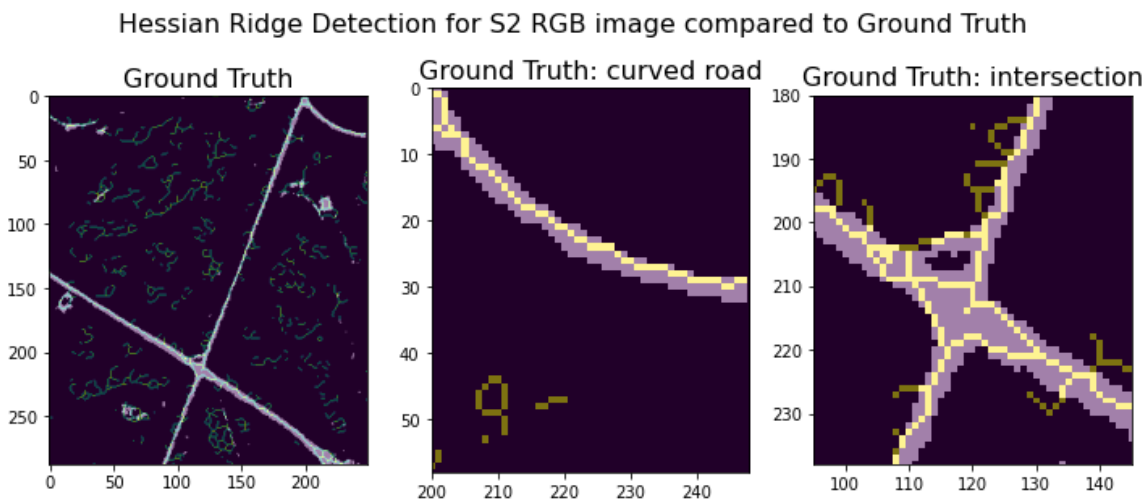

Figure 3.33: Edge detection compared to Ground Truth for S2 RGB

4 Creation of the algorithm

In this section, a tool that actually outputs an overlay Road Vector Layer will be created. From the previous sections, it is clear that the Sobel edge detector applied to the Sentinel-2 RGB image yields the best results for the considered area of interest. Therefore, this detection method will be retained for the creation of the final algorithm.

A block scheme of the functioning of the algorithm is given in figure 4.1.

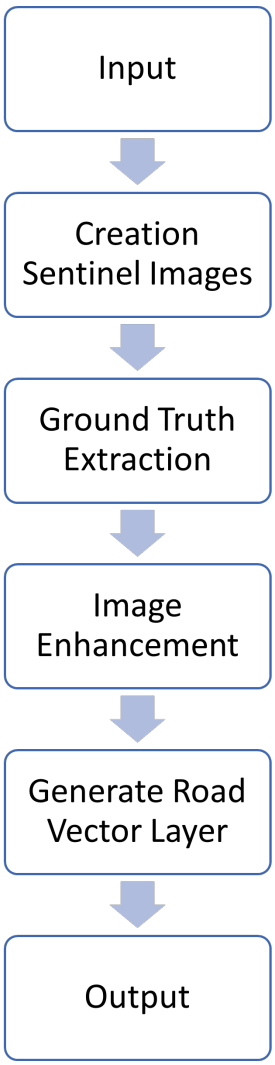

Figure 4.1: Block scheme of the road detection algorithm

The algorithm takes as input an area of interest, a range of dates and a pixel reversing threshold. If the threshold is set to *'auto'*, the algorithm will automatically calculate the best value for this threshold, based on the asymptotic behavior of the accuracy of the

detected roads as explained in section 3.4.2.4.

After creating the Sentinel-2 RGB image, the algorithm automatically extracts a ground truth from this image. Also, the RGB image is enhanced using the same techniques as explained before, namely top-hat filtering and unsharp masking. Finally, the algorithm will extract the road pixels and generate a Road Vector Layer.

At the output, the algorithm provides the detected roads as a vector of horizontal and vertical coordinates. Moreover, one can choose to receive statistics about the Road Vector Layer (number of noise points & accuracy of the detected roads).

The Python code of this algorithm can entirely be found in Appendix D.

The output of the algorithm for the test AOI is given in figure 4.2. The main crossing roads are clearly detected, as is the curved road in the top right part of the AOI. In the top left corner, the beginning of the smaller road is detected as well, but as this road fades into the trees, the detection is not accurate.

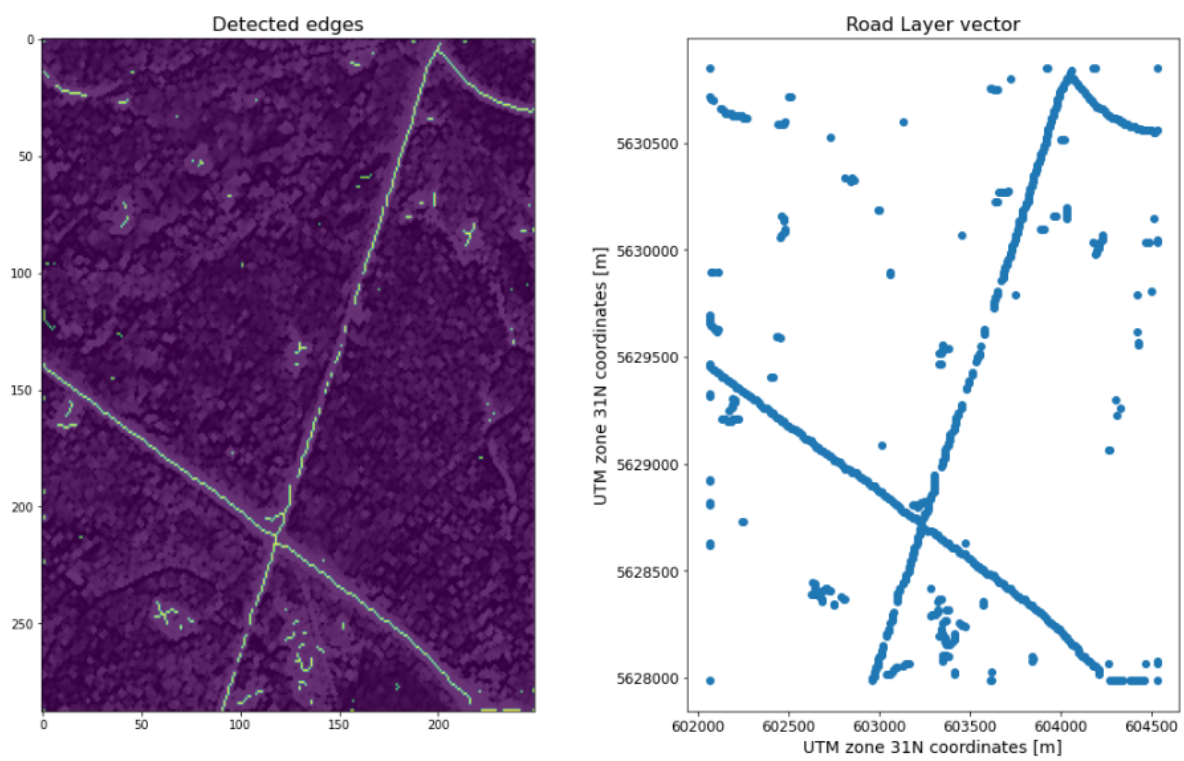

Figure 4.2: Output of the algorithm for the considered 'test' AOI

5 Validation of the algorithm on another AOI

In this final section, the algorithm will be tested on another AOI, to verify whether it still yields acceptable results in different situations. Therefore, AOIs that contains several interesting features will be considered

5.1 AOI with one main road

The first AOI that will be considered to validate the algorithm is a wooded area in Maasmechelen, Belgium. The exact boundaries of the AOI are:

- Top-Left corner: (50.98161, 5.56603)
- Bottom-Right corner: (50.9603, 5.60061)

In this first AOI, there is one main road which goes through a wooded area. It is therefore very much comparable to the original AOI, used to create the algorithm. One can thus expect a fairly accurate detection. The results of the edge detection algorithm are shown in figure 5.1.

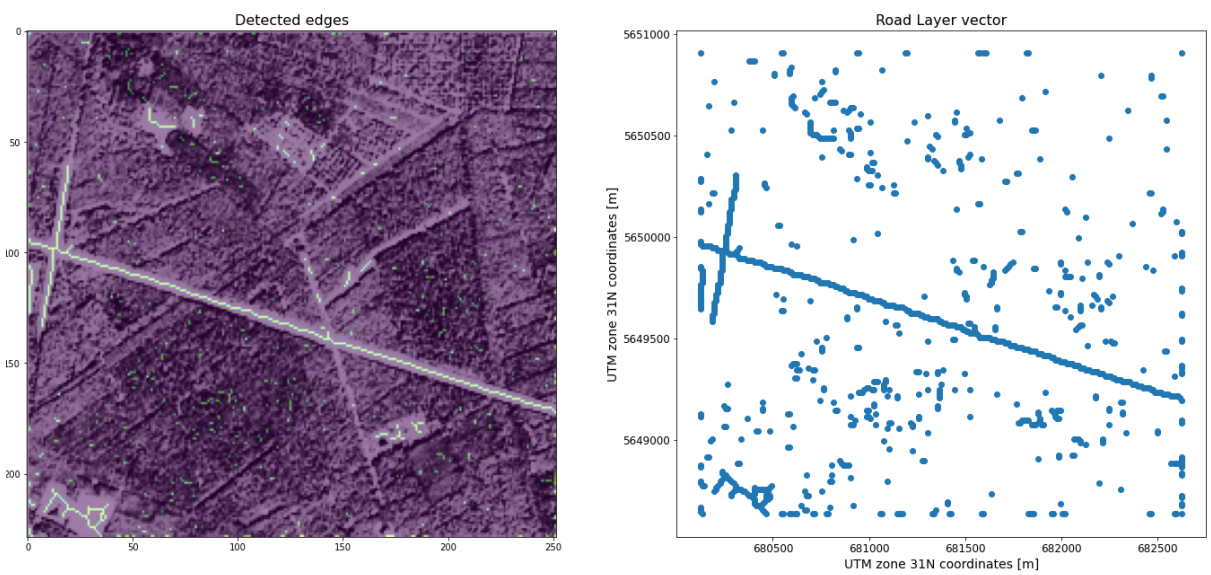

Figure 5.1: Result on AOI with one main road

The main road is indeed detected very well, while the smaller roads are left undetected. This is not too surprising, since the resolution of the Sentinel-2 images is 10 meters. With these satellites, one will never be able to accurately detect these small roads. Note that there are a considerable amount of noise points. These are mostly coming from clouds and in lesser extent from bright areas in the woods. This again shows the importance of working with cloudless Sentinel-2 images.

The contour finding algorithm should be able to also extract (part of) the smaller roads. This is shown in figure 5.2, where the algorithm looks for a constant value $c = 0.81$. A loop for multiple values of c can be found in figure C.1.

Figure 5.2: Result on AOI with one main road

5.2 AOI with water alongside the road

A bit further along the road from the previous AOI, there are 2 large water areas. Including these in the AOI for the edge detection algorithm yields the results shown in figure 5.3. It is clear that the road detection is now a lot less accurate then for the previous AOI. This is because the water areas have similar pixel values as the main road, making it very hard for the edge detection algorithm to distinguish the two.

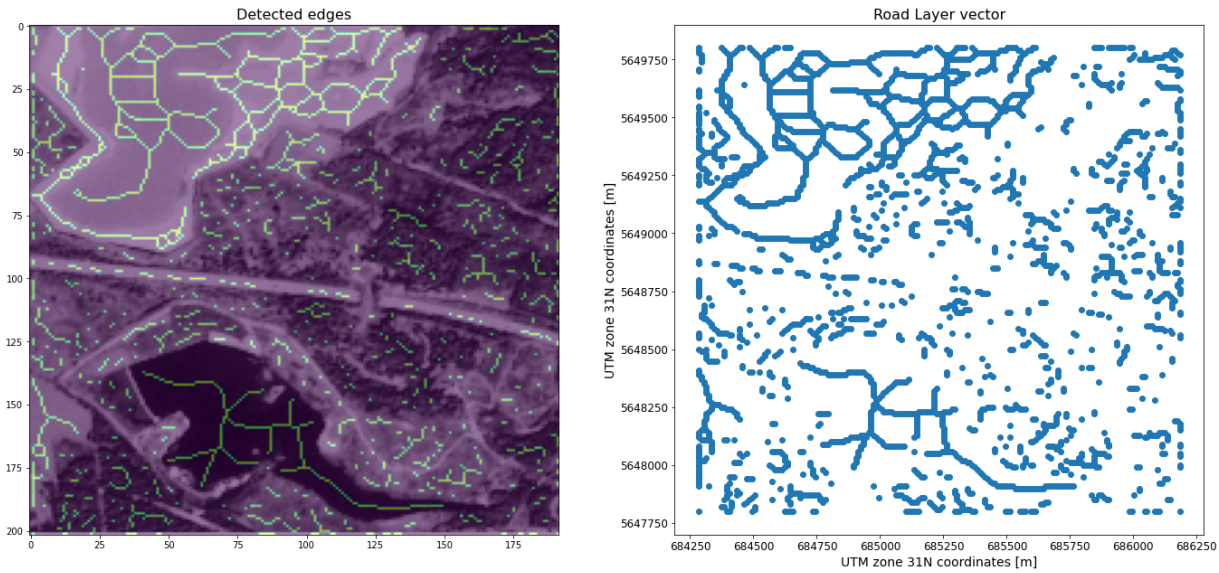

Figure 5.3: Result on AOI with one main road and water areas

For this AOI, the contour finding algorithm actually yields better results, as shown in figure 5.4. The contours of the main road are clearly found and indicated by the green and red lines. Note however a small discontinuity caused by clouds. The contour of the water area is also clearly indicated. Again, an overview of the results for multiple values of c can be found in figure C.2

One can conclude that this contour finding algorithm shows promising results in situations where the edge detection algorithm clearly fails. However, it will not be further elaborated in this paper, given it is less apt to extract an actual Road Vector Layer.

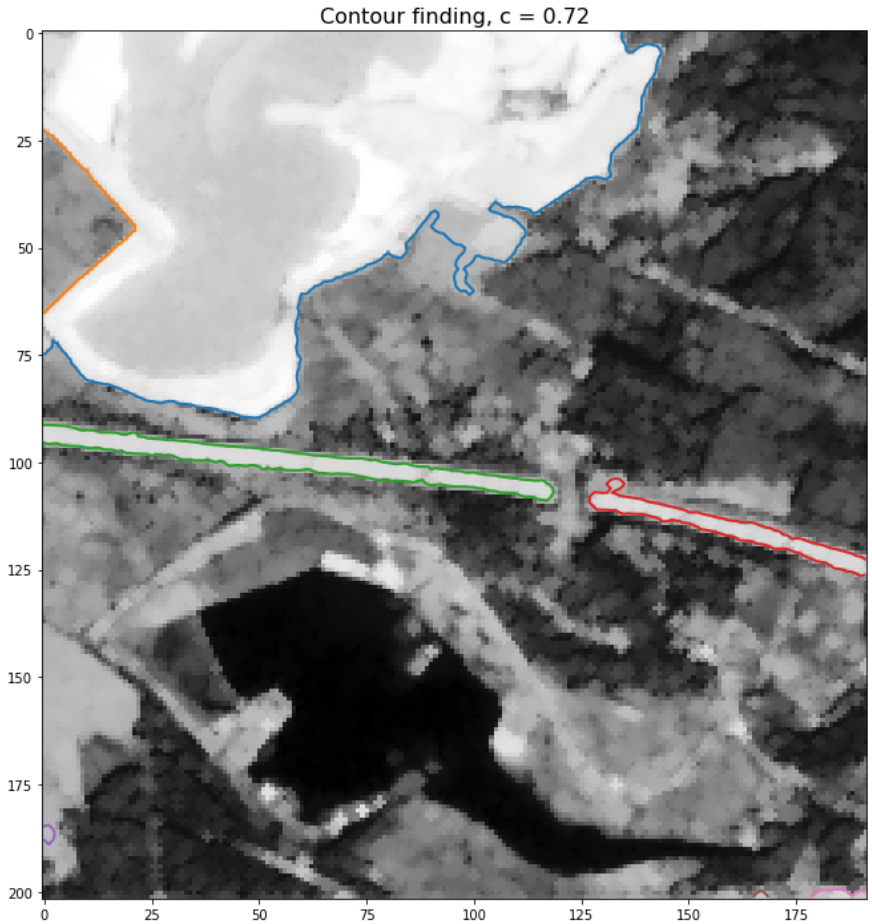

Figure 5.4: Result of contour finding on AOI with one main road and water areas

6 Conclusion

To goal of this paper was to find a method to accurately detect roads in Sentinel-1 and Sentinel-2 images. To achieve this goal, several steps have been followed.

In a first section, the Sentinel-1 and Sentinel-2 images have been created for a certain AOI. The most important parameter in choosing which images to use is the cloud mask for the Sentinel-2 image. We have shown the negative influence of clouds later in the paper.

After selecting an appropriate date for the satellite images, the ground truth has been extracted. This ground truth forms a baseline to assess the performance of the different proposed algorithms. In the AOI considered in this paper, two particularly interesting regions have been appointed, namely the intersection between the two main sets of roads and the curved road in the left top corner of the AOI.

Next, several road detection algorithms were proposed and tested on both the Sentinel-1 and Sentinel-2 images. First of all, the contour finding algorithm was found to deliver fairly accurate results after discarding the small closed loops. However, due to the difficulties in extracting the output of this algorithm and transforming it into a GIS, this method was not retained.
Secondly, five different edge detection algorithms have been exploited. It became clear that the S2 RGB image is by far the most appropriate image to use for road detection. Of these filters, the Sobel filter turned out to deliver the best results, based on a trade-off between the amount of noise and the accuracy of the detected roads.
Lastly, four different ridge operators were tried on the Sentinel-2 images. It is shown that the Hessian ridge operators detects the roads quite well, but with a lot of noise. In conclusion, the Sobel filter applied to the S2 RGB image was considered the best road detection algorithm.

Finally, an actual algorithm was implemented that combines the different steps explained throughout this report. When validating the algorithm, it was found that the algorithm performs well for wider main roads, but has difficulties picking up smaller side roads. Also, clouds were proven to have a very negative effect on the performance of the algorithm. The contour finding algorithm on the other hand did manage to pick up some of the smaller roads.
As a final step, it has been shown that large open areas in the image could mislead the Sobel detector. In this case, the contour finding algorithm actually delivers more promising results.

A Evaluation of the Edge Detection algorithms

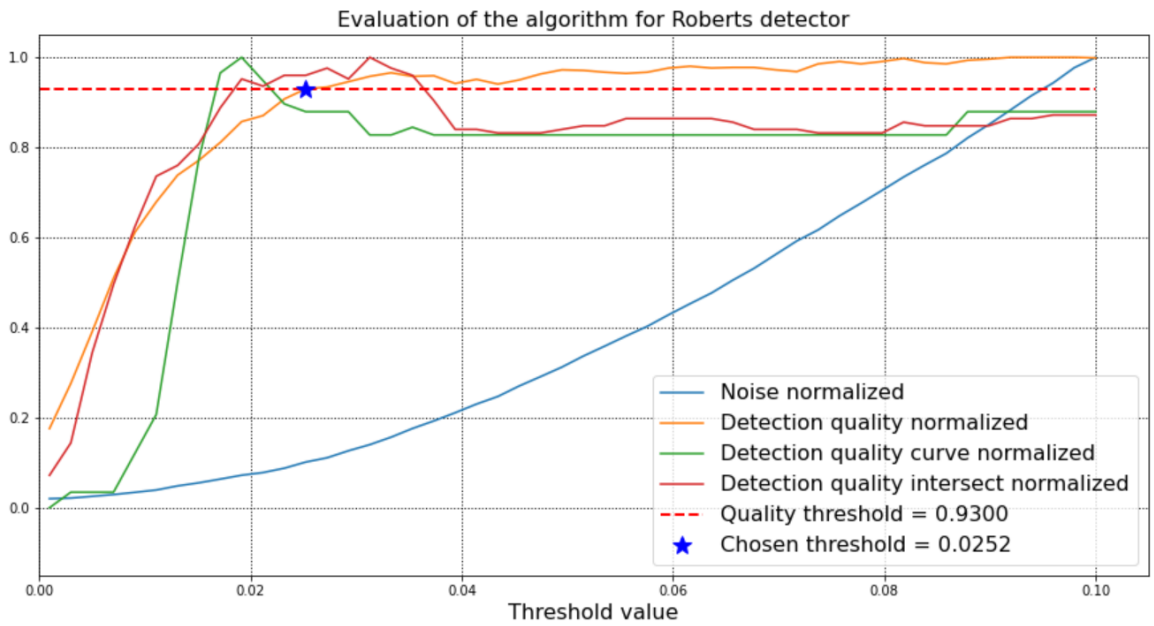

Figure A.1: Optimal pixel threshold value for the Roberts detector

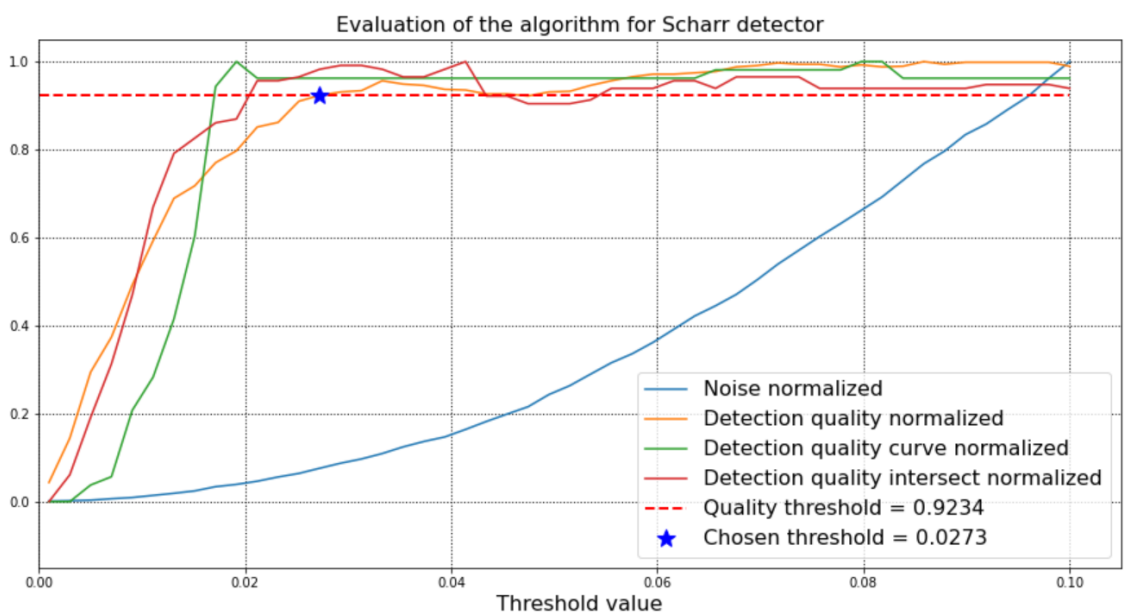

Figure A.2: Optimal pixel threshold value for the Scharr detector

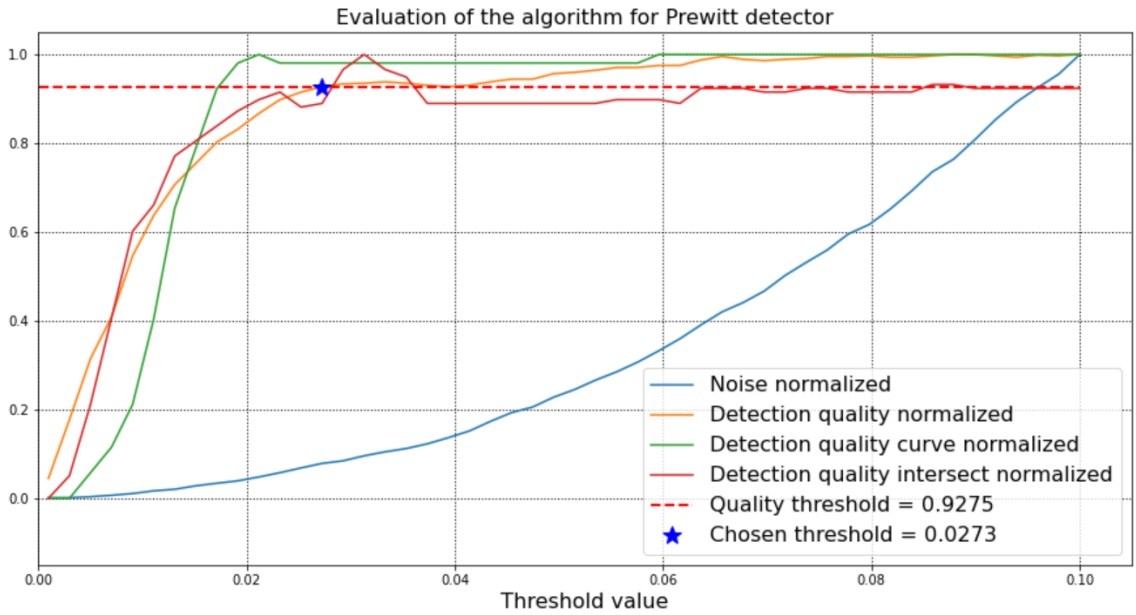

Figure A.3: Optimal pixel threshold value for the Prewitt detector

B Contour Finding

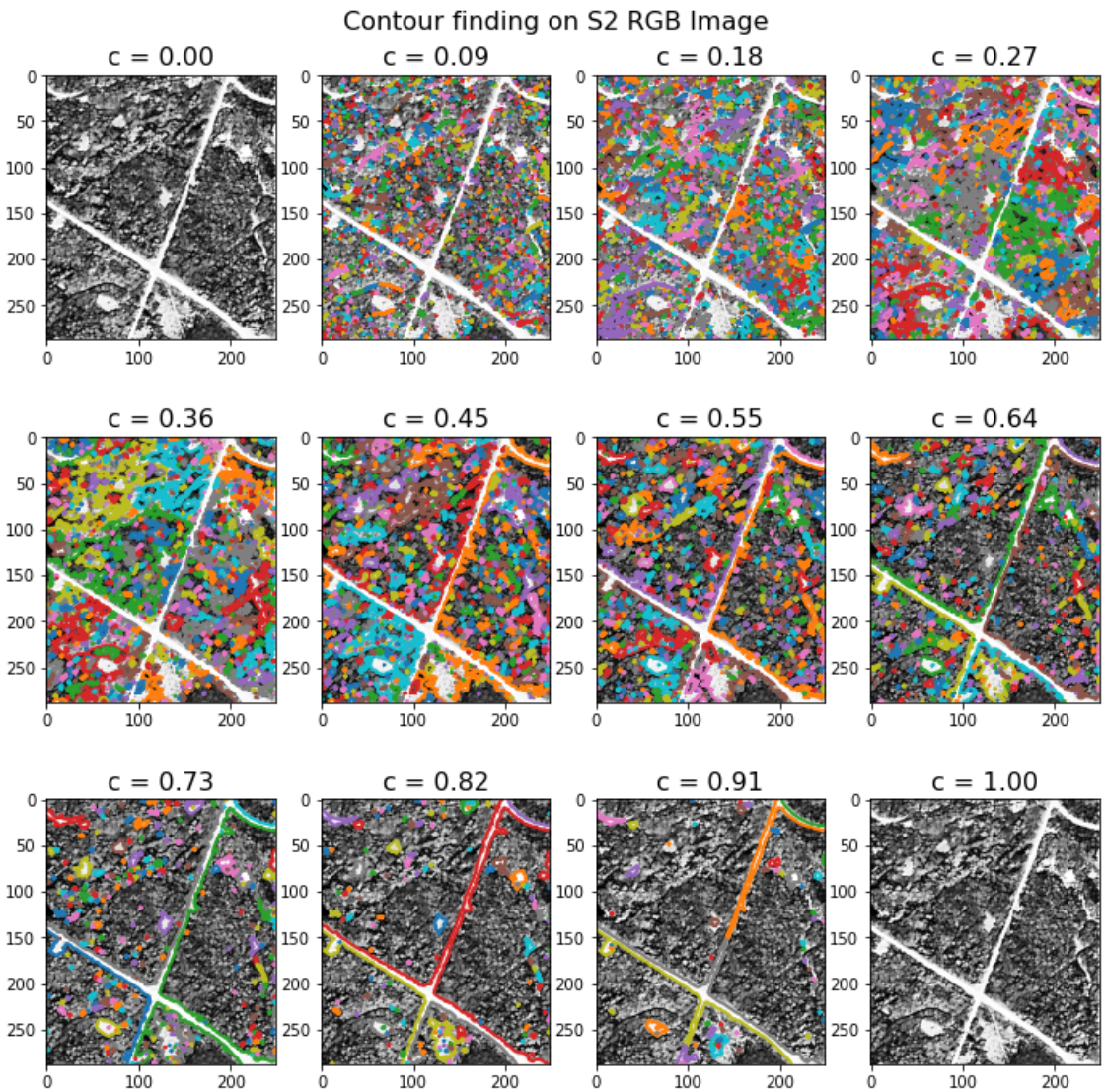

Figure B.1: Finding c for the S2 RGB image

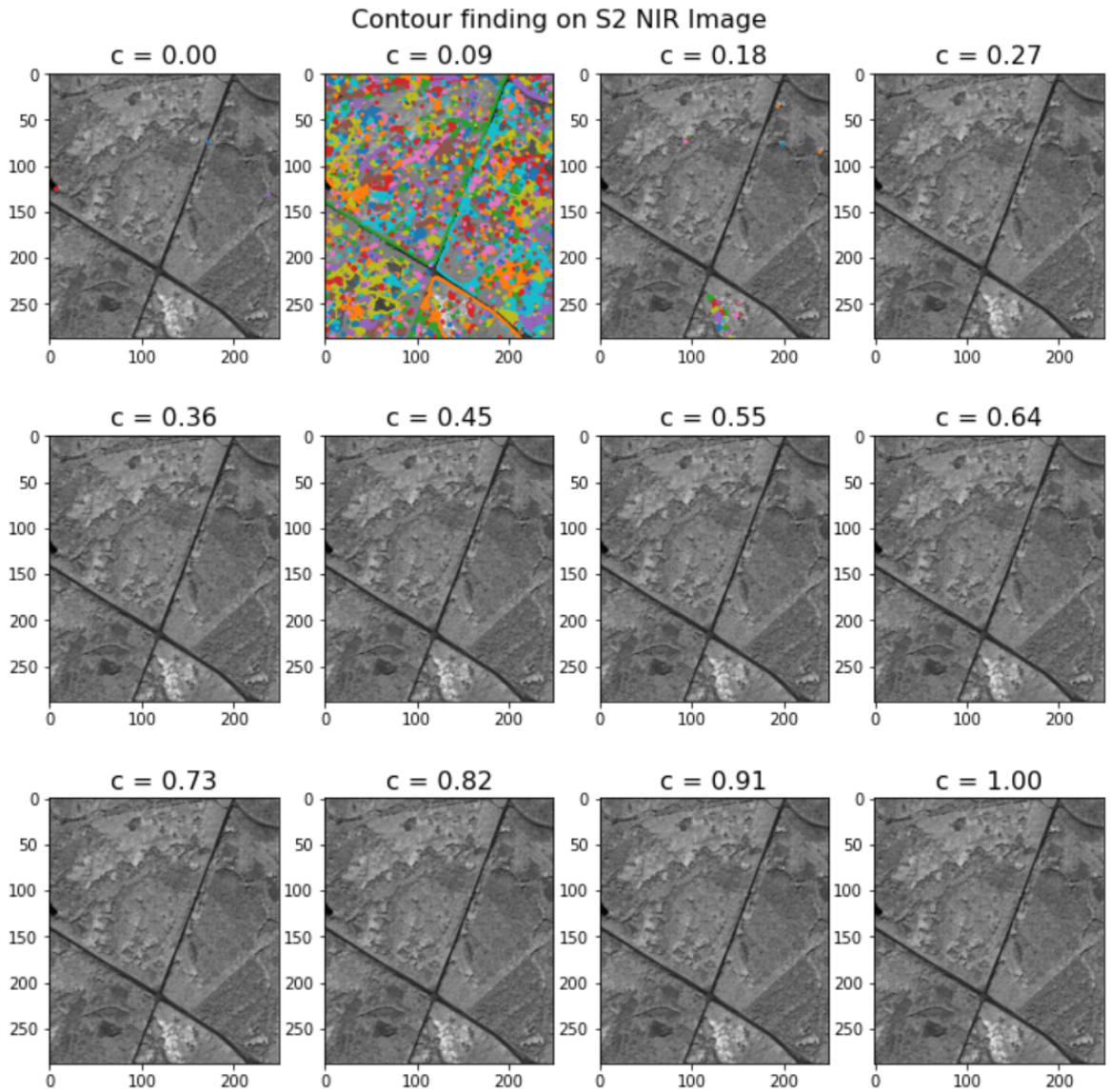

Figure B.2: Finding c for the S2 NIR image

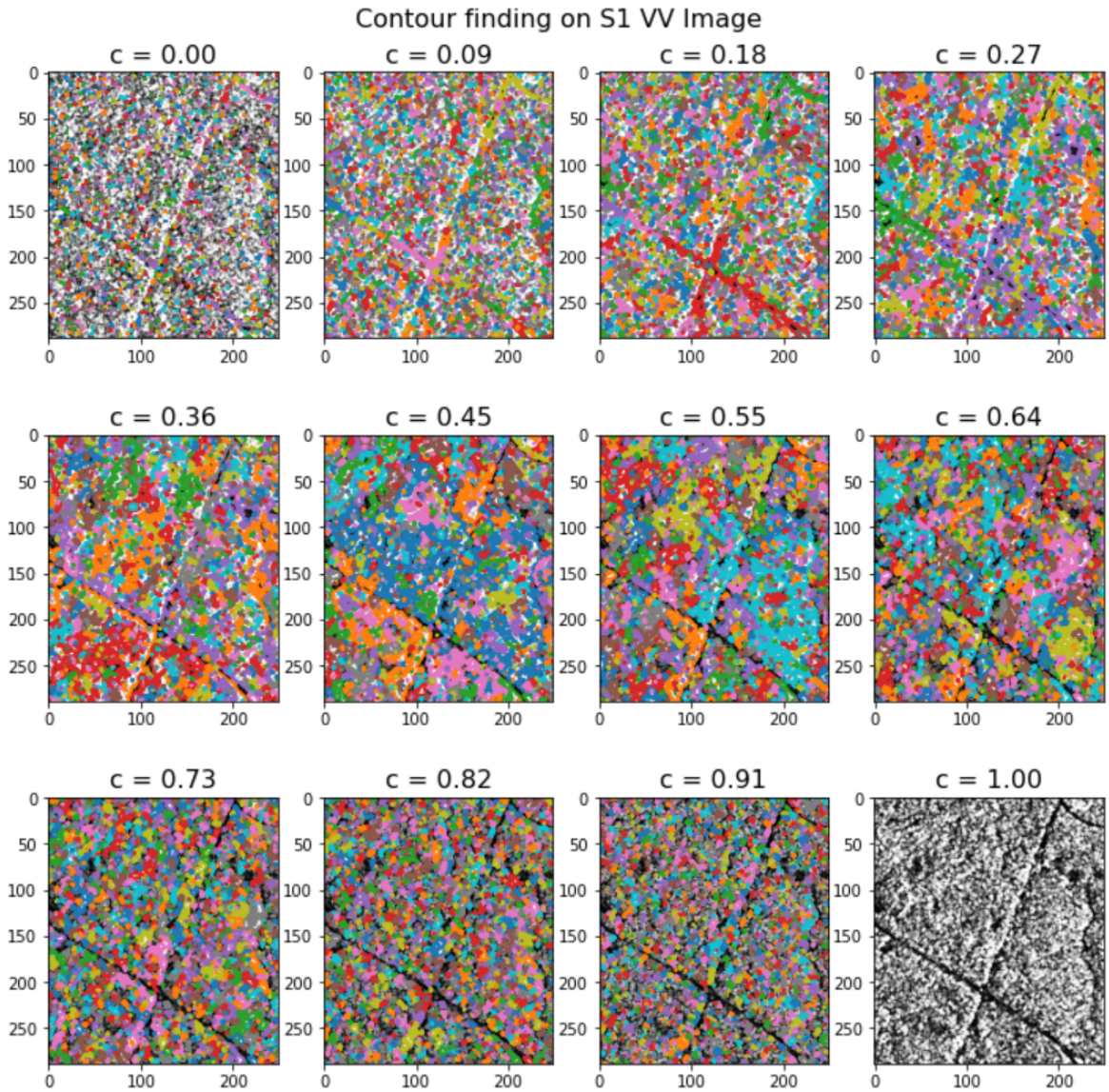

Figure B.3: Finding c for the S1 VV image

C Validation

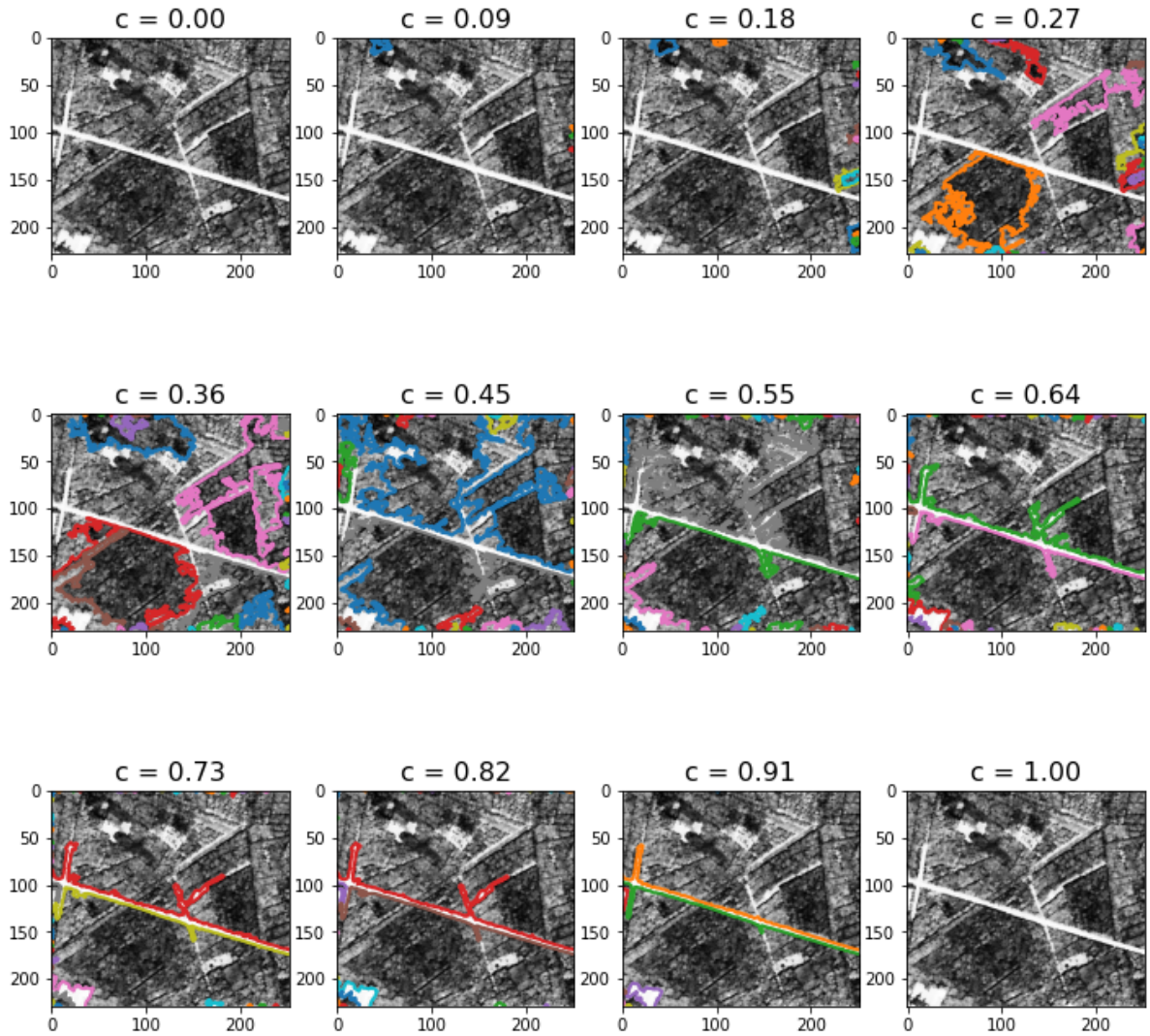

Figure C.1: Finding c for the first S2 RGB validation image

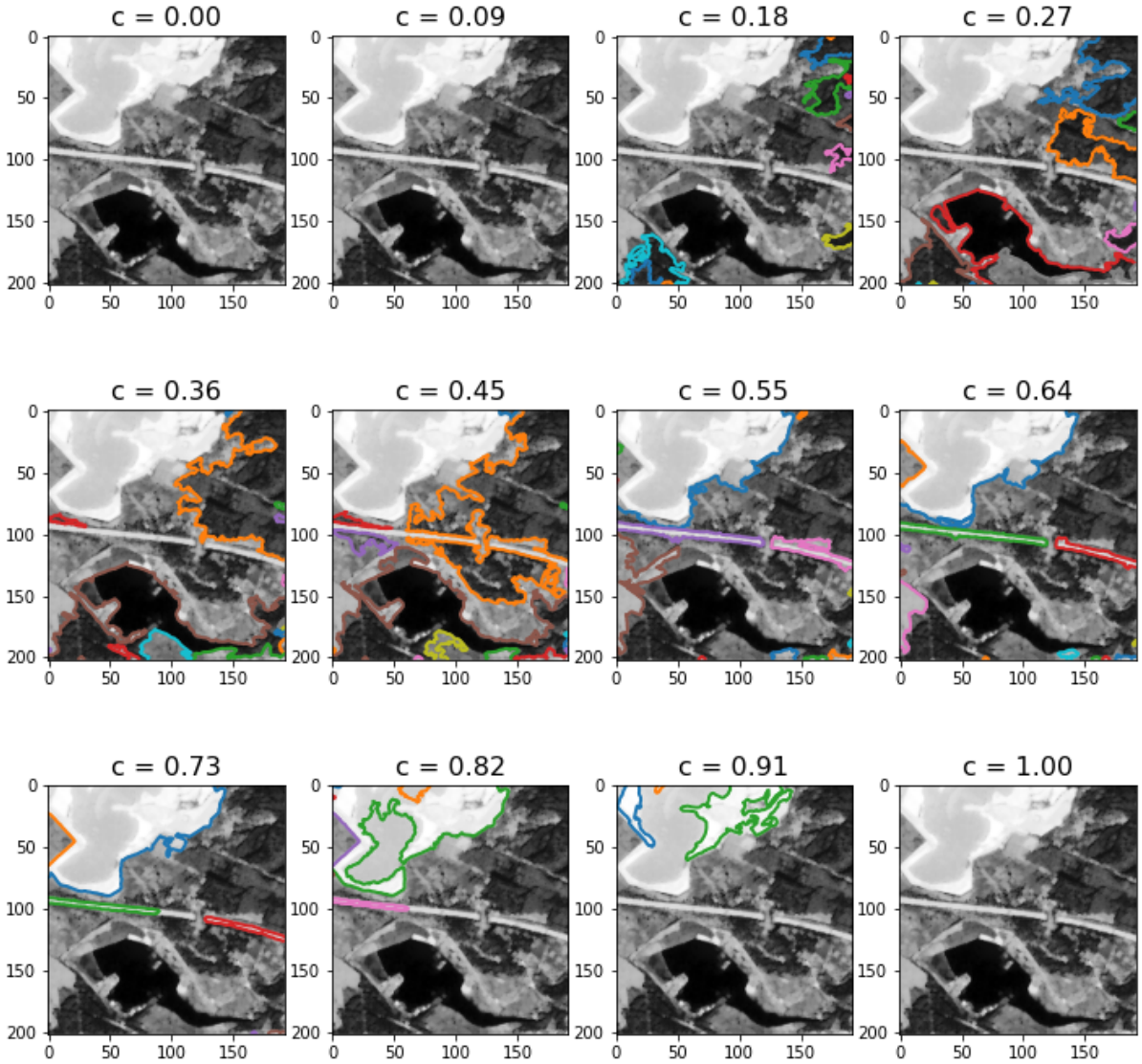

Figure C.2: Finding c for the second S2 RGB validation image

D Code

```
def extract_S1_image(start_date, end_date, aoi_lonlat, aoi_crs, do_plot):
    catalogue = Catalogue()
    collections_S1 = catalogue.get_collections(platform="SENTINEL-1")
    products_S1 = list(catalogue.get_products('urn:eop:VITO:CGS_S1_GRD_SIGMA0_L1',start_date,end_date))

    for idx, product in enumerate(products_S1):
        if aoi_lonlat.within(box(*product.bbox)):
            idx_S1 = idx
    image_S1 = products_S1[idx_S1]

    if do_plot:
        files = dict([(item.title, get_filename(item)) for item in image_S1.data])
        images_aoi = extract_aoi(aoi_crs, files)
        plot_bands(images_aoi)
        plt.figtext(0.5, 0.75, 'Sentinel-1 bands', ha='center',fontsize = 16)

    return image_S1

def extract_S2_image(start_date, end_date, aoi_lonlat, aoi_crs, do_plot):
    cat = catalog.Catalog()
    products_S2 = cat.get_products(
        'CGS_S2_RADIOMETRY_10M',
        'GEOTIFF',
        start_date,
        end_date,
        mapping(aoi_lonlat)['coordinates'][0][3][0],
        mapping(aoi_lonlat)['coordinates'][0][1][0],
        mapping(aoi_lonlat)['coordinates'][0][3][1],
        mapping(aoi_lonlat)['coordinates'][0][1][1],
    )
    for counter, img in enumerate(products_S2):
        if (img._timestamp.year,img._timestamp.month,img._timestamp.day) == (start_date.year,start_date.month,start_date.day):
            break
    image_S2 = products_S2[counter]

    bands = image_S2.bands()
    for counter, value in enumerate(bands):
        if value == 'B02':
            B_index = counter # Blue component of visible image
        elif value == 'B03':
            G_index = counter # Green component of visible image
        elif value == 'B04':
            R_index = counter # Red component of visible image
        elif value == 'B08':
```

```
            NIR_index = counter
        elif value == 'CLOUDMASK':
            CL_index = counter
        elif value == 'SHADOWMASK':
            SH_index = counter
    imagename_B = str(image_S2.files[B_index])[5:]
    imagename_G = str(image_S2.files[G_index])[5:]
    imagename_R = str(image_S2.files[R_index])[5:]
    imagename_NIR = str(image_S2.files[NIR_index])[5:]
    imagename_CL = str(image_S2.files[CL_index])[5:]
    imagename_SH = str(image_S2.files[SH_index])[5:]

    equalized_image_R = equalized_image(imagename_R,aoi_crs)
    equalized_image_G = equalized_image(imagename_G,aoi_crs)
    equalized_image_B = equalized_image(imagename_B,aoi_crs)

    img_RGB = np.dstack((equalized_image_R, equalized_image_G, equalized_image_B))
    img_NIR = msk.mask(
        rasterio.open(imagename_NIR),
        [aoi_crs],
        all_touched=False,
        invert=False,
        nodata=None,
        filled=True,
        crop=True,
        pad=False,
        indexes=None,
    )[0][0]

    if do_plot:
        fig, ax = plt.subplots(1, 3,figsize=(12,12))
        ax[0].imshow(equalized_image_R)
        ax[0].set_title('R')
        ax[1].imshow(equalized_image_G)
        ax[1].set_title('G')
        ax[2].imshow(equalized_image_B)
        ax[2].set_title('B')
        plt.figtext(0.5, 0.67, 'RGB components', ha='center',fontsize = 16)
        fig, ax = plt.subplots(1,2,figsize=(12,12))
        ax[0].imshow(img_RGB)
        ax[0].set_title('RGB')
        ax[1].imshow(img_NIR, cmap=plt.cm.gray)
        ax[1].set_title('NIR')
        plt.figtext(0.5, 0.74, 'Visible and Near-Infrared image', ha='center',fontsize = 16)

    return image_S2, equalized_image_R, equalized_image_G, equalized_image_B, img_RGB, img_NIR
```

Step 1: Creating Sentinel images

```python
def groundtruthgenerator(image, tol):
    ground_truth = image > tol
    ground_truth[:,image.shape[1]-1] = 0

    return ground_truth
```

Step 2: Ground Truth extraction

```python
def tophat(image,radius):
    image_gray = rgb2gray(image)
    footprint = morphology.disk(radius)
    res = morphology.white_tophat(image_gray, footprint)
    tophat_filtered_image = image_gray - res

    return tophat_filtered_image

def unsharp_masking(image, radius, amplification):
    masked_image = unsharp_mask(image, radius, amplification)

    return masked_image
```

Step 3: Image Enhancement

```python
def vectorlayer_generation(image, ground_truth, AOI, th, do_plot, stats):
    if th == 'auto':
        threshold = find_threshold(0.8, 50, image, ground_truth, 'Sobel',
    1,0)[0]
    else:
        threshold = th

    print('The selected pixel threshold is {}'.format(threshold))
    edges = edge_extraction(image, threshold)

    if stats:
        noise, noise_filtr, qual_edges = edge_detection(image,
   ground_truth, 'Sobel', threshold, 0, 1, 0)

    m,n = image.shape
    top, left = AOI[0], AOI[1]
    resolution = 10
    edge_layer_hor, edge_layer_vert = [],  []

    for i in range(m):
        for j in range(n):
            if edges[i,j] == True:
                coord = (top-resolution*i, left+resolution*j)
                edge_layer_hor.append(left+resolution*j)
                edge_layer_vert.append(top-resolution*i)

    if do_plot:
        fig, ax = plt.subplots(1,2, figsize=(15,10))

```

```
        ax[0].imshow(image, cmap='gray')
        ax[0].imshow(edges, alpha=0.8, aspect="auto")
        ax[0].set_title('Detected edges', fontsize = 16)

        ax[1].plot(edge_layer_hor,edge_layer_vert,'o')
        ax[1].set_title('Road Layer vector', fontsize = 16)
        ax[1].set_xlabel('UTM zone 31N coordinates [m]', fontsize = 14)
        ax[1].set_ylabel('UTM zone 31N coordinates [m]', fontsize = 14)
        ax[1].ticklabel_format(useOffset=False, style='plain')
        ax[1].xaxis.set_tick_params(labelsize=12)
        ax[1].yaxis.set_tick_params(labelsize=12)
        fig.tight_layout(pad=5.0)

    if stats:
        return edges, edge_layer_hor, edge_layer_vert, noise, noise_filtr
    , qual_edges
    else:
        return edges, edge_layer_hor, edge_layer_vert
```

Step 4: Generate Road Vector Layer

```

def edge_algorithm(AOI_longlat, AOI_31N, ground_truth_thresh, date_start,
    date_stop, th, do_plot, stats):

    # Initialize AOI in correct format
    maxLat_31N =  AOI_31N[0]
    minLon_31N = AOI_31N[1]
    minLat_31N =  AOI_31N[2]
    maxlon_31N =  AOI_31N[3]

    AOI_31N_extract = Polygon([(minLon_31N, maxLat_31N),
                               (minLon_31N, minLat_31N),
                               (maxlon_31N, minLat_31N),
                               (maxlon_31N, maxLat_31N)])

    # Create Sentinel Images corresponding to AOI
    S1 = extract_S1_image(date_start, date_stop, AOI_longlat,
   AOI_31N_extract, do_plot)
    S2_RGB = extract_S2_image(date_start, date_stop, AOI_longlat,
   AOI_31N_extract, do_plot)[4]

    # Extract ground truth
    ground_truth = groundtruthgenerator(rgb2gray(S2_RGB),
   ground_truth_thresh)

    # Enhance Image
    tophat_image = tophat(S2_RGB, 1)
    unsharp_mask = unsharp_masking(tophat_image, 1, 1.5)
    image = unsharp_mask

    # Generate Road Vector Layer
```

```
    if stats:
        edges, hor, vert, noise, noise_filtr, qual_edges = vectorlayer_generation(image, ground_truth, AOI_31N, th, do_plot, stats)
    else:
        edges, hor, vert = vectorlayer_generation(image, ground_truth, AOI_31N, th, do_plot, stats)

    if stats:
        return edges, S2_RGB, image, hor, vert, noise, noise_filtr, qual_edges
    else:
        return edges, S2_RGB, image, hor, vert
```

Algorithm 1: Edge detection Algorithm

```
def contours_algorithm(AOI_longlat, AOI_31N, ground_truth_thresh, date_start, date_stop, cval, tol_loop, do_plot):

    # Initialize AOI in correct format
    maxLat_31N =  AOI_31N[0]
    minLon_31N = AOI_31N[1]
    minLat_31N =  AOI_31N[2]
    maxlon_31N =  AOI_31N[3]

    AOI_31N_extract = Polygon([(minLon_31N, maxLat_31N),
                               (minLon_31N, minLat_31N),
                               (maxlon_31N, minLat_31N),
                               (maxlon_31N, maxLat_31N)])

    # Create Sentinel Images corresponding to AOI
    S2_RGB = extract_S2_image(date_start, date_stop, AOI_longlat, AOI_31N_extract, do_plot)[4]

    # Extract ground truth
    ground_truth = groundtruthgenerator(rgb2gray(S2_RGB), ground_truth_thresh)

    # Enhance Image
    image = tophat(S2_RGB, 1)

    contours = measure.find_contours(image, cval)
    contours = loopdetection(contours,tol_loop,20)

    if do_plot:
        fig, ax = plt.subplots(1,1,figsize=(12,12))
        ax.imshow(image, cmap=plt.cm.gray)
        for contour in contours:
            ax.plot(contour[:, 1], contour[:, 0], linewidth=2)

        plt.title('Contour finding, c = {}'.format(cval),fontsize = 16)
        plt.show()
```

```

    return contours
```

Algorithm 2: Contour Finding Algorithm

References

[1] *Terrascope Documentation Portal*. URL: https://docs.terrascope.be/#/DataProducts/Sentinel-1/ProductsOverview?id=sentinel-1-data-products.

[2] Stéfan van der Walt et al. "scikit-image: image processing in Python". In: *PeerJ* 2 (June 2014), e453. ISSN: 2167-8359. DOI: 10.7717/peerj.453. URL: https://doi.org/10.7717/peerj.453.

[3] *Top-hat transform*. Aug. 2022. URL: https://en.wikipedia.org/wiki/Top-hat_transform.

[4] URL: https://scikit-image.org/docs/dev/auto_examples/filters/plot_unsharp_mask.html.

[5] *Contour finding*. URL: https://scikit-image.org/docs/dev/auto_examples/edges/plot_contours.html#contour-finding.

[6] *Edge operators*. URL: https://scikit-image.org/docs/dev/auto_examples/edges/plot_edge_filter.html.

[7] *Roberts Cross*. Nov. 2022. URL: https://en.wikipedia.org/wiki/Roberts_cross.

[8] *Prewitt operator*. Nov. 2022. URL: https://en.wikipedia.org/wiki/Prewitt_operator.

[9] *Sobel operator*. Nov. 2022. URL: https://en.wikipedia.org/wiki/Sobel_operator.

[10] *Skeletonize*. URL: https://scikit-image.org/docs/stable/auto_examples/edges/plot_skeleton.html.

[11] *Canny edge detector*. Oct. 2022. URL: https://en.wikipedia.org/wiki/Canny_edge_detector.

[12] *Ridge operators*. URL: https://scikit-image.org/docs/dev/auto_examples/edges/plot_ridge_filter.html#sphx-glr-auto-examples-edges-plot-ridge-filter-py.