# Extraction of urban features with knowledge based engineering and local maxima filtering

## Demonstrated for the city of Trier

## Masterarbeit

Fachbereich VI: Umweltfernerkundung und Geoinformatik



| | |
|---|---|
| Autor: | Andreas Löwe |
| Fach: | Angewandte Geoinformatik |
| Matrikelnummer: | 1002405 |
| Erstgutachter: | Prof. Dr. Thomas Udelhoven |
| Zweitgutachter: | Dr. Henning Buddenbaum |


Trier, 2015

## Abstract

The aim of this study is creating a tool, which has the ability to incorporate different types of remotely sensed data in such a way, that not only buildings but also tree tops can be extracted from it. Another aim is to extract the desired results with an automated approach. Numerous algorithms are created with the help of the programming language *Python*. The first step is using a knowledge based engineering to identify buildings and tree tops. The next step is to enhance the data with the help of binary filters, which are used to reduce the noise and refill missing data points. To extract the building the raster data are converted into vectors and then structures are removed, which don't fulfil user set criteria. The tree tops are located by a local maxima filtering and the final product is created by applying rules for the tree tops. It can be demonstrated, that the success mainly depends on the careful selected values for the user set parameters.

## Zusammenfassung

Das Ziel der Arbeit ist es ein Tool zu erarbeiten, das in der Lage ist, unterschiedliche Fernerkundungsdaten in der Weise zu verarbeiten, dass aus den Ergebnissen nicht nur Gebäudestrukturen erkannt werden können, sondern auch die Positionen einzelner Bäume. Dies nicht nur zu erreichen, sondern auch zu automatisieren ist ein weiteres Ziel der Arbeit. Dabei werden mithilfe der Programmiersprache *Python* verschiedene Funktionen umgesetzt. Dabei werden wissensbasierte Auswahlkriterien nicht nur für die Gebäude, sondern auch für die Bäume angewendet. Der nächste Schritt besteht in der Verarbeitung der Daten mit Filtern. Diese sollen neben der Verminderung des Rauschens auch fehlende Bildteile rekonstruieren. Für die Erstellung der Bauten werden die Rasterdaten vektorisiert und fehlerhafte Strukturen, die eine zu kleine Fläche aufweisen, um Gebäude zu sein, entfernt. Die Bäumespitzen werden durch einen lokale Maxima Filter identifiziert und nach mehreren Kriterien ausgewählt. Es zeigt sich, dass der Erfolg maßgeblich von der Auswahl der kritischen Werte Abhängt, um ein genaues Ergebniss zu bekommen.

# Table of Content

**Abstract** I

**List of Figures** III

**List of Tables** IV

**List of Listings** V

**1. Introduction** 1
- 1.1. Literature . . . 1
- 1.2. Motivation . . . 2
- 1.3. Aims and Hypotheses . . . 3
- 1.4. Structure of the work . . . 4
- 1.5. Study area . . . 5
- 1.6. Choosing the test areas . . . 6

**2. Theory** 9
- 2.1. Knowledge based engineering . . . 9
- 2.2. Filter algorithms . . . 10
- 2.3. Local maximum filtering . . . 12

**3. Methods** 14
- 3.1. Involved Data . . . 14
- 3.2. Preprocessing . . . 16
  - 3.2.1. Creating of digital elevation models . . . 16
  - 3.2.2. Creating the NDVI images . . . 17
- 3.3. The Program - An Overview . . . 18
  - 3.3.1. The Main window . . . 20
  - 3.3.2. The Building Options . . . 22
  - 3.3.3. The Tree Options . . . 25
  - 3.3.4. Handling the processing chain . . . 29

**4. Results** 31
- 4.1. Results for the district Altstadt . . . 32
- 4.2. Results for the district Feyen . . . 36
- 4.3. Results for the district Gartenfeld . . . 38
- 4.4. Results for the district Maximin . . . 40

4.5. Results for the district Pallien . . . . . 43
4.6. Results for the district Trier West . . . . . 46

**5. Discussion** **49**
5.1. Evaluation of the Program . . . . . 49
5.2. Evaluating the Buildings . . . . . 51
5.3. Evaluating the Tree Positions . . . . . 54

**6. Conclusions** **56**

**Bibliography** **58**

**Declaration of authorship** **61**

**A. Appendix** **i**

# List of Figures

1.1. The extend of the data . . . . . 6
1.2. Chosen test areas in Trier . . . . . 7

2.1. The functionality of erosion filter . . . . . 11
2.2. A comparison of two filters . . . . . 11
2.3. The functionality of fill holes filter . . . . . 12

3.1. The data used to test the program . . . . . 15
3.2. The workflow of the program . . . . . 16
3.3. The start window . . . . . 20
3.4. Comparing the building filter window . . . . . 22
3.5. Comparing the tree filter window . . . . . 25
3.6. Comparing the impact of changing the allowed no data values . . . . 27
3.7. Comparing the impact of changing the kernel size . . . . . 28

4.1. Results for the District of Altstadt . . . . . 33
4.2. Details of the District of Altstadt . . . . . 34
4.3. Evaluation results of the district Altstadt . . . . . 35
4.4. Evaluation of 42 tree positions with ground truth data . . . . . 36
4.5. Results for the District of Feyen . . . . . 37
4.6. Details of the District of Feyen . . . . . 38
4.7. Results for the District of Gartenfeld . . . . . 39
4.8. Details of the District of Gartenfeld . . . . . 40
4.9. Evaluation of the district Gartenfeld . . . . . 41
4.10. Results for the District of Maximin . . . . . 42
4.11. Details of the District of Maximin . . . . . 43
4.12. Results for the District of Pallien . . . . . 44
4.13. Details of the District of Pallien . . . . . 45
4.14. Evaluation of the district Pallien . . . . . 46
4.15. Results for the District of Trier West . . . . . 47
4.16. Details of the District of Trier West . . . . . 48

# List of Tables

1.1. Comparison of four cities . . . . . 5
1.2. Overview of the six districts . . . . . 8

3.1. Imported modules from external *Python* libraries . . . . . 18
3.2. Modules created for *CityEX* . . . . . 19
3.3. Created *Python* modules and their inputs . . . . . 30

4.1. Options for creating buildings . . . . . 31
4.2. Options for creating the tree tops . . . . . 32

5.1. Evaluation of the results by different metrics . . . . . 52

# List of Listings

2.1. Example of how a local maximum filter works . . . . . 13

3.1. Excerpt from the function *validate_inputs* . . . . . 21
3.2. The Add and Delete functions . . . . . 23
3.3. Function to handle the filter settings . . . . . 24
3.4. Function to handle the smoothing . . . . . 26
3.5. Handling the critical height parameter . . . . . 29

# 1. Introduction

The availability of high resolution remotely sensed data increased over the last few years. The same holds true for products created by *LIDAR* (light detection and ranging). Thus, the fields in which those technologies can be used, increased as well. This leads to the thesis, which will be presented here. The aim is to create a reliable model of an urban environment, which contains buildings as well as green objects.
This chapter will introduce the reader into the work and gives an overview of the upcoming sections, the aims, the theory as well as the results and the final discussion, which leads to the conclusions drawn from the results. While this chapter aims to point out the significance of the work (see 1.2 and 1.3), the following chapters will present the main work containing the methods.

## 1.1. Literature

Automated detection of man-made objects such as buildings from different types of data are subject of many studies. They aim to understand the scene and thus segment objects upon this understanding. As numerous as the amount of studies, the methods to extract the desired objects varying in a great deal. The first distinction can be made between studies, which only uses one type of data, as demonstrated by Cao et al. and Attarzadeh and Momeni. They only used aerial images to extract man-made objects. The other kind of data used are digital surface models with different shaping. Those models are extracted from *LIDAR* datasets and either the raw points are used, as the study from Awrangjeb et al. suggests, or the data are transformed into surface models as shown by Huang et al.. This author also used ancillary data such as Normalized Difference Vegetation Index (*NDVI*), which find use in many other studies. This is the second branch of extracting objects, by using a height model and additional data. Rottensteiner et al. attempted to extract buildings by fusing *LIDAR* and aerial images (Rottensteiner et al. 2007). The last data driven approach is to use multitemporal data. This is the way, which Tigges et al. chose, when they classified urban vegetation (Tigges et al. 2013).
While the earlier approaches demonstrate, how the data present influence the way, how a study is conducted, this part will show the different methodology approaches, to derive either urban structures or trees. Again the choice of data is a driving force

for settling with a appropriated approach. On way to get the desired models is to derive from certain objects. Attarzadeh and Momeni derived the objects from stable features, which are inherent characteristics of building, and variable features, marked with the their associated thresholds and separability (Attarzadeh and Momeni 2012). That the approach for estimating urban green with an object driven method works as well, which is proven by Huang et al.. While the focus for the study of Johnson and Xie lays on the extraction of building, the algorithm is capable of extracting trees as well. One approach wildly used is the one proposed by Sohn and Dowman. The study from Al Momani et al. demonstrate its use for segmentation objects in urban areas.
The next way, of extracting men-made objects is with prior knowledge of the environment. The functionality of this approach is proven by Bouziani et al..
A paper published by Awrangjeb et al. demonstrate, how buildings can be extracted with a *NDVI* image, two building masks and a *YIQ* image by using both, knowledge and object based approaches (Awrangjeb et al. 2010). The *YIQ* image contains the intensity, the hue and the saturation.
All the studies have in common,that they seek to solve the problems with an automatic approach. It also become clear, that the field of remote sensing can have a large impact on urban studies. It has the potential to improve decision making processes by automating computation of data and thus reduce the time of evaluating decisions made earlier. The next section will point out the need for those attempts.

## 1.2. Motivation

The main motivation is drawn from the fact, that a the author has a strong background the field of urban planing. Thus the aim was always to create a thesis, which combines the fields of geoinformatics and urban planing. Because of this aim, the present work deals with the creation of buildings derived from remotely sensed data and *LIDAR*. To complete a model of an urban environment, the extraction of tree position is another goal. The reasoning is, that they have a strong impact on the value of a neighbourhood and thus influence the attractiveness of a district.
The data density is high in developed countries, which is in contrast to many developing countries, where the availability of consistent data is a huge problem. Not only are cities developing in a higher pace, but also the growing has a high dynamic here. Due to the lack of planing tools or the implementation of those tools, it is hard to map those areas and thus the process of making decisions is flawed. Informal settlements can be seen as an good example of such struggles. They appear fast, have a high dynamic due to in- and out-fluxes of people and the positions of housing or streets can change because the materials used are not permanent. So if a government

has the aim of supplying those settlements with infrastructure, then a plan has to be available to plan the routes of pipes for example, which is not only precise but also created recently, in order to avoid flaws. Those aims can be achieved by the means of remote sensing. Thus the aims and hypotheses from section 1.3 are leant on those ideas and the thesis try to create those tools needed.

## 1.3. Aims and Hypotheses

The author aims to achieve two main goals. First, to create an application capable of computing building models and determining the positions of tree tops, and second to figure out, in what extend the density, the overall greenness and the amount of trees in a specific districts influences the value of the property in those districts. While the first task is solely for compressing the different scripts into one handy program, the later task is meant to determine the hypotheses for this work.
The idea is, to derive parameters for defining an urban area. To be more precise, to discriminate against different types of cities. This parameters can be in the form of population parameters such as density or age structure. Cities can also be separated by means of economic reasoning. Such as the mean net income, the development of housing prices as well as the prices paid for rents. Those parameters are derived from statistics an can be measured easily and thus are often applied (Qihao 2010; Weng 2007). From the point of the geoinformatics, data derived from spatial sources such as satellites, flight campaigns or field measurements, can be used to create models of cities, which measure physical parameters rather than statistical ones. When both data-sources are combined, one can make better assumptions and decisions as from one kind of data alone (Weng 2007).
The first goals is, to derive meaningful parameters from both domains (statistics as well as remote sensing). So the overall hypothesis is, if one can discriminate against different cities with the means of remote sensing. The question, which follows is, what parameters can represent an urban area and therefore a city. The choice of parameters is limited by the availability of data, which has a spatial as well as a temporal aspect and plays a key role, so this point has to be fairly considered. The next aim is to chose a suited target area, which leads to the hypotheses, that an area suitable for testing the overall hypotheses has to have a certain grade of urbanity and a broad availability of data. So the question is, which areas fulfil those requirements and which data can be used.
The university provides data from the ‟Landesvermessungs Amt Rhineland-Palatinate″, which includes aerial photographies and *LIDAR* data. Thus those two types of data were used. From this point the question is, what products can be derived from those

data in order to create a urban model. The author's choice was made for the buildings and tree, because they are key elements in an urban area and can be extracted from the data. This decision is backed by many studies, as proven by the previous section 1.1. The next question is, what characteristics the data demonstrate, to extract building and trees from them. The key characteristics can be cell size of an image or the temporal and spatial consistency of the data. This completes the background of this work concerning the data. The next step is to derive the parameters of the urban area and decide, what meaning they have. So the question is, how to derive the parameters and with which methods this is accomplished. Then the created results have to be interpreted, which leads to the final hypotheses, that the amount of trees and the share of building from the whole district are essential for an urban region. To summarize the section, the hypotheses are listed here as follows:

1. Cities can be discriminate against with the means of remote sensing
2. There are statistical and physical variables, which can determine the extend of a city
3. There are methods to distinguish buildings and trees from other urban objects
4. Differences between districts can be determined by means of remote sensing
5. Creating a program capable of handling all the options and inputs

With all those questions in mind, the structure of the work becomes obvious and is shown in detail in the next section 1.4. The evaluation of the hypotheses is conducted in the final chapter 6.

## 1.4. Structure of the work

The work is parted into five main chapters, which all deal with one aspect of the work. The first chapter *Introduction* gives the reader an inside into the aims, a review of the literature in this field as well as the current state of science. Then the study area is laid out to set the place. In this part the reasoning why the specific area and, in more detail, why the districts are chosen, is presented. The next chapter explains the theories which are used and combined to achieve the extraction of building from the data. The chapter *Methods* present then the ways, in which the those theories are put to work. In this part of the work, the program, which is created for this thesis is explained as well. The details of the different steps are shown and examples are given. From here the chapter 4 *Results* is a logic step. Here the author explains the outputs and gives a detailed explanation, how the results were created and

what the reasoning is behind. The next chapter deals then with the discussion and evaluation of the results while showing problems in the processing chain. Finally the last chapter deals with the hypotheses as well as proposing future developments and improvements.

## 1.5. Study area

Depending on the structure, the geographic complexity and thus the diversity of the area, the quality of the outcomes can have a great variation. With this in mind, The decision of choosing the test area is an important task. The test area for this thesis is the city of Trier, which is situated in the western part of Rhineland-Palatinate. The river, which flows through the city is the Moselle river. he city is famous for its wine and is regarded as the oldest settlement in Germany with its roughly 2000 years of existence (Jätzold 1984). Its centre is at 49°45´20″ North and 6°38´21″ East and has a elevation above sea level of 141 meters. The Eifel borders Trier in the north, while the Hunsrück mountains are situated to the south-east. The Saarland, with its capital Saarbrucken, which is roughly 80 km to the south borders Trier as well. West of Trier the Grand Duchy of Luxembourg can be found. Its capital Luxemburg city is roundabout 50 km apart from Trier (Jätzold 1984).
Trier itself has an area of 117,13 sq km with a population of 106.680, which are split up among the 19 districts. The number is from the 31. of December 2013 (Trier 2015a, online). From this numbers, we can derive a population density of 910,783 inhabitants per square kilometre.
By comparing it with the density of Germany, where 80,767 million people life on an area of 357.169 square kilometres, which results in 226,0223 people per square kilometre (Statistisches Bundesamt Deutschland 2014), a better perceptive can be made. It shows, that the city of Trier is densely populated. By comparing Trier to Koblenz, Kaiserslautern and Mainz in following table 1.1, it can put into a more detailed context, because the three other cities are also situated in Rhineland Palatinate. All the dates are derived from Statistisches Bundesamt Deutschland and dates back to the 31. December 2013.

| **City** | **Population** | **Area in sq km** | **People per sq km** |
|---|---|---|---|
| Trier | 106.680 | 117,13 | 910 |
| Mainz | 204.268 | 97,76 | 2089 |
| Koblenz | 110.643 | 105,13 | 1052 |
| Kaiserslautern | 97.162 | 139,72 | 695 |

Table 1.1.: A comparison of four cities in from Rhineland Palatinate

It shows, that the four cities has similar properties regarding their area, the inhabitants and therefore have a comparable population density. The only city standing out is Mainz. It has almost twice as many inhabitants as the other three, which is reflected by a very high density of inhabitants.

## 1.6. Choosing the test areas

After introducing the general test side and show it in it's geographic context in section 1.5, this part shows the districts, which are chosen to help determine, if the hypotheses from section 1.3 are valid. This is done in three steps.

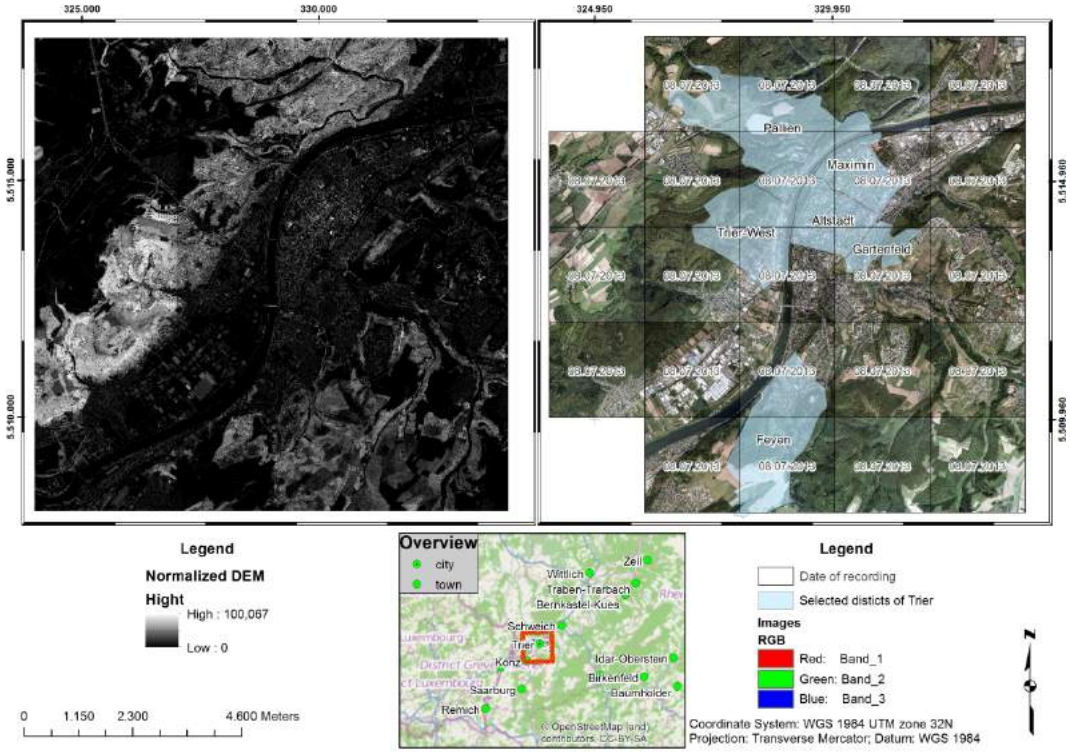



Figure 1.1.: The extend of the data

The first step is to evaluate the existing data and their geographic extend. The quality of the data will be topic of the section 3.1. The map in figure 1.1 illustrate the two kinds of data involved and the area, which is covered by the data. It can be seen, that the western districts like Tarforst or Filsch are missing. This has to be taken account of, when the areas to test the work flow and the quality of the approach are chosen. The result is, that of the 12 districts only 17 are covered completely by the data and thus can be selected as test area.
The second step is to attach meaningful data to the districts, which supports the

hypotheses. In this case the amount of people per square kilometre living in the area are calculated respectively the mean price for each square metre.
The third step consists of choosing the test sides. Out of the 17 potential areas, six are picked. Two with low values for inhabitants per square kilometre and mean price per square meter, two with medium values around the mean of the whole group and two with high values. Reflecting the ideas from part 1.3, the point of choosing the test sides in this way is to determine, if certain parameters like green volume, number of trees or the relationship between green and urban areas differ significantly.

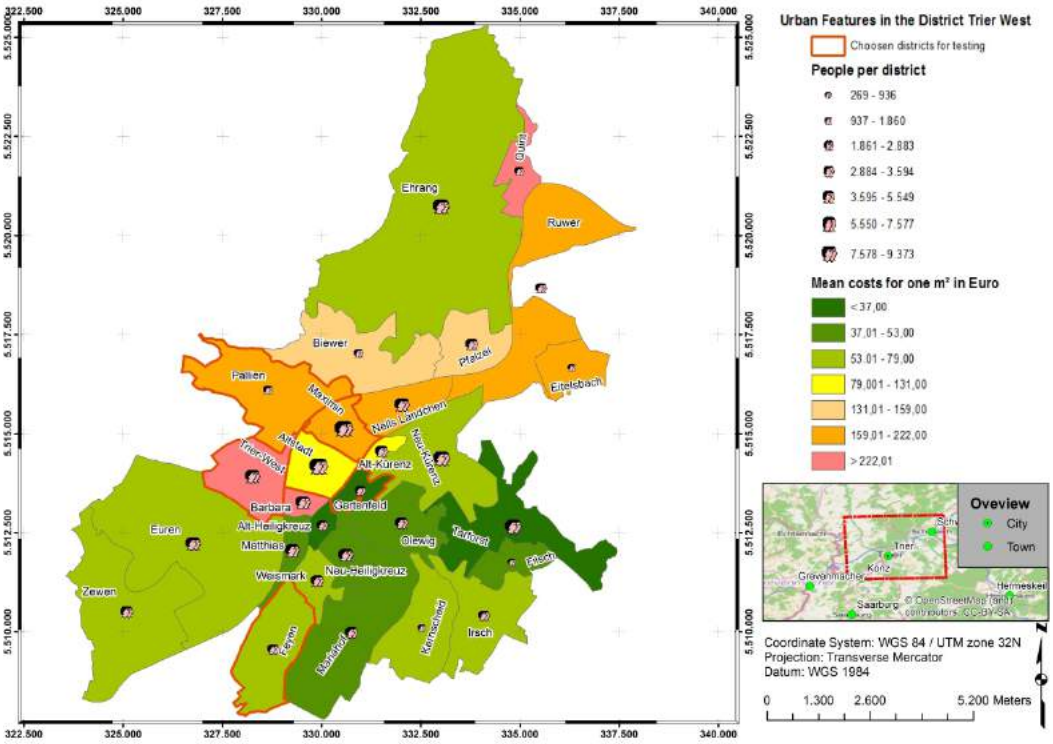



Figure 1.2.: Mean population per square kilometre(size of the symbol) and mean price per square metre in Euro(colours). The orange lines represent the test areas

As seen in figure 1.2, the six test sides are spread around the city of Trier. The test sides for the low values are *"Pallien"* situated at the north-west side and *"Feyen"* in the opposite direction(straight south). The districts for the high values are *"Altstadt"* and *"Maximin"*, which are both situated at the core of the city. The test sides for the results around the mean are *"Gartenfeld"*, being in the eastern part of the centre and *"Trier-West"*, which is in the western part of the city. The table 1.2 will sum up the results from the map and gives a straight forward overview about the six areas. As a new information, which is introduced in the table, the area of each district is displayed. We can see, that the two regions witch are sparsely populated, are the ones with the largest area. In contrast, the two densely populated districts Alstadt

| **District** | **Mean ground price** | **Population** | **Area in $km^2$** | **People per $km^2$** |
|---|---|---|---|---|
| Altstadt | 234,705 € | 9.373 | 2,022 | 4635,509 |
| Fayen | 746,205 € | 2.731 | 3,999 | 682,921 |
| Gartenfeld | 361,593 € | 2.883 | 0,913 | 3157,722 |
| Maximin | 200,358 € | 9.114 | 1,673 | 5447,699 |
| Pallien | 375,141 € | 1.504 | 5,361 | 280,545 |
| Trier-West | 258,785 € | 5.549 | 3,104 | 1787,693 |

Table 1.2.: An overview for statistical data of the six test districts

and Maximin, are the ones which have a small area, resulting in a high density. It can be concluded, that the later districts have a more urban structure, than the first two. We can support this hypotheses by having a look at figure 3.1 or 1.1. Both images are showing a high amount of green (in figure 3.1 the *NDVI* image in figure 1.1 the RGB part).

# 2. Theory

The subject of the following chapter is to clarify the essential concepts of this work. It will be discussed in detail, which algorithms were used and how they work. Later sections (see section 3.3) will show how the theory comes to work and how the principals were incorporated to generate the desired data.
The first section will deal with the knowledge engineering of data, while the second will show how filter algorithms work. The last section will focus on the principals of another kind of filter, to demonstrate the difference of those.

## 2.1. Knowledge based engineering

This theory is the base of this work. From here, the steps are evolving to form the program which will be discussed in section 3.3.
At first the step of knowledge engineering is to organize the former unstructured data. With the help of knowledge provided by experts (people how have a strong understanding of the specific task and environment) one can develop a concept. This include the following task according to Czap and Nedobity:

- Strategic insight of the data to plan the tasks and to set the order of each operation
- Creation of the tasks and setting their purpose
- Making inferences by using theories and hypotheses
- Make assumptions about the objects to be identified
- Apply the rules extracted from the knowledge
- Validate the results and update the assumptions and inferences

The list sums up the whole process of systematic knowledge generation. Now a deeper look into the detailed steps from the bottom up is presented.
The basic element of knowledge is a rule. A rule in this context is defined as basic mathematical operation. Combining rules leads to assumptions, which can support the hypotheses about the objects. The strategic level deals with the theories and

the used data. The decision, which kind of data will be used is made on this level. The first outcome will be examined for it's accuracy and if need, the rules and assumptions will be updated, meaning a circle of validation is embedded into the knowledge base. This ensures, that rules are improved and therefore the results are more precise.
Setting this basic understanding of knowledge engineering into the context of remote sensing, the strategic view and the structure remains the same while the aim is to identify objects which fulfil certain criteria namely spectral, shape, textural and contextual properties of those objects (Tiwari and Pande 2011). This is carried out by having a prior knowledge about the desired objects in the landscape. In the sense of geospatial data this can be the amount of trees to estimate the number of groups or parameters like hight, the crown diameter or the hight of the trees (Heinzel et al. 2011; Niccolai et al. 2010; Huang et al. 2008). Another application is the use of a priori knowledge to classify images as demonstrated by Al Momani et al..
In conclusion, knowledge engineering is an easy to use tool to identify objects in terms of spatial information. Combining it with other tools (see 2.2) can amplify its use further regarding accuracy and usability. By evaluating the results and thus improving them the accuracy of this method can be further amplified.

## 2.2. Filter algorithms

The next theoretical approach is to show the power of filters. The downsides of them will be discussed in the chapter 5, while this segment of the work will deal with the filters used in the work (see 3.3) and their functionality. All the filters in this context have in common, that they work in a 2D environment. This kind of filters are summarized as morphological filter. Their functionality is based on the characteristics of the shape of features in an image.
The research shows, that filters are widely used in various fields of study. Rottensteiner et al. uses a hierarchic morphologic filter to create digital surface models (Rottensteiner et al. 2007), as well as Meng et al., who uses a ground-filtering algorithm to separates ground pixels from buildings, trees, and other objects (Rottensteiner et al. 2007; Meng et al. 2009).
The idea is, that the filter steps through the image and with a structuring element, which is altered according to the filter used. This means in summary, that a morphological filter is applied to the neighbourhood of a pixel.The data structure has to be binary (either as *True* or *False* values or as 0 or 1) in order to have the ability to use the filter. An application of is to remove noise in an image (Heilbronner and Barrett 2014, p. 95f).
The first filter discussed in this section is erosion filter. As demonstrated in figure

2.1, the erosion works with a moving window, which slides over the image. Depending on the size of the structure element or so called kernel, the image is altered in such a way, that all pixels of the foreground are set to the background, if in their neighbourhood are a defined amount of non foreground pixel. For the figure 2.1 the kernel is set to three. Meaning, that every pixel $p_{ik}$ is removed from the foreground, if less than or equal to three surrounding pixels are belonging to the foreground (Heilbronner and Barrett 2014, p. 98f). As demonstrated by Huang et al. an erosion filter can be applied to remove noise from an image (Huang et al. 2008).
The dilation on the other side works the other way around. This filter adds a pixel if its neighbours have a user defined amount of foreground pixels (Heilbronner and Barrett 2014, p. 98f). Kim and Muller uses it in a repeated filter approach to split tree crowns (Kim and Muller 2011). This form of use indicates, that the filter can be

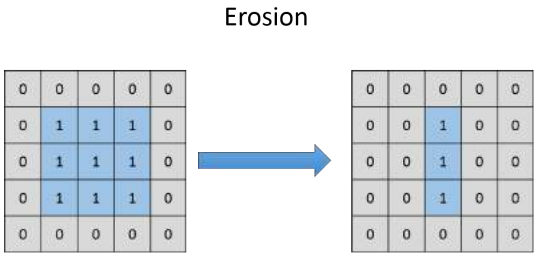
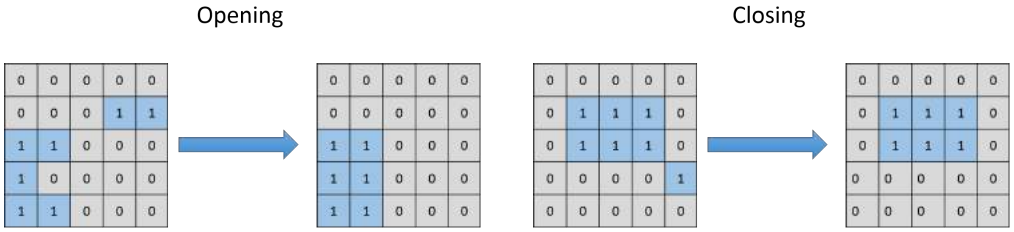



Figure 2.1.: The functionality of erosion filter

applied to remove noise in an image if needed, because it has the ability to remove small junks of foreground pixels. Setting this in the relation to remote sensing, it can be shown, that this two filters can refine classification results.
The next two examples filters will be a binary closing and opening. The figure 2.2 illustrate, how this filters are applied to the data. They have in common, that both make use of the dilation and the erosion filters, which were explained in the previous part of this section. The difference is, that they use a different order of the filters. While the opening filter uses first the dilation and than the erosion, the closing filter handle it vis versa. newline




(a) The functionality of opening filter (b) The functionality of closing filter

Figure 2.2.: A comparison of two filters

The last filter presented in this section is the fill holes filter. As shown in figure 2.3 the filter has looks for holes in the data. As the three other filters, the fill hole filter uses a moving window. It also has a kernel with an user defined size. Whenever the centre is surrounded by foreground pixels, and its belongs to the background, then it will become part of the foreground. This works not only for a neighbourhood of eight but also for more (meaning that a larger kernel than a $3x3$ is possible).

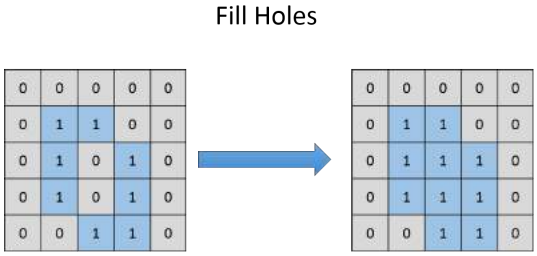



Figure 2.3.: The functionality of fill holes filter

## 2.3. Local maximum filtering

In contrast to the filers presented in the previous section 2.2, the filter, which will be explained in this section, works on numerical instead of binary data.
Different approaches are made in the literature. One is an local maxima filter, which alters the window size according to the structure inside the window to identify single trees (Chen et al. 2006). A similar idea is discussed from Monnet et al.. The author first smooths the data derived from a point cloud, and chooses then different window sizes to determine the final maxima points (Monnet et al. 2010). the approach from Huang et al. is similar to the one used in this work. The author first creates a *nDEM*, which is filtered, then computes a normalized vegetation model, from which he extracts the tree tops via local maxima filters (Huang et al. 2013).
The local maximum filter makes use of the fact, that the difference in the values can be an indicator for certain phenomena. It can be viewed as a window, which is moving over the image with a user defined size. It checks, if the value of the pixel $p_{ik}$ is the maximum for the pixels in the earlier defined window, here called $n$, which includes also the pixel $p_{ik}$.

$$p_{ik} ismax(n)$$

With this concept, there are some problems, if combined with algorithms to smooth the data, like suggested from Niccolai et al. or Tigges et al., the smoothing can, if used too much, result in more than one maximum inside the window (Niccolai et al.

2010; Tigges et al. 2013). The same can happen, if the window size is chosen too large and thus $n$ is increased. Even if this two parameters ar set with reasonable values, there is a chance, that two or more pixels inside the window have the same value. The listing 2.1 explains, how the theory is translated into a small code example in *Python*.

```
1 import numpy as np
2
3 artificial_img = np.random.rand(20,20)
4 window_size = 5
5
6 for start in xrange(0, artificial_img.shape[1], window_size):
7   data = artifical_img[start:start + window_size, start:start + window_size]
8   print data.max()
```

Listing 2.1: Example of how a local maximum filter works

The first step is creating an artificial array, which contains a set of random numbers and represents an image. Those values could stand for the height or the colour captured with aircraft. Then the *for-loop* iterates over the image with a window size of five, which is set by the variable *window_size*. The subset is saved in the variable *data*. The loop is created in such a way, that it steps through the image, without the different subsets intersecting with each other. The last step is to identify the maximum point in the subset, which is carried out by the *.max()* function of the *numpy*-library. The result is a set of four local maxima. They are the maximum value in their respective neighbourhood.
When dealing with air-born taken images, the data-format *.tif* will be encountered. A python library to deal with this kind of data is *OGR*. With the subpackage *GDAL*, a powerful tool to alter various data-formats is present. How the different tools are incorporated into the work will be discussed in detail in section 3.3.
Concluding those findings, the filter has a good potential of detecting various objects. The later chapters will demonstrate, how this theory is incorporated into the program, and how the filter is used to detect tree tops in an urban environment, by manipulating and optimizing the presented theory.

# 3. Methods

This chapter will show in detail, what data were used and what kind of methods where applied to produce the results, which are created by the final application. While at first the inputs will be lined out, the second step will be the evolution of these inputs to the final product. The last step is the explanation on how these steps are arranged in the application called *CityEX*, which will be discussed in detail in this chapter.

## 3.1. Involved Data

To conduct the thesis, two kind of data were used. First 504 files in the form of *.xzy*, which represents the returns from *LIDAR* and were taken by a flight campaign during 2013. The data differ in such a form, that the files ending with *GRD* containing the last return and therefore are the ground points only. On the other hand, the files ending with *VEF* are the first return points. Thus they are standing for the points created by man-made objects and tops or branches of trees. Each file has a size of 80 Mb to about 220 Mb, depending on how many returns were captured.
The other set of input data are 32 airborne taken images. They have four channels. Three for the visible light part of the image and one for the infra-red part. The image has a panchromatic size of 20.010 times 13.080 pixel with a physical pixel size of 5.2 $\mu m$ . The colour image has 6.670 times 4.360 pixels as image size with the same value for the physical pixel size. With this outline, the images are feasible of creating different indices such as *NDVI* or *PVI*, which are used as a support to the *.xyz* data. All of the used images were recorded on the 8th of July 2013. This provides a consistency throughout the all the products which will be produced with the program. On the other hand, the *.xyz* files were taken on the 8th of July 2013. This can result in errors, which will be discussed in chapter 5.
As mentioned in the section 1.5 just six out of the 19 districts are chosen. With this decision a data reduction can be conducted. Due to the fact, that out of the 32 aerial images, just 23 are needed to picture those districts, a reduction from 504 *.xyz* files to 184 can be achieved. This ensures, that the processes can be tested for their accuracy, while the whole process chain is not extremely time consuming.

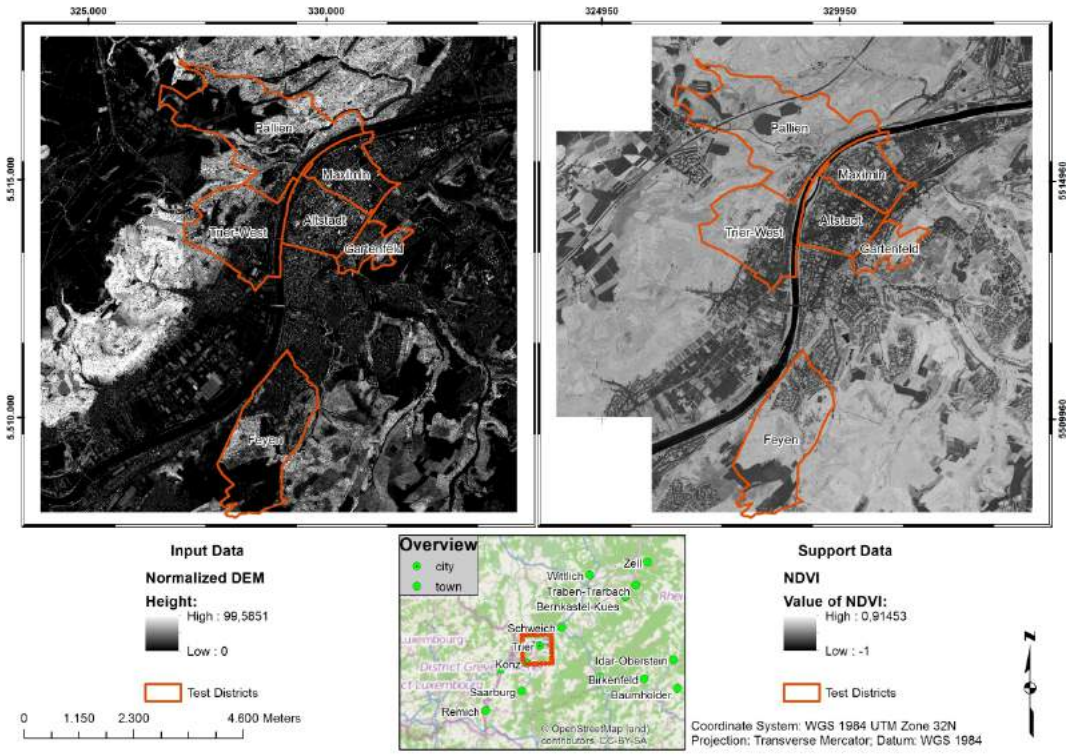



Figure 3.1.: The data used to test the program, left: *nDEM*, right: *NDVI* orange: chosen test areas

The figure 3.1 will show the dimensions of the data and the geographic extend. Thus the position of the chosen test areas in section 1.6 can be put in a better context. On the left side of the image, the result of normalising the elevation is displayed, which is the data source which is mandatory. Lighter colours stand for high objects, while darker patches are areas of low elevation. The centre of the image is the city of Trier with its districts to test the algorithms highlighted in orange. On the right side the support data are shown. In the case of this work, a *NDVI* image is used to enhance the results. Other authors used for example an *EVI* image (Wang et al. 2005, p. 246; Huang et al. 2013, p. 46). The image has the same geographic extend as the one on right. Here the borders of the districts are also indicated with an orange line. Light colours are stand for a high *NDVI*- value, which correlate with high density of plants, like trees bushes or grassland. On the other hand, low values displayed in black or grey can be seen as either water bodies (values near $-1$) or man-made objects like streets or buildings. With this information at hand, it is possible to separate green patches against other objects and in addition streets from buildings. To enhance the results further, looking at the area of a resulting object plays a key role and is demonstrated in the section 3.3.2.

## 3.2. Preprocessing

This section shows in detail, how the data mentioned in 3.1 are computed, and how the processes are taken care of. All the examples showed in this part of the thesis are written in *Python*. The orientation of this section is based on the order of the workflow. While the first section is dealing with the first steps, the later sections will show the last steps of the processing chain as presented in figure 3.2.

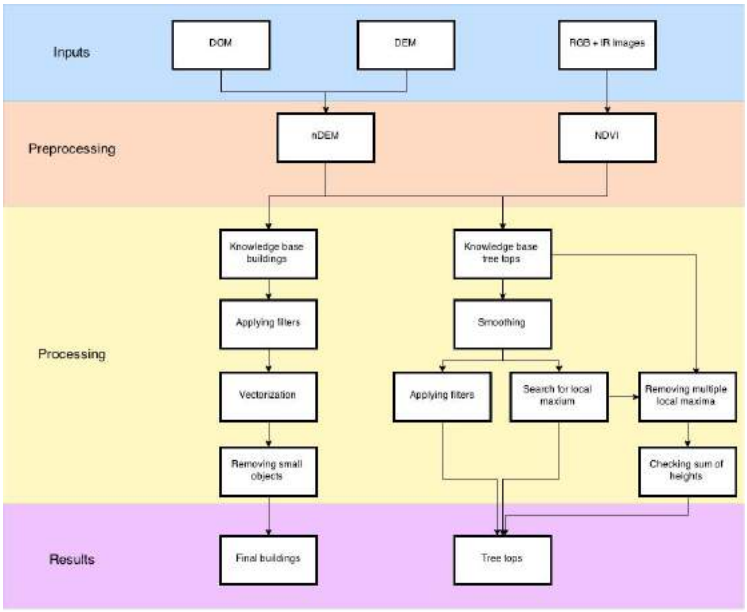



Figure 3.2.: Workflow of the program

The figure illustrates, the four stages, in which the workflow is divided. The first step describes the three different input data, which will be shown in the next sections 3.2.1 and 3.2.2. The second step is part of the same section and presents the results of the preprocessing. Section 3.3 covers in detail, how the processing is handled and how the different aspects of the theoretical part from chapter 2 are incorporated into the program. The last part is dedicated to the results of the program and will be shown (see 4) and evaluated (see 5) at the indicated chapters.

### 3.2.1. Creating of digital elevation models

The very first step of the processing chain is creating digital elevation models for the respective area of interest. This is done with the Delauny triangulation, and was conducted by *Henning Buddenbaum*. The models are separated into a surface model, representing the bare ground (digital elevation model (*DEM*)) and an object model, containing all the objects above the ground for example trees or man-made objects (digital object model (*DOM*)).

The next step is to create a model, which contains only the heights of the objects above ground. With this so called normalized digital elevation model (*nDEM*), it is possible to compensate the effects of different land types, for example a hilly terrain. To achieve this a simple formula is applied:

$$nDEM(p) = \begin{cases} p & \text{if } DOM - DEM \geq 0 \\ 0 & \text{if } DOM - DEM < 0 \end{cases}$$

The distinction between the two cases is not necessary needed because a pixel $p$ can set as no data values from the program side, if $p \leq 0$. The notation above stresses out, that for some user cases, it could be important to take values lower than 0 into account. Areas with such values could indicate regions of error, where future processing steps need to be conducted. A second step in the workflow is to adjust the cell size of the resulting image. In this case 0.5 meter is chosen, because it yields good shapes for buildings and can handle outlier in the *.xyz* data. The reason is, that the amount of points in a raster cell is large enough, to compensate the outliers. A raster with the cell size of 1 meter was created as well to compare the results and evaluate, which cell size fits best. This is discussed in the final chapter 5.
With the *nDEM*, which only addresses the height of an object, rules to distinguish different objects by its height can be applied. How this is achieved is demonstrated in the subsections 3.3.2 and 3.3.3. It will show, that the program has the ability to create a *nDEM* by itself, but for this a *DOM* and *DEM* has to be provided first.

### 3.2.2. Creating the NDVI images

The next step is carry out the computation of the *NDVI*. On this step, it is crucial to set the cell size to the same value as the one of the elevation models, else all further steps regarding knowledge based altering of the data will create problems, because the algorithm compares the values of the cells and decide, if the pixel will stay in the foreground or if it will become background. Thus the cell size is set to 0.5 meter (see 3.2.1). The computation is carried out according to Wang et al. with the formula:

$$NDVI = \frac{Band_{Red} - Band_{IR}}{Band_{RED} + Band_{IR}} \tag{3.1}$$

The *NDVI* can determine the density of green on a patch of land, by distinguishing the wavelengths of visible and near-infrared sunlight reflected by plants. To capture the needed wavelengths, different satellite system can be used. Wang et al. used the Advanced Very High Resolution Radiometer(AVHRR) from *NOAA*, to create

the *NDVI*. It has five detectors, two which are sensitive to the wavelengths of light ranging from 0.55 - 0.70 and 0.73 - 1.0 micrometres (Wang et al. 2005). As stated in section 3.1, the images used in this work were suitable for using the formula 3.1. As stated in section 3.1, the data was captured with the Eagle IC-2 sensor.

## 3.3. The Program - An Overview

This section provides a detailed explanation of how the program, called *CityEX*, which is developed for this work, is structured and how the different steps are solved regarding the programming part. The name consists of the noun *city*, which indicates, that the main targets for the program are urban areas, and *EX*, which is an abbreviation for extraction. As earlier explained for figure 3.2, the program does not deal with the creation of the data needed, but only on the steps to obtain the final results. Those steps can be manipulated according to the needs of the user. The program itself is divided into four tabs, each containing options to a specific theme. The first tap called main window covers the inputs and the preprocessing part of figure 3.2. The second and third tab are dealing with the processing steps. The last tab gives the user a short help for all the functions in *CityEX* and guides the user through the options.
The program is structured in such a way, that all functions discussed in the sections 3.3.2 and 3.3.3 are imported from other modules to exchange parts of the program easily in case a function is updated. The following table 3.1 provides information about the modules, which are used to run the program. While *os* and *sys* are utilized to access basic functions in *Python*, for example to alter a path, the *json* library has the ability to alter files in the *.json* format. In this work, the files are needed to create the help section and allows fast customization. The overall framework is created with the *PyQt4*, which provides the functions to create the graphical user interface (GUI). The library has also many options to manufacture custom buttons.

| **Module name** | **Functionality** |
|---|---|
| os | Using operating system dependent functionality |
| sys | Access to some variables used or maintained by the interpreter |
| json | Creating and manipulating *.jsom* files |
| PyQt4 | Library to create user interfaces |
| gdal | Manipulating spatial raster data |
| ogr | Manipulating spatial vector data |
| osr | Creating and transforming coordinates |
| NumPy | An array-processing package |
| SciPy | For mathematics, science and engineering. Depends on numpy |

Table 3.1.: Imported modules from external *Python* libraries

The library called Geospatial Data Abstraction Library, or short *gdal*, is an open source library for geospatial data formats, according to its website. The module *ogr* is developed in the same framework as *gdal*. Both provide primary data access engine for many applications and can handle over 50 raster formats (*gdal*) and more than 20 vector formats (*ogr*). In addition, both are multi-platform-libraries, which can be accessed from *Python*, *Java*, *C#*, *Ruby*, *VB6* and *Perl* (Foundation 2015, online). The modules provided by *osr* are highly connected to the other two spatial libraries and allow the transformation of coordinates, changing the spatial system and the creation of *.prj* files, which contains information about the coordinate system in a well-known text format.
The next two libraries are mainly scientific ones, which support multiple mathematical operations. The first is called *numpy* and deals with different types of arrays. How this module is used in the program is shown in 3.3.2. *Numpy* is a keystone for *CityEX* and used in nearly all modules created for it. *SciPy* on the other side contains toolboxes to tackle common problems in scientific programming. Therefore it has many functions for interpolation, image processing, statistics and special applications (Haenel et al. 2013). In this work the functions of the *ndimage* sub-module are used.
While external libraries are important for solving tasked often repeated, spacial problems need a set of algorithms for creating outputs as demonstrated in this work. Therefore table 3.2 gives an overview of the created modules and their purpose.

| **Module name** | **Functionality** |
|---|---|
| module_ndem | Creation of *nDEM*, checks plausibility |
| module_ndvi | Creation of *NDVI*, checks plausibility |
| module_knwlgd | Creates a rule based raster for buildings |
| module_math | Provides basic mathematical functions |
| module_filter | Enhances the raster from *module_knwlgd* |
| module_classification | Forms a binary raster from *module_filter* |
| module_raster2poly | Creates & filters polygons from *module_classification* |
| module_green_knwlgd | Creates a rule based raster for trees |
| module_local_maxima | Creates tree tops from *module_green_knwlgd* data |

Table 3.2.: Modules created for *CityEX*

The first two modules are needed to create the proper input data as seen in figure 3.2. The function will be explained in section 3.3.1. The next five modules are used to create the building and are described in section 3.3.2. The last two are needed to construct the tree tops and their positions.
The first subsection will explain the overall settings, which can - or have to - be made to create one of the desired outputs. The next part in section 3.3.2 will show the way the data are incorporated into the workflow and what options can be set to

alter the data. The last part of this chapter will be the workaround to create tree positions.

### 3.3.1. The Main window

This part of the thesis will explain in detail, what options can be set in the first tab of the program. As stated above in section 3.3, the first tab an user gets to see, is the one presented in figure 3.3. To structure the work in a more straightforward sense, the window is parted into the three parts a), b) and c), which will be explained.

The first part, indicated with a), takes the inputs from the digital elevation models. The data can be either a *nDEM* or a *DEM* and a *DOM*. The condition, under which the two later ones can be used is, that both have the same coordinate system, the same cell size and the same extend. This is checked in the module *module_ndem*. The availability of the two input buttons labelled *Choose* is triggered by the function *handel_inputs*, which is demonstrated in listing 3.1. It shows, how the different buttons are accessed and how their values are changed. The function *validate_inputs* is a helper function to check, whether certain conditions are met and if enough information are present, to start calculating the outputs. Thus this function is integrated into many other algorithms as shown in the appendix. The variable *value* is altered by the drop down menu for the inputs and changes the status of buttons to guide the user through the program. In addition, the availability of the *Ok* button is triggered by this function, thus an user can not start the processing chain by accident. The downsides of such an approach are, that in case a values was wrongly chosen, or has a problem with the data format, then an user first has to look by himself for the source of the error. To tackle this issue, an information text field is provided at the bottom, to let the user know, whether any errors occurred or if everything runs correctly.

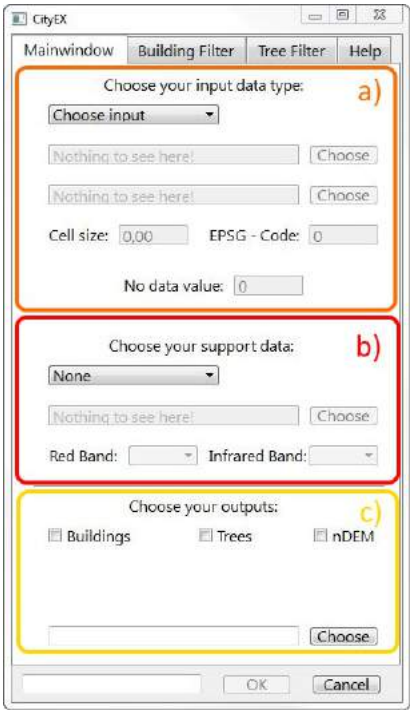



Figure 3.3.: The Start window

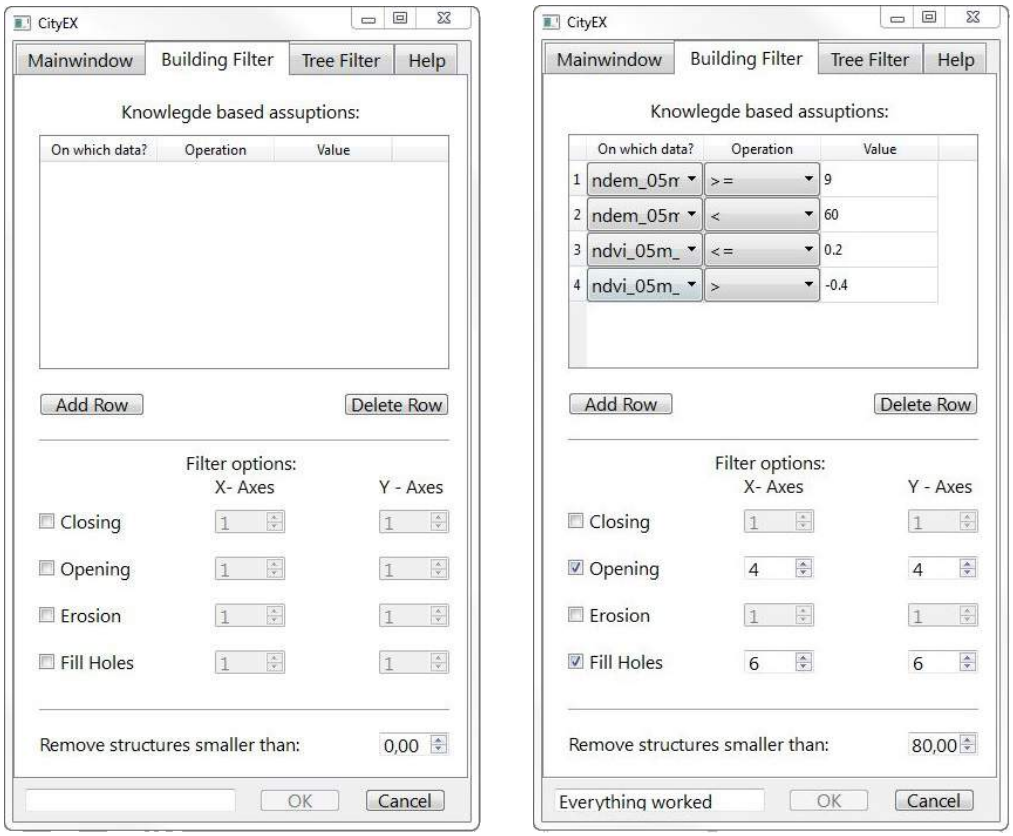
```
self.validat_inputs()
  if value == 0:
      self.lineEdit_tab1_input_dem.setPlaceholderText(self.now_you_see_me)
      self.lineEdit_tab1_input_dom.setPlaceholderText(self.now_you_see_me)
      self.checkBox_tab1_ndem.setChecked(False)
      self.checkBox_tab1_ndem.setEnabled(False)
      self.lineEdit_tab1_input_dem.setEnabled(False)
      self.lineEdit_tab1_input_dom.setEnabled(False)
      self.pushButton_tab1_input_dem.setEnabled(False)
      self.pushButton_tab1_input_dom.setEnabled(False)
      self.spinBox_tab1_EPSG.setEnabled(False)
      self.doubleSpinBox_tab1_cellsize.setEnabled(False)
      self.spinBox_tab1_nodata.setEnabled(False)
      self.input_value = 0
```

Listing 3.1: Excerpt from the function *validate_inputs*

This mechanism ensures, that the user just has those options, which are meaningful for the specific input data. Thus in the further process of creating the next files, unneeded options are automatically set on a default value or are not available at all. A user could give the program any kind of data to extract objects, but as many studies prove, using a height model of any form is most of the time the starting point of extracting various types of objects, regardless whether it is a rules based or an object orientated approach (Tiwari and Pande 2011).
The next step is presented in part b) of figure 3.3. Here are the options, which the user can set, to influence the support image. Either there is no such image, then the options are not considered, or the user uses ancillary data, in such a case, the program gives the user the choice, using a *RGB-image* with an additional infra red channel, or using another kind of data, like an already computed *NDVI*, *EVI* or any other kind of data, which allows the user to distinguish between buildings and natural objects like trees or fields. Thus the program provides flexibility in the choice of data. This is also true for the height data. To make use of a *RGB-image*, the program uses the module *module_ndvi* to compute the image. In future, the choice of creating other ancillary data could be implemented. Further explanations are made in section 5.1.
Part c) provides the user with the choice of the types of object, he want to extract and additionally, if a *nDEM* should be created. By default, this option is set to *True* if the input data provided are a *DEM* and a *DOM*. In all other cases, this option is greyed out, because the user already provided a *nDEM*. The last step is to set an output-folder, where all files created in the processing are stored. It could be an advantage to create the different files into sub-folders, to separate them by the type of object they create. This could gain importance if multiple files have to be created.

### 3.3.2. The Building Options

This subsection deals with the next tab in the program. The tab is called *Building Filters*. Here, the user can decide which rules has to be set. The figure 3.4 compares the program before and after data was put in. Thus the altering of the behaviour is shown in figure 3.4b. The explanation is divided into three steps. First the different parts of the program will be shown from the user interface, then the *Python* functions will be displayed and finally the strings are tied together to illuminate the assembly of those functions.




(a) (b)

Figure 3.4.: A comparison the building window before any data is put in (a) and when the data and filters are set (b)

The figure shows, that the window is divided into four main parts. The first deals with the knowledge rules. Here the user can set his assumptions for the main data (in this study the height data of a *nDEM*, see 3.2.1) and the ancillary data (here a *NDVI* image is used, see 3.2.2). The two buttons below the table are self-explaining. The *Add* button allows the user to add a new line at the end of the table, while the *Delete* button deletes a selected row or, if no row is selected, the last row will be

deleted. Thus the two functions for the buttons are short and straight forward, as the listing 3.2 shows.

```
1  def row_add(self):
2      self.cur_row = self.tableWidget_tab2_knowlegde.rowCount()
3      self.tableWidget_tab2_knowlegde.setRowCount(self.cur_row + 1)
4      self.comboboxes_for_input()
5
6  def row_delete(self):
7      try:
8        if self.rowRemove < self.tableWidget_tab2_knowlegde.rowCount() and not self.rowRemove ==
             -999:
9          self.tableWidget_tab2_knowlegde.removeRow(self.rowRemove)
10       else:
11         self.cur_row = self.tableWidget_tab2_knowlegde.rowCount()
12         self.tableWidget_tab2_knowlegde.setRowCount(self.cur_row - 1)
13     except:
14       self.cur_row = self.tableWidget_tab2_knowlegde.rowCount()
15       self.tableWidget_tab2_knowlegde.setRowCount(self.cur_row - 1)
16       self.rowRemove = -999
17
18
19 def returnRowForRemove(self, rowRemove, colRemove):
20     self.rowRemove = rowRemove
```

Listing 3.2: The Add and Delete functions

The function *row_add* makes use of the in section 3.3 mentioned *PyQt4* library. It accesses the variable *tableWidget_tab2_knowlegde* to get the count of rows, and then simply adds one row to the number. The function, which is called in line 4 ensures, that the drop down menus for the first and second column are provided. The *row_delete* on the other hand is more complex. The *if* - *else* statement is embedded into a *try* - *except* environment, which is created to catch errors. The *if* statement deletes a selected from the table, while the *else* part deletes the last row of the table. The function *returnRowForRemove* is thus a helper function and provides the needed variable and is called every time the user selects a row in the table.
The next part of the window displays the four different filters, where the user can activate the filters independently from the others. The option, which is displayed to the right of the check box allows to set the x and y axes of the filter and thus alters the kernel to adapt the image which needs to be filtered (see section 2.2 for theoretical background). The listing 3.3, which is presented underneath, defines the two functions, which are needed. While the *handel_filter_settings* function is responsible for updating the final outputs, which is saved in the variable *self.filter_settings*, the function *handel_filter_checked* fills the list and provides them for the later use in the final function. The number 42 is used in this context because the function which is called needs an integer value, and thus this number is set and does not have any other use, because the variable $v$ is not used. The list *self.ndimage_func_lst* and

the dictionary called *self.check_b_filter_dic* are providing the access to the widgets of the user interface. The first list has the functions from the *SciPy* library in it and will pass them onto the final function which will be discussed in the section 3.3.4.

```
1  self.ndimage_func_lst = [ndimage.binary_closing, ndimage.binary_opening,
2            ndimage.binary_erosion, ndimage.binary_fill_holes]
3  self.check_b_filter_dic = {
4      self.checkBox_tab2_closing : [self.spinBox_tab2_x_closing, self.spinBox_tab2_y_closing],\
5      self.checkBox_tab2_erosion : [self.spinBox_tab2_x_erosion, self.spinBox_tab2_y_erosion],\
6          self.checkBox_tab2_opening : [self.spinBox_tab2_x_opening, self.spinBox_tab2_y_opening
               ],\
7          self.checkBox_tab2_fillhole : [self.spinBox_tab2_x_filling, self.
               spinBox_tab2_y_filling] }
8
9  def handel_filter_settings(self, v):
10   self.handel_filter_checked(42)
11
12 def handel_filter_checked(self, v):
13     self.validat_inputs()
14     self.filter_settings = []
15     for wi in range(len(self.check_box_lst_filter)):
16       value_spin = self.check_b_filter_dic.get(self.check_box_lst_filter[wi])
17       if self.check_box_lst_filter[wi].isChecked():
18         value_spin[0].setEnabled(True)
19         value_spin[1].setEnabled(True)
20         self.filter_settings.append([self.ndimage_func_lst[wi], (value_spin[0].value(),
               value_spin[1].value())])
21       else:
22         value_spin[0].setEnabled(False)
23         value_spin[1].setEnabled(False)
```

Listing 3.3: Function to handle the filter settings

The function *handel_filter_checked* has two tasks. First it creates and fills the output list *self.filter_settings* for later use. For this purposes, the function iterates over the list mentioned on line 3 and creates a list inside the list with the arguments of the filter used, drawn from the variable *self.ndimage_func_lst* and the x and y values as integers. The second use is to switch the boxes for altering the x and y axes on and off (see line 19 and 20 as well as 24 and 25 in listing 3.3) to prevent the user of using wrong values. Because it is updated with the check boxes as well as with the inputs of the axes, the resulting list always mirrors the current settings the user has made.

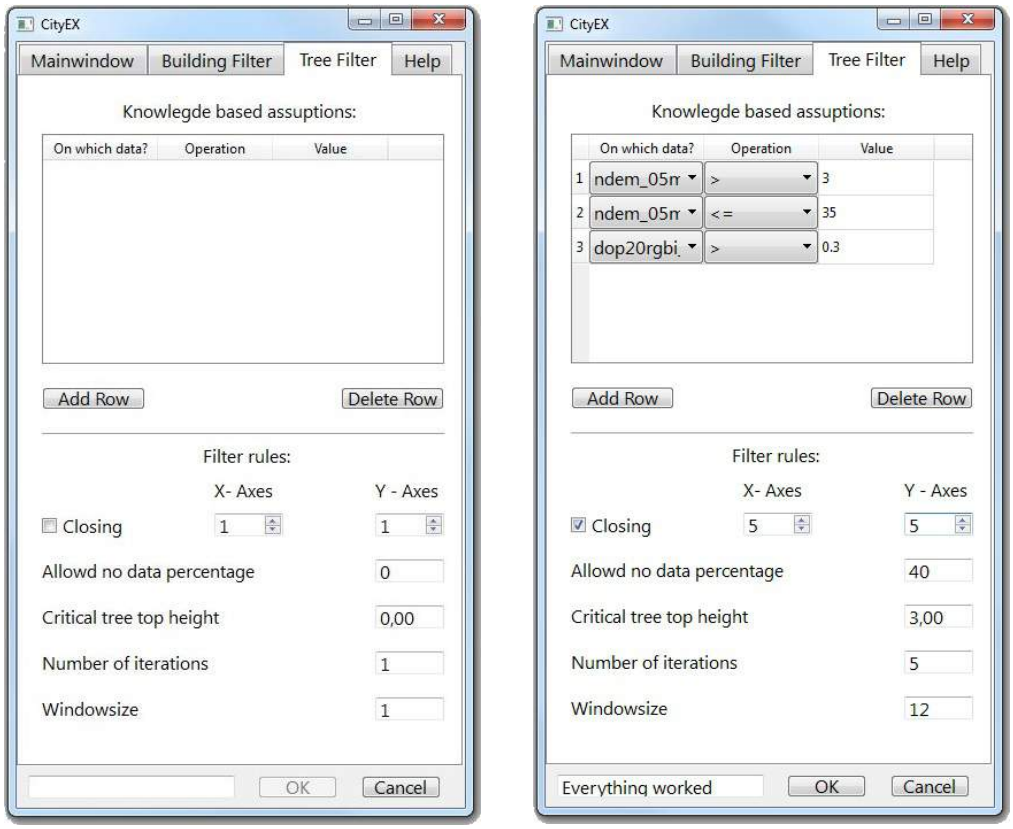
The next part of the window is the menu, where the user can set a value, which will remove structures with an area smaller than this said value. This value can be given or set to 0 if objects like trees or errors in the data need to be removed. It also ensures, that objects, which can not be buildings due to their small area, are removed. Those objects are mostly artefacts created by the knowledge engineering or the filtering.

The last part is again the buttons *Ok* and *Cancel*. They are displayed in every tab

of the program in order to allow the user to more straight forward handling of the program and will be discussed in length in the section 3.3.4.

### 3.3.3. The Tree Options

This section deals with the functionality of the tree filters tab. Here the user can make assumptions which has the same effect on the data, as the ones made in section 3.3.2. The structure remains the same. First, the user interface elements will be described, then the functions belonging to those elements are presented and explained. The figure 3.5 shows, how the program alters when data are provided. The functionality is the very same as the one presented in section 3.3.2 with just some names of variables altered, in order to prevent names colliding with each other. This holds truth for the part of the knowledge based assumptions and the function used here is similar to the one shown in the listing 3.2 and thus will not be explained here.




(a) (b)

Figure 3.5.: A comparison of the tree filter window before any data is put in (a) and when the data and filters are set (b)

The next part is the option, if the user wants to apply a filter-algorithm to the data or not. This filter is a closing filter and works as demonstrated with the figure 2.2b

in section 2.2. But before the filter is applied, the user has the choice of denoising the image. This is set by the box labelled with *Number of iterations*. This iteration works as in listing 3.4 highlighted. The image is converted into a mask file, which only has binary values (see section 2.2 and 3.3.4 for details), and then pixels, which are not surrounded by a defined set of non-background pixels are set to background, or if a pixel has a certain amount of foreground neighbours, then it is set to foreground.
For this task the first*for* loop sets the number of iteration to the user set value or - if nothing was changed - to one. Than the next two *for* loops are accessing first the row, next the column, to run over the image and analyse it. Than the two *if* statements ensures a faster processing, by only looping over those values, which are 0.

```
for kk in range(self.num_iter):
  for i in range(len(mask)):
    for k in range(len(mask[i])):
      if mask[i][k] == 0:
        if (i < len(mask)-1 and k < len(mask[i])-1) and i != 0 and k !=0:
          value_new =  [mask[i-1][k-1], mask[i-1][k], mask[i-1][k+1], mask[i][k-1], mask[i][k
              +1],mask[i+1][k-1], mask[i+1][k], mask[i+1][k+1]]
          counter_negativ = value_new.count(0)
          counter =       value_new.count(1)
        elif i != 0 and i < len(mask)-1 and k == len(mask[i]) -1:
          value_new = [mask[i-1][k-1], mask[i-1][k], mask[i][k-1], mask[i+1][k-1], mask[i+1][k
              ]]
          counter_negativ = value_new.count(0)
          counter =       value_new.count(1)
        elif  i != 0 and i < len(mask)-1 and k == 0:
          value_new = [mask[i-1][k], mask[i-1][k+1], mask[i][k+1], mask[i+1][k], mask[i+1][k
              +1]]
          counter_negativ = value_new.count(0)
          counter =       value_new.count(1)
        elif i == 0 and k != 0 and k < len(mask[i]) -1 :
          value_new = [ mask[i][k-1], mask[i][k+1],mask[i+1][k-1], mask[i+1][k], mask[i+1][k
              +1]]
          counter_negativ = value_new.count(0)
          counter =       value_new.count(1)
        elif i == len(mask)-1 and k != 0 and k < len(mask[i])- 1:
          value_new = [ mask[i][k-1], mask[i][k+1],mask[i-1][k-1], mask[i-1][k], mask[i-1][k
              +1]]
          counter_negativ = value_new.count(0)
          counter =       value_new.count(1)
        else:
          value_new = [1]
          counter_negativ = value_new.count(0)
          counter =       value_new.count(1)
        if counter_negativ <= 3:
          mask[i][k] = 1
        elif counter <= 2:
          mask[i][k] = 0
        else:
          pass
```

Listing 3.4: Function to handle the smoothing

The *if* - *elif* - *else* statements nested inside the *for* loops ensure, that all cases can be handled, to prevent error regions around the edges of the image. The variable *value_new* creates a list of binary values. The next step is to evaluate, if the neighbourhood, which is stored in the variable, contains not too many zeros (in this case not more then 3), or too few pixels with the value one (as seen in line 31 less than 2). With this rules, the value of the pixel is altered accordingly.
The next parameter to set is the percentage of the allowed no data values. This allows to reduce the amount of tree top points set to an edge of a tree, the figure 3.6 picture the difference well.

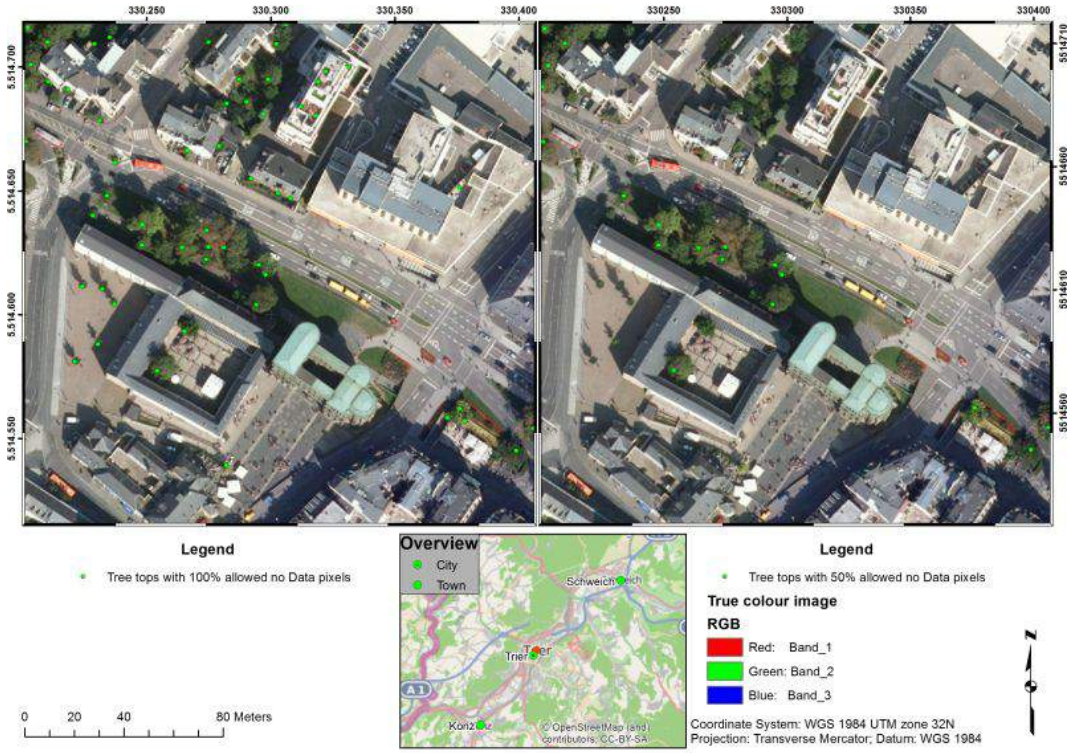



Figure 3.6.: Comparing the impact of changing the allowed no data values from 100% (left) to 50% (right)

The images shows a part of the Altstadt district - to be exact the area around the Porta Nigra - to highlight the impact of choosing an appropriated no data amount. Both images only differ in the choosing of the no data rule. The other parameters are the same. It is demonstrated, that the image on the right has a much smaller amount of detected tree positions, as the one on the left. For example the trees on the left of the image are detected while on the right image they are not detected at all. While this is an improvement, the tree tops in the centre of the image seems to be an overestimation. A fatal error of recognition the tree tops is shown on the top of the image. This is the result of a lax handling of the no data rule. While on the right map, there are detected tree tops on the building, which are false detections, the rules

applied on the left map have prevented those false detections. It is demonstrated, that this rule is crucial in areas, where the trees are not predominate but just are a part of the whole urban area. With this concept of choosing tree tops, which have a certain amount of surrounding foreground pixels in their neighbourhood, it is ensured, that the amount of trees in an area is not overestimated and that the filtering of the image can be improved further. Problems, which results in discarding actual tree tops are discussed in the section 5.3.
Another option, to improve the quality of the final results, is the size of the kernel. The details of this approach are discussed in section 2.3. The program let a window slide over the image and with the help of the libraries *numpy* and *gdal*, the results are evaluated. That this option is critical above all the other options is demonstrated in the figure 3.7.

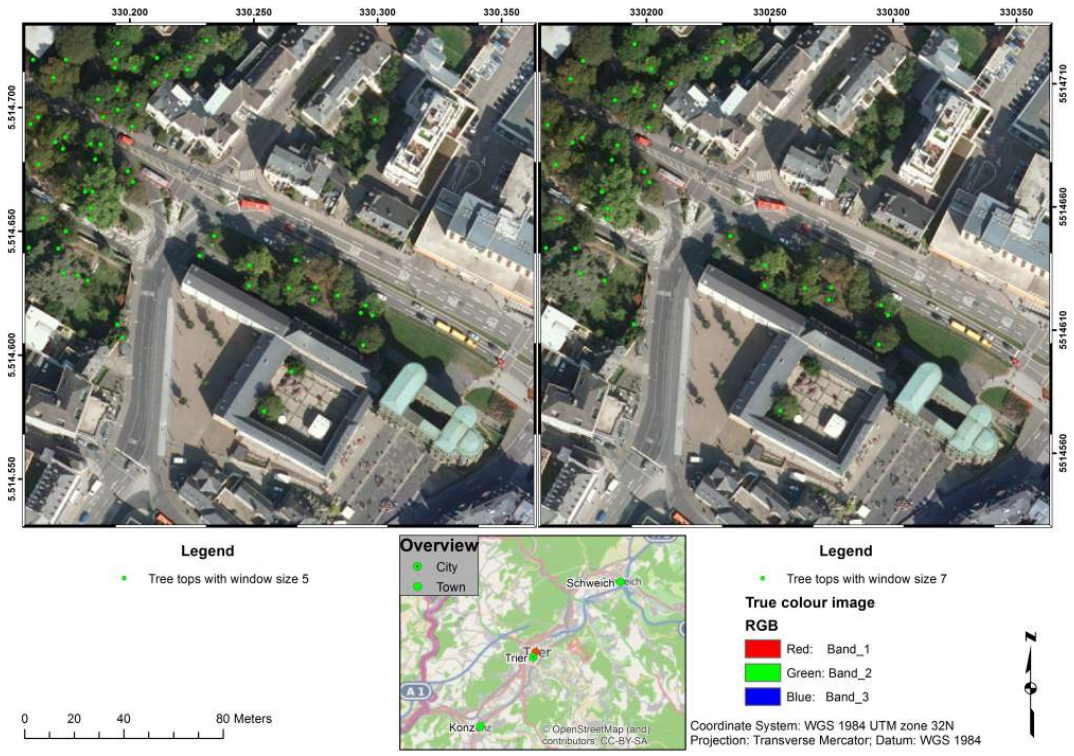



Figure 3.7.: Comparing the impact of changing the kernel size from 5 (left) pixels to 7 (right) pixels

The figure shows a part of the district Altstadt. The green dots represent detected tree tops from the algorithm. By comparing the upper left part of the two images, it is clear demonstrated, that the kernel size of five pixels produces significant more tree tops, than the seven pixel kernel. While for inner courtyard this seems to improve the detection rate, the tree tops in the centre of the image are clearly an overestimation. The same holds true for the upper left part of the image. Thus the choice of an appropriate kernel size has a higher importance, than the usage of a different no data

proportion. The right kernel size also depends on the cell size of an image. Again, the knowledge of trees in general and their crone diameter in detail is important to derive meaningful results.
The last setting is made by the value labelled *critical tree height*. If set to a value $crit\_height > 0$, then the variable will be taken into account. If the value is not changed, the program will create an value according to the listing 3.5 and will remove all potential tree tops which does not fulfil this requirement.

```
if self.crit_height == None or self.crit_height == 0:
  green_mean = green_band[green_band != green_filterd.GetNoDataValue()].mean()
  crit_height = green_mean - np.std(green_band[green_band != green_filterd.GetNoDataValue()])
else:
  self.crit_height = float(self.crit_height)
```

Listing 3.5: Handling the critical height parameter

The code snippet demonstrates, how the value of the variable *crit_height* is chosen. The *if* statement checks, if the user has set a value. In this case, this value is used. In the case, the user didn't chose a value, it is created in such a way, that the mean of the filtered image,which is computed without considering the no data values, is the basis of the height. This can be done, because the image only contains green patches, which are associated with trees.
The next step is to allow, that values within a standard deviation below the mean are also considered. This is done by not considering the no data values of the image. The *else* part of this statement is not necessary, but is implemented to reduce the chances of errors occurring.
In conclusion, the different settings differ in the force of impact they have on the final results. This means, that the wrong value for one parameter doesn't pose a hug threat for the final result. This indicates, that the algorithm is robust and can be applied to different areas. The main driver of error seems to be the kernel size.

### 3.3.4. Handling the processing chain

The last section is dealing with the program and describes the handling of the different modules presented in 3.3 with table 3.2. To be exact, the section deals with the questions, how the program executes the orders, when finally the user has set all parameters, to create the desired outputs and can press the *Ok* button.
The first task the program has, is to evaluate, what options were set. This includes, if the user has set the kind of outputs. As the figure 3.2 suggests, The creation of the two main outputs (building and tree tops) are separated from each other. Then the program checks, whether or not the inputs are already processed, so the program does not need to run the preprocessing part of the workflow. The next step

is checking the output - settings. The reasoning is, that the output largely impacts the needed functions. Another impact on the final processing chain are the options, as described in the sections 3.3.2 and 3.3.3. From here the main program lets the different modules handle the processing. There, the secondary options are checked and finally the outputs are created. The following table shows the abstract of how the program works.

| **Module** | **Needed Variables** | | | | | | **Outputs** |
|---|---|---|---|---|---|---|---|
| ndem | DEM | DOM | EPSG | × | × | nD | nDEM |
| ndvi | RGB | Red Band | IR Band | × | × | × | NDVI |
| knwldg | nDEM | *NDVI* | knwlgd list | *band* | × | *nD* | knwldg building |
| filter | nDEM | knwldg building | × | × | × | × | filter building |
| raster2poly | filter building | × | × | × | × | × | buildings |
| green_knwlgd | nDEM | *NDVI* | knwldg list | *filter list* | *iterator* | *nD* | knwlgd trees |
| local_maxima | knwldg trees | window size | crit. nD | *crit. height* | *EPGS* | × | tree tops |

Table 3.3.: Created *Python* modules and their inputs and outputs. nD = no Data; crit. = critical; knwldg = knowledge; italic names = optional values

The different modules are displayed on the left column, while the variables, which are needed, are presented in the middle section. Every module needs the path of the output folder as well as the name of the file it will create. The names written in italic are optional values and thus can be set by the user or standard values will be provided by the program. The names, which are needed to create the files are provided by main program. This ensures, that the names are created correctly and are provided from one hand and that - if the modules are adjusted - the naming stays the same. Another task managed by the modules are the validation of the presented data. If the user did not set any ancillary data, then the modules has to adapt to this change. This holds true for the modules *module_knwldg* and *module_green_knwldg*.
At this stage,the program runs the function *ok_event*. This function consists of roundabout 200 lines of code and thus the function will not be displayed here but can be seen on the appendix. The function consists of two *if* statements, where another *if* statements is nested inside. The first checks, whether the building check-box is checked or not. The second statement does the same for the tree top check-box. The *nDEM* and*NDVI*options are checked in the nested *if* statements along with two variables, which prevent the program from trying to create the files twice. The nested statements are therefore structured as a tree and ensures, that the right variables are passed to the modules.

# 4. Results

In this chapter, the results will be presented and interpreted according to the section 1.3. The results are separated into the six districts and therefore the interpretation will focus on the regarded district.
When regarding the results, the rules which are applied only differ slightly and were only adjusted, if the first results are not precise enough. This is the case in the district of Pallien for example, where the knowledge base assumptions and the building recognition had to be adapted to the terrain. To create the final results, a set of options are carried out. The only parameter changed for the buildings are the knowledge assumptions, because here the chances of misinterpreting the buildings are the highest. So three different sets of first options were created, as table 4.1 shows. All the other rules are test rules for the outputs of the first module and thus only depends on those assumptions. If for every module three options has to be tested, and this for each district, then this would yield a total of 162 possible buildings, or 27 options for each district. With this number in mind, it is reasonable to limit the options, which was done, as the following table shows.

| **Option** | **District** | **Value(s)** |
|---|---|---|
| knowledge assumptions 1 | five | $5 < h < 60\ and\ -0.3 < a \leq 0.2$ |
| knowledge assumptions 2 | five | $4 < h < 60\ and\ -0.4 < a \leq 0.3$ |
| knowledge assumptions 3 | five | $3 < h < 60\ and\ -0.5 < a \leq 0.1$ |
| knowledge assumptions 4 | Pallien | $2 < h < 60\ and\ -0.5 < a \leq 0.1$ |
| filter options | all | $fill(4,4); closing(3,3); opening(5,6)$ |
| rules to extract buildings | five | $area \geq 80$ |
| rules to extract buildings | Pallien | $area \geq 60$ |

Table 4.1.: Options for different modules and districts to create the building shapefiles with $h$ = height data and $a$ = ancillary data

The table shows the applied rules for creating the buildings. As explained earlier, the values for Pallien had to be adjusted, to get a similar result as for the other districts. The four options for the knowledge engineering differ in the acceptance of values for the pixels and therefore the amount of truly, non and false detection differ, as the section 5.2 will point out.
When regarding the buildings, one can say, that the main driver for errors is the first step of knowledge options. All other steps are carried out to improve the results.

The extraction of tree tops depends on a knowledge engineering result and on the settings of the local maxima search. Here the options of the local maxima has an impact. Table 4.2 illustrate the different settings of the rules.

| Option | District | Value(s) |
|---|---|---|
| knowledge assumptions 1 | all | $5 < h < 60 \; and \; a \geq 0.3$ |
| knowledge assumptions 2 | all | $4 < h < 60 \; and \; a \geq 0.4$ |
| filter options | all | $closing(3,3)$ |
| rules of extraction | all | $ws = 7 \; and \; 50\% \; nD \; and \; min(h) = auto$ |

Table 4.2.: Options for different modules to create the tree top shapefiles, with $h$ = height data, $a$ = ancillary data, $ws$ = window size and $nD$ = no Data

For the tree tops, only 2 sets of knowledge assumptions yielded sufficient results. The option "knowledge assumptions 3"where too strict and therefore not considered. As in the section 3.3.3 demonstrated, $ws$ was set to 7, after several empirical tests showed, that this is the best practise solution, because with a smaller $ws$, the overestimation was too high and larger window sizes in combination with the 50 % $nD$ missed out too many small trees. The 50 % of allowed no data values ensures, that also smaller trees can be detected and thus can counter rules set too strict in the knowledge base. Automatic search for a solid estimated height parameter is set to true, which is calculated according to the code example from listing 3.5. The ancillary data is for this side a $NDVI$ image.
The following results will be presented alphabetically. The accuracy is discussed in the sections 5.2 and 5.3 and will show the positive qualities as well as the drawbacks of the proposed method.

## 4.1. Results for the district Altstadt

The first district, which will be presented is Altstadt. It has 9373 inhabitants and an area of 2.022 *sqm* (Trier 2015a). It is the core of the historical centre and its extent overlaps with the ancient wall built by the Roman Empire. It holds the most shops of retailers and is therefore an important centre of commerce. The figure 4.1 shows the overview of the district and indicates its position in Trier, while figure 4.2 zooms into four places, to explain the details of the resulting data.

The figure 4.1 shows, that the district of Altstadt is an area, which is densely build up. To the west, the river Moselle can be found, with an island, which has many trees on it as the results suggest. The borders of the district, which have many trees, are the former city wall. This are is converted into an avenue. The oblong building on the eastern edge of map is the main station and their roofs and accommodation.

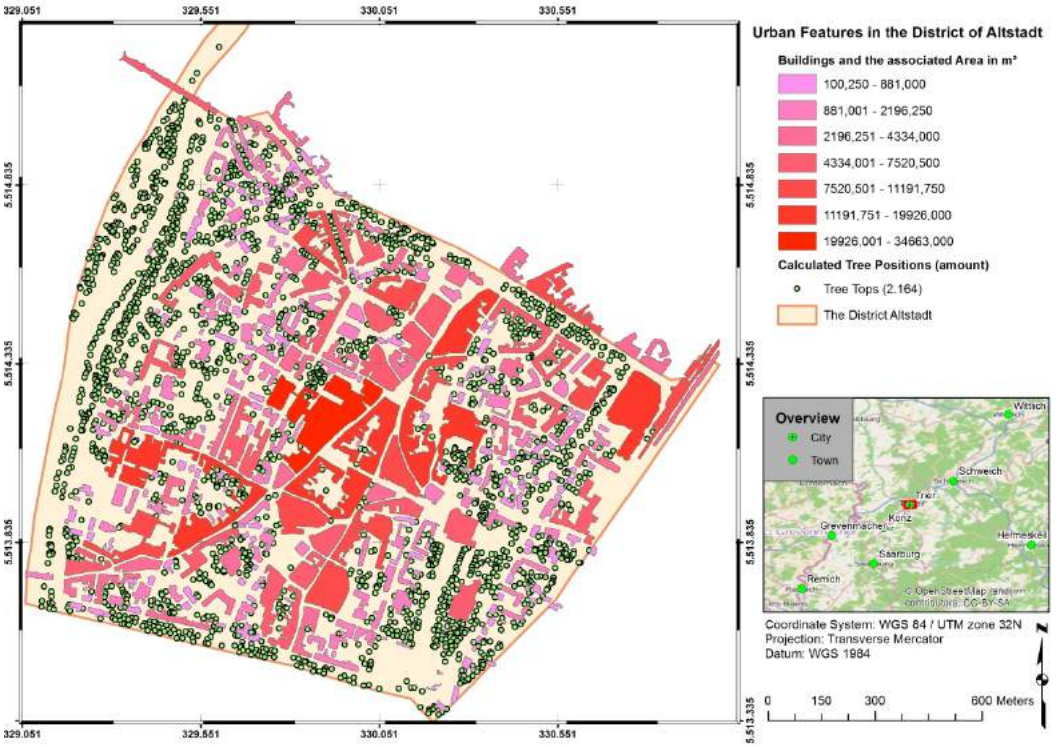



Figure 4.1.: The results of the building and tree top extraction for the district Altstadt

The south east part of the image are the so called ˝Kaiserthermen ˝and the area of the ˝Palastgarten˝. The separation between buildings only worked, when they are separated by a street or a square. This may lead to inconveniences when assigning uses to the created building models. This holds especially true for buildings near the centre of the map, where the main market is situated. In conclusion, the shape, the streets and squares of the district can be recognised easily and picture the true buildings accurately as section 5.2 will show.
As for the tree position, the accuracy is a question of the correct amount rather than the exact position. To determine this, the overview isn't the right choice to make accurate assumptions. Therefore the figure 4.2 gives a detailed look on four chosen points of interest. They show not only the tree tops, but also the buildings, so the earlier made statements can be confirmed. The true-colour images in the background improves the ability to interpret the results. The small overview map indicates the positions of the four subsets.

The upper left subset shows the area around the Porta Nigra (on the East of the subset). The buildings have a high accuracy as the one in the centre of the image indicates. Small features are preserved as well as the general shape of the structures. A problem can be seen, when looking for the tree positions. Here the amount may be overestimated. This seems to be a problem in the upper left map. On the other

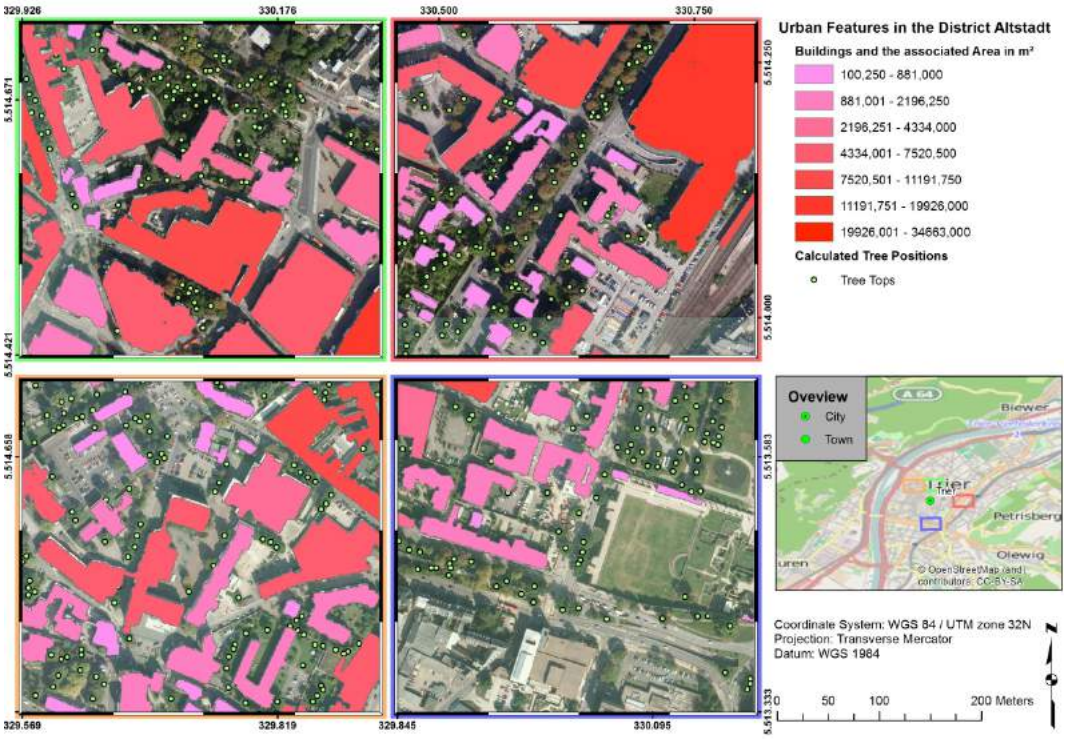



Figure 4.2.: A detailed look on the results of the building and tree top extraction for the district Altstadt

three maps, the amount and position seems to fit the tree locations of the underlying orthophoto. This can be seen by checking the alignment of the trees on the lower left image, showing a part of the Deutschherrenstraße. This behaviour points out that the trees are planet to enhance the city environment on the one hand, on the other hand the algorithm detected them correctly. On the upper right subset, situated south of the main station, some trees are missed out. As the true colour image indicates, those trees are already losing their leaves and therefore the *NDVI*, which was used to support the *nDEM*, had low values resulting in ignoring those trees, because all values lower than 0.4 were ignored (see table 4.2 for details). The area around the "Kaiserthermen"demonstrates, that the method can handle evenly distributed trees. Again the alignment of the trees is obvious and the detection only misses out some smaller trees (for example on the South of the subset).
The next maps, which will show the same subsets of Trier, as the ones presented in chapter 4, will outline problems, which are sources for the errors and thus can explain a portion of the false detections.

In figure 4.3 the Altstadt district is presented. can be shown, that only small buildings are missed out and therefore producing false negative values. Only on the upper right subset, a larger building isn't detected. This incidence can be explained by the fact, that objects, smaller than a certain threshold are removed (see table 4.1 for a

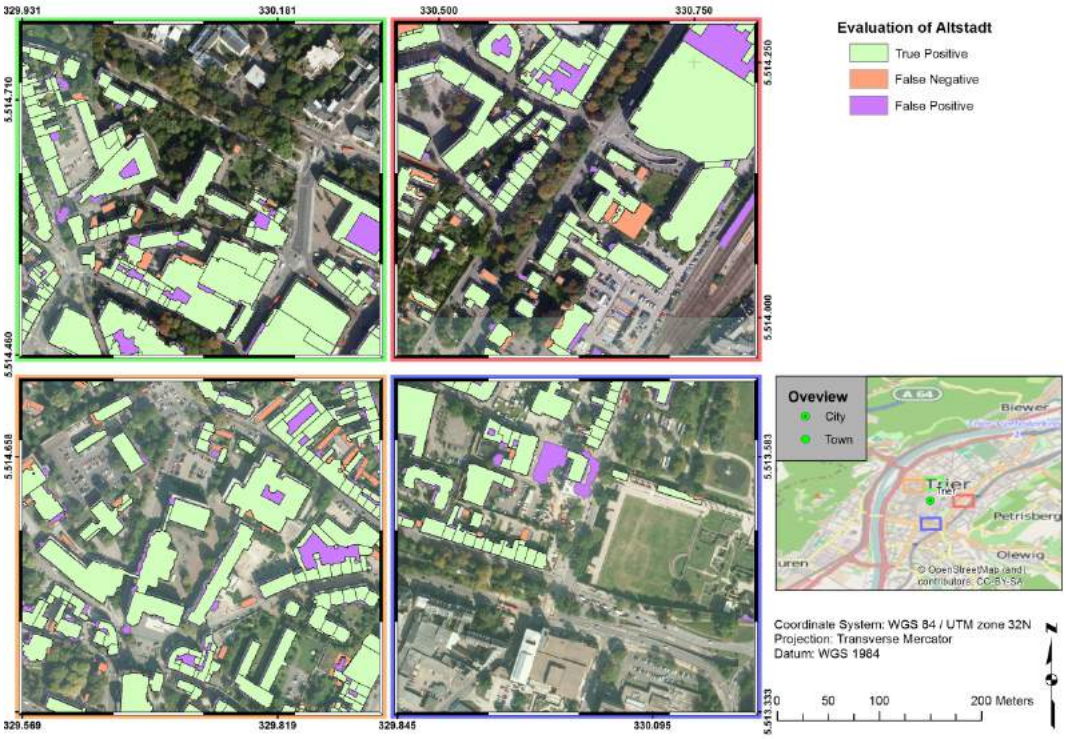



Figure 4.3.: Evaluation of the district Altstadt with four detailed maps; Green = true positive, orange = false negative, purple = false positive

detailed look). This includes portions of trees, but also smaller buildings like carports and outbuildings.
To underline the necessary of evaluating the tree positions in the urban environment, the data, presented in figure 4.4, uses the similarity search from *ArcMap*, to rank the points according to their similarity. The output contains a field called *SIMINDEX*, which represents the sum of squared values and quantifies, how similar each solution is to the target feature.

The figure shows, that nearly all trees are hit, with the exception of three outliers, which are not present in the ground truth data. Regarding the precision, the row of trees in the southern part of the map illustrate, that the method performs well, when the trees are spaced or regularly planted.
In conclusion, the yielded results show promising outputs regarding their accuracy and the detection rate, demonstrating that the method is well suited for urban areas. Small parts seem to be overestimated and thus adjusting the rules could solve those problems. The evaluation underline those findings.

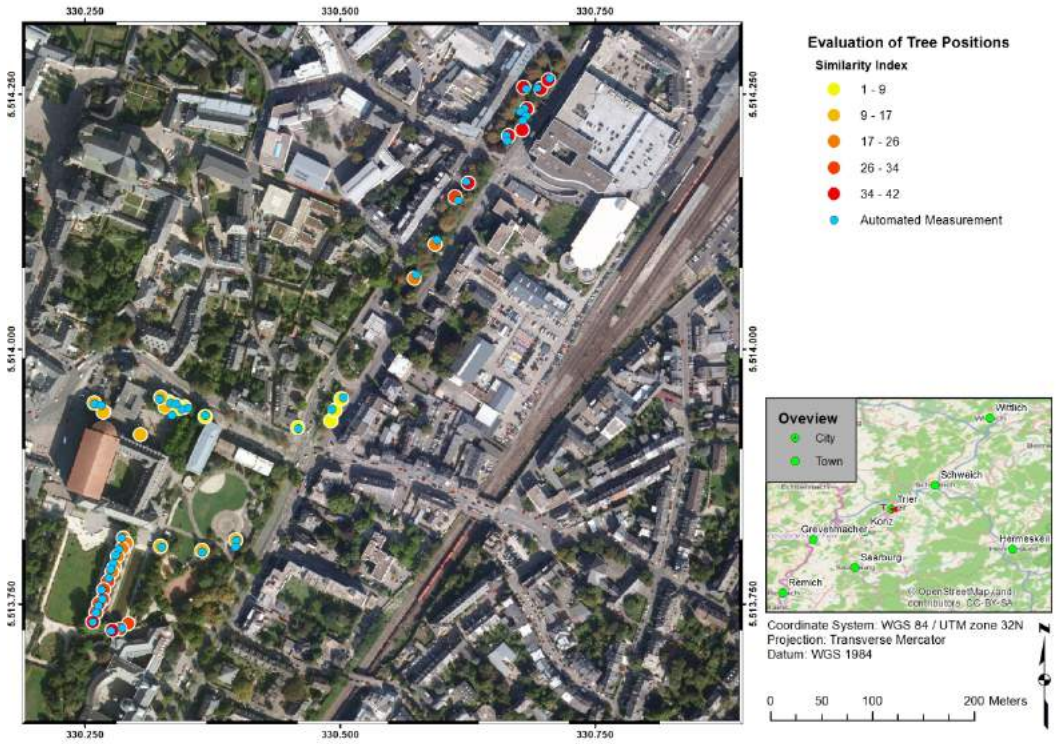



Figure 4.4.: Evaluation of 42 tree positions with ground truth data

## 4.2. Results for the district Feyen

The next section is dedicated to the district of Feyen. It is situated at the southeast edge of Trier and has a forest on its borders, as the map from figure 4.5 shows. The retail segment is non existent and only some minor industrial housings are situated here. It can be assumed, that the main part of the district consists of private households.

The forest is situated at the south west and the east side of the district and - assuming the algorithm for tree top detection also works well in forest environments with densely packed stands - is quite dense. The housing takes place only in the northern part, with few exceptions in the east. Two clearings can be observed in the map. The one on the East can be explained by a waste disposal site. The one to the South are fields, which seems to be fallows. A huge supposed building, situated at the northwest edge of the district, turns out to be a part of the highway (see also the upper right map on figure 4.6). This problem can not be solved by this method, because it highly depends on the results of the *nDEM*, where man-made objects like bridges or ramps are included. Those error are only revealed by visual interpretation and an automated detection can not solve this problem.
Figure 4.6 will show in detail, how accurate the method works. The four subsets

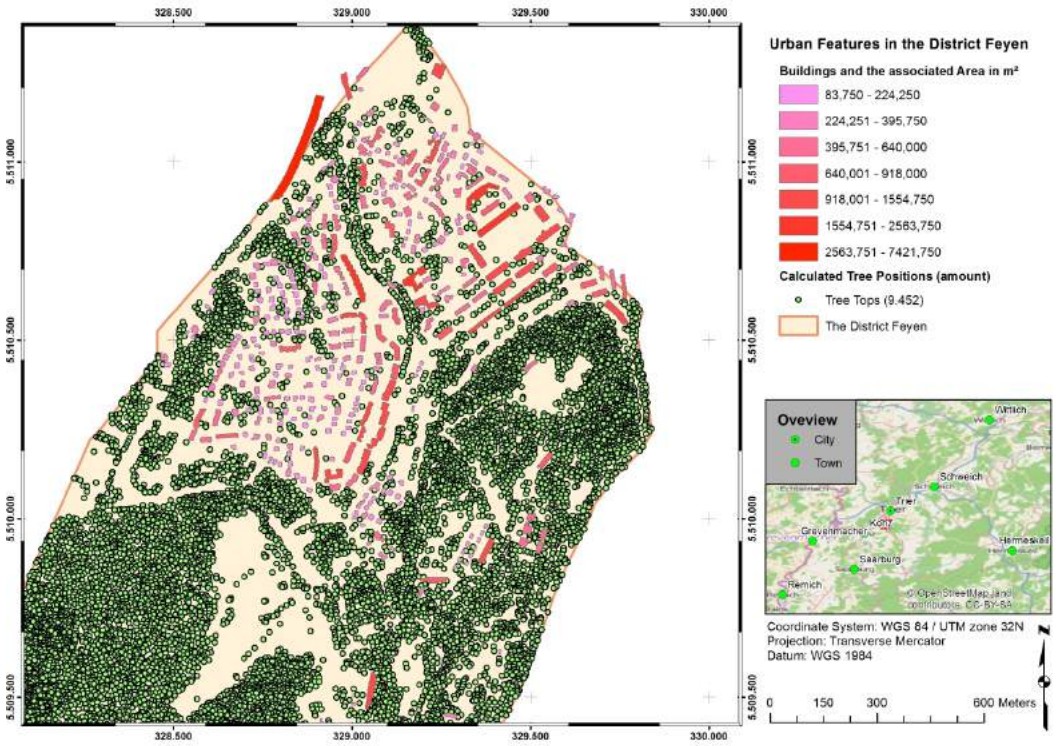



Figure 4.5.: The results of the building and tree top extraction for the district Feyen

represent the district well. While two show the urban part, the other two demonstrate the results from the view point of a more rural environment.

The upper left map shows the earlier mentioned highway ramp, as well as a bridge on the North. Both objects are man-made but are not regarded as buildings. Reducing this errors could only be possible by adding additional data, for example a file containing all the roads, to remove objects, intersecting with those roads. It seems, that the trees in the western part of the subset are not recognized, this is due to the fact, that they are beyond the border of the district. Other trees can be found in the map although a problem with the overestimation can be seen, when looking at the southern part, for example. Regarding the accuracy of the building segmentation, it shows, that some smaller buildings are missed. Those are mainly garages or sheds.
The upper right and the lower left subsets are showing the urban housing area and former barracks. It can be demonstrated, that the results here are accurate regarding both buildings and tree positions. On the other hand, the subset showing a part of the forest (lower right) indicates a huge overestimation for dense forest environments. This problem could be solved by increasing the window size of the local maxima algorithm. On the other hand, small trees are missed, which reduces the accuracy in urban areas.
Combining those findings, it is shown, that areas with dense forest can be a problem


Figure 4.6.: A detailed look on the results of the building and tree top extraction for the district Feyen

when trying to estimate the amount of trees in a district. The section 5.3 will deal in detail with possible improvements. On the other hand, the recognition of single family houses works well. Not only all houses are detected, but also they are separated and thus the ability to assign additional information to those houses is easy accomplished.

## 4.3. Results for the district Gartenfeld

This section will introduce the district Gartenfeld. It has with Feyen in common, that both are mainly housing areas with a few retailers. To the East an area with forest can be seen in figure 4.7. The southeastern part shows some larger buildings, which are a monastery. In conclusion, the district is divided into two parts. One is the housing area to the West, and the forestry area to the East. Figure 4.7 picture these differences. The housing in Gartenfeld has more similarities to the one in the Altstadt district. The buildings are tightly packed together, indicating a more urban area. By comparing the density of Feyen and Gartenfeld with the help of table 1.2, the assumption can be proven, because Gartenfeld is 4,6 times denser than Feyen. Those areas seems to be on the western part of the map, which is situated near the

city centre. The northern part consists of buildings, which can be separated more easily, indicating an area with either lower population density or higher housing prices.

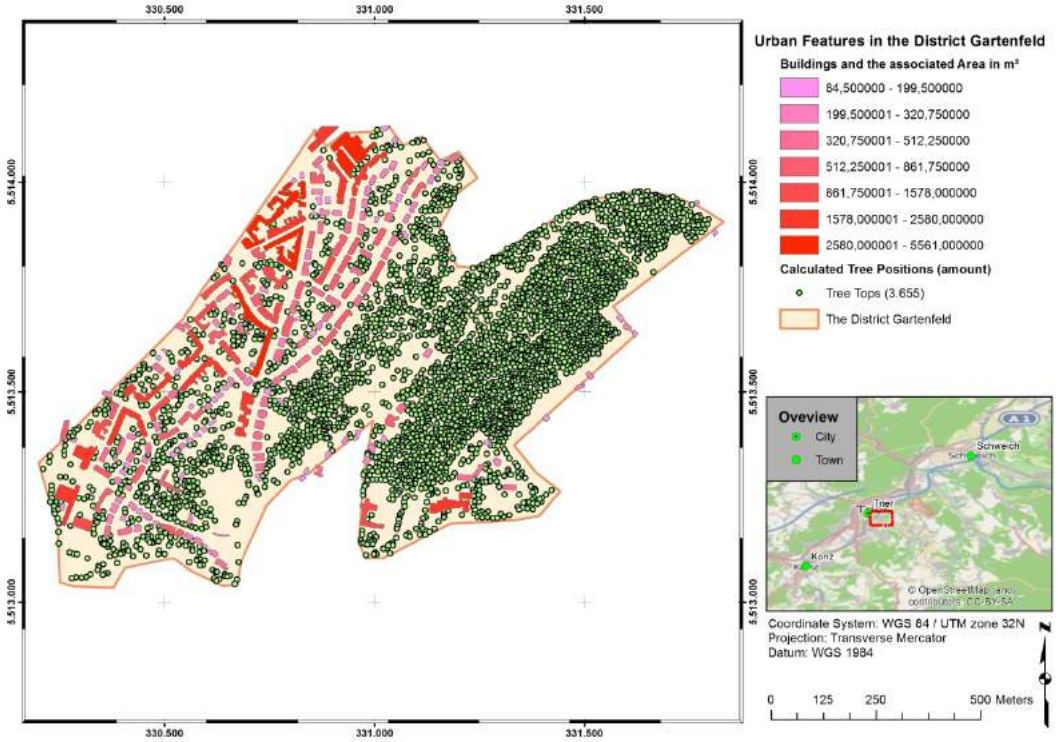



Figure 4.7.: The results of the building and tree top extraction for the district Gartenfeld

The four areas picked in the detailed look on this district are pictured in figure 4.8. The upper two are showing the urban part, while the lower two focus on the more rural characteristic of this district.

The upper left map shows a part of the main station and its railroads, which are the northern border of the district. Here the building are detected well. The same holds true for the tree tops. The positions seem to be accurate. Only some smaller trees are left out. This could be caused by shadows created by nearby buildings. A small overestimation is shown in the eastern part between two large buildings. The upper right map is situated South of the first map and also shows part of the main station and the housing area. While tree position are computed, the building part seems to miss a huge proportion of a structure on the West. The lower left image shows the ancient Amphitheatre in the centre of the map, surrounded by a small forest, which seems to fit good with the calculated model. The buildings are aligned with its real counterparts, which is proven by rich details on the eastern buildings for example. The last detail map on the lower right portraits the monastery and a part of the latest district called Petrisberg. As in the other three maps, the buildings outlines

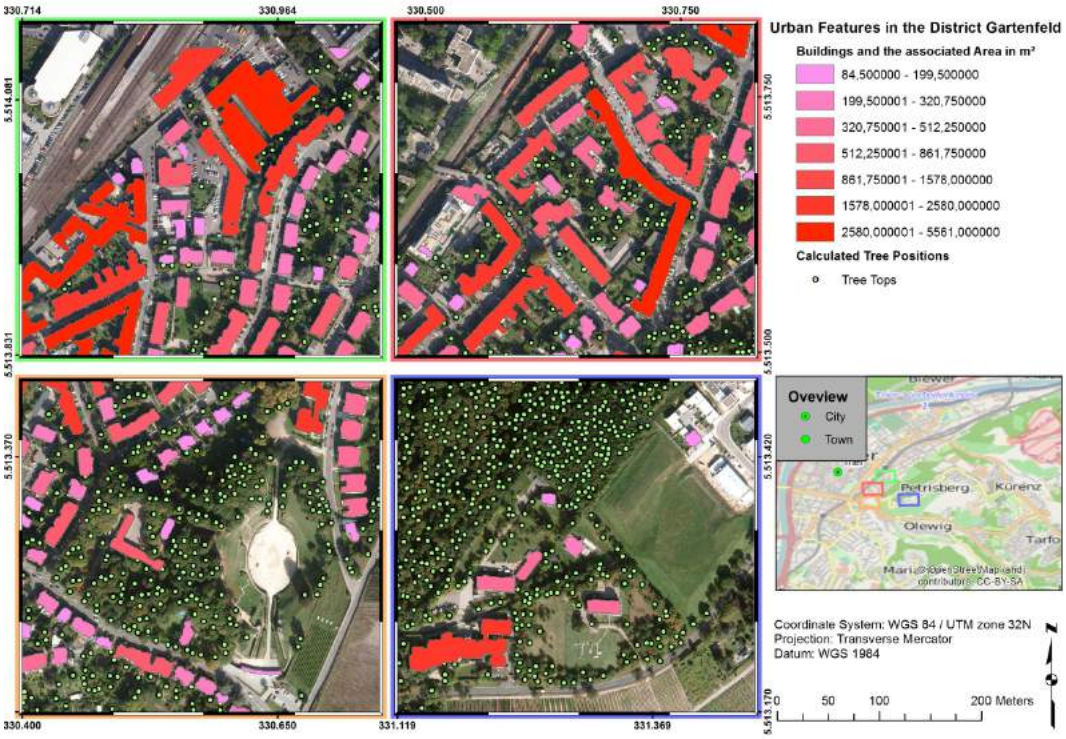



Figure 4.8.: A detailed look on the results of the building and tree top extraction for the district Gartenfeld

are met precisely. The structures missed are seen on the northeastern edge of the image. This has two reasons. First some of those buildings does not belong to this district, and second, the *nDEM* and the aerial images were taken on different dates. Which means, that within the *NDVI* image, the buildings are recognizable but the height image does not include them, leading to the seen result. Those mismatches are presented in figure 4.9.

## 4.4. Results for the district Maximin

The district of Maximin has the second highest population but is first concerning the density. 9.114 people living on 1,673 *sqkm* resulting in a density of 5.447,669 people per square kilometre. The map in figure 4.10 underlines the statement with two facts. First the amount of trees detected inside the borders of the district are low, indicating either an environment which consists mainly of bare soil or man-made objects. The latter assumption is the case, as the map shows. The district is bordered by the Moselle to the West, and to the East by railroads. In the South it shares a border with the in section 4.1 explained district. In the northern edge of Maximin, the football stadium of the local team is located. The main amount

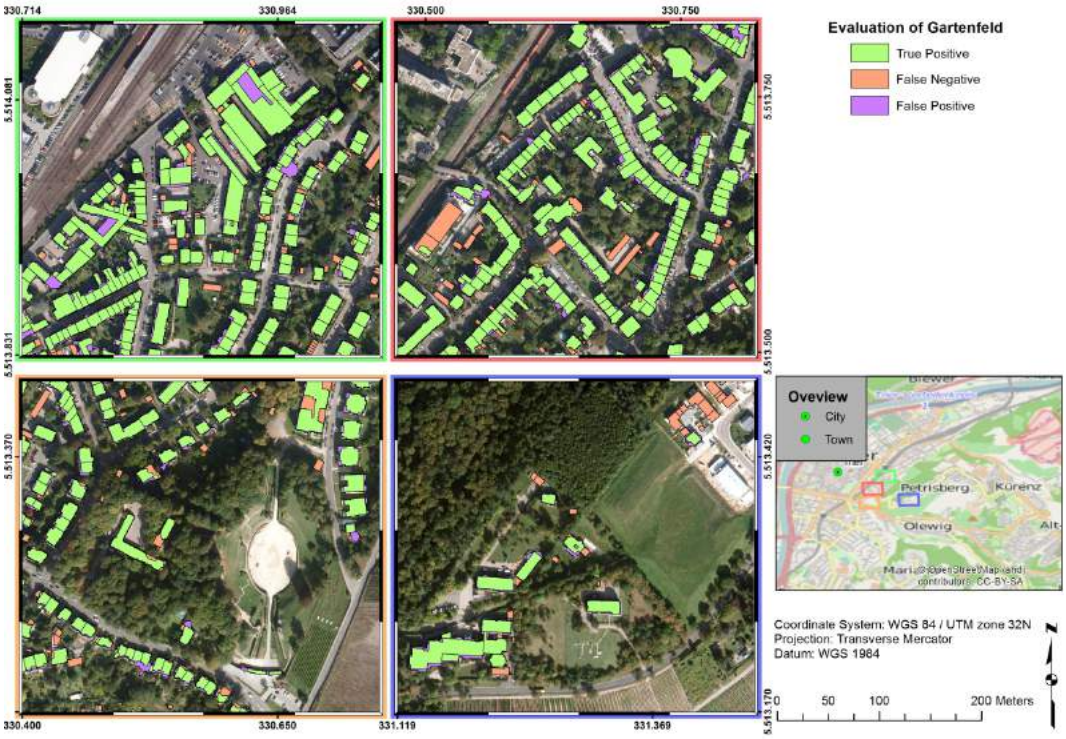



Figure 4.9.: Evaluation of the district Gartenfeld with four detailed maps; Light green = true positive, orange = false negative, purple = false positive

of trees can be found alongside the Moselle and to the North behind the stadium. The centre of the district is characterised by densely packed buildings, indicating an apprentice of businesses. The two rows of trees on the northern edge are showing the borders of a four line highway leading through Trier.

The four maps, which show details of the district is pictured in figure 4.11. As previous the overview map on the right indicates, where the four subsets are situated. All are showing, how urban Maximin is. The two upper ones demonstrating the look of a business district, while the focus of the lower two are on housing.

The upper left map is showing how the algorithm works in densely built up areas. While not able to differentiate between houses, which are build next to each other, the algorithm preserves the shape quit good. Problems can be seen, when trees are growing close to a building. In such a case, the proposed method has difficulties to distinguish between actually man-made objects and trees. The tree positions calculated are good for the western part of the subset, but are overestimating the trees on the eastern part.
The upper right map shows a logistic centre with an associated storage. Those buildings are hit very good due to the fact, that no green areas are surrounding them. The trees alongside the railroads are accurate but the amount of trees on the eastern

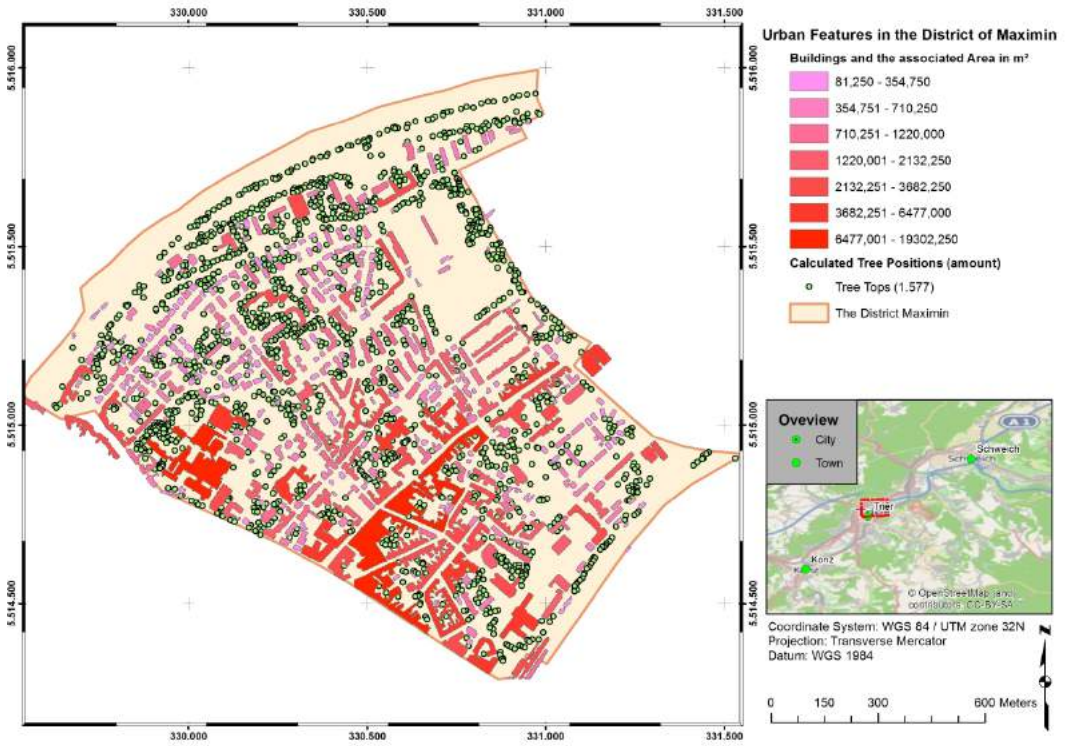



Figure 4.10.: The results of the building and tree top extraction for the district Maximin

edge of the subset are clearly overestimated.

The map on the lower left shows an intersection and parts of a bridge. The building in the southwestern corner is outside of the district and therefore not detected. Considering the tree tops, it can be said, that the estimation is accurate. Nonetheless the buildings, although detected good, have problems with irregular borders, which holds true for the northern part of the map. This problems seems to increase, when an object is surrounded by trees. The last map shows the strength of the algorithm. Every building is detected, the boundaries are regular and following the real buildings, the tree positions are good as well as the amount of trees. Some overestimation in the centre of the map can be observed.

In summary, the results reflects the pros and cons of the algorithm in a way, that improvements can be considered. It is shown, that the method works well in this urban environment, which can be seen as typical for German cities with this population. So it is proven, that with a general rule to extract buildings and trees, the results are fairly good and can be used to work on similar structured cities.

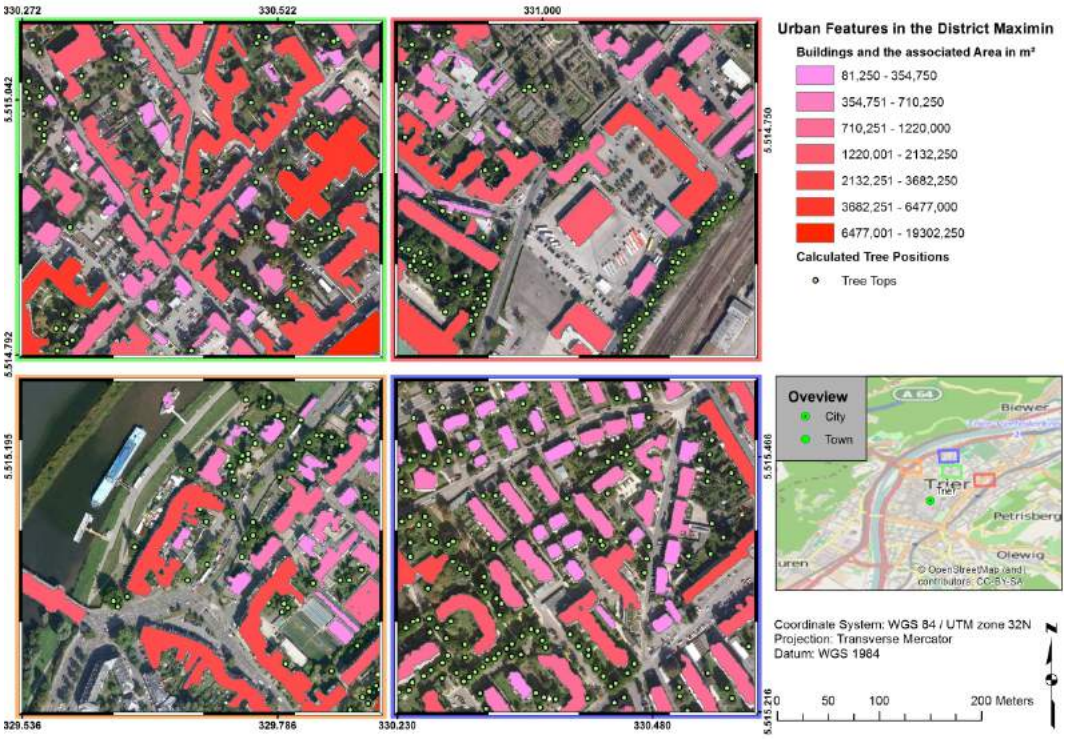



Figure 4.11.: A detailed look on the results of the building and tree top extraction for the district Maximin

## 4.5. Results for the district Pallien

This district is characterized by a huge forest, which grows from the northwest to the southwest part of the district. Due to the fact, that Pallien is situated on a edge of the slate mountains, which divides it into two parts, the core of the settlement is on the shore of the river Moselle and only some minor settlements including the ˜Hochschule Trier˝are situated on the hillside. On the map presented in figure 4.12, this diversity can be seen. The eastern part is predominated by buildings and in the centre the bridge. The western and the northern part are covered by dense forest. On the first look, the man-made objects are represented well. Only the clearings to the northwest and in the centre of the image seems to be unclear. By looking at the true colour images, the reasons for this clearings are revealed. The northern part has some fields, on which crops are growing. The parts North of the ˜Hochschule Trier˝are sport fields for football and tennis. The spaces on the southern end of Pallien could be fields as well or just lawn but it can not be said clearly only from the images. At first sight, the amount of trees seems to be very high. Due to the fact, that exact numbers to validate the results are missing, an evaluation of the result is hard to conduct. Ways to reduce those insecurities are discussed in the following chapter.

The rural frame of the district is underlined by its very low population density with just 280,545 people per square kilometre, which is also the lowest value of all six researched districts. There few retailers, which may be also a reason, why this district is in the program "Soziale Stadt"alongside with Trier West (Trier 2015b).

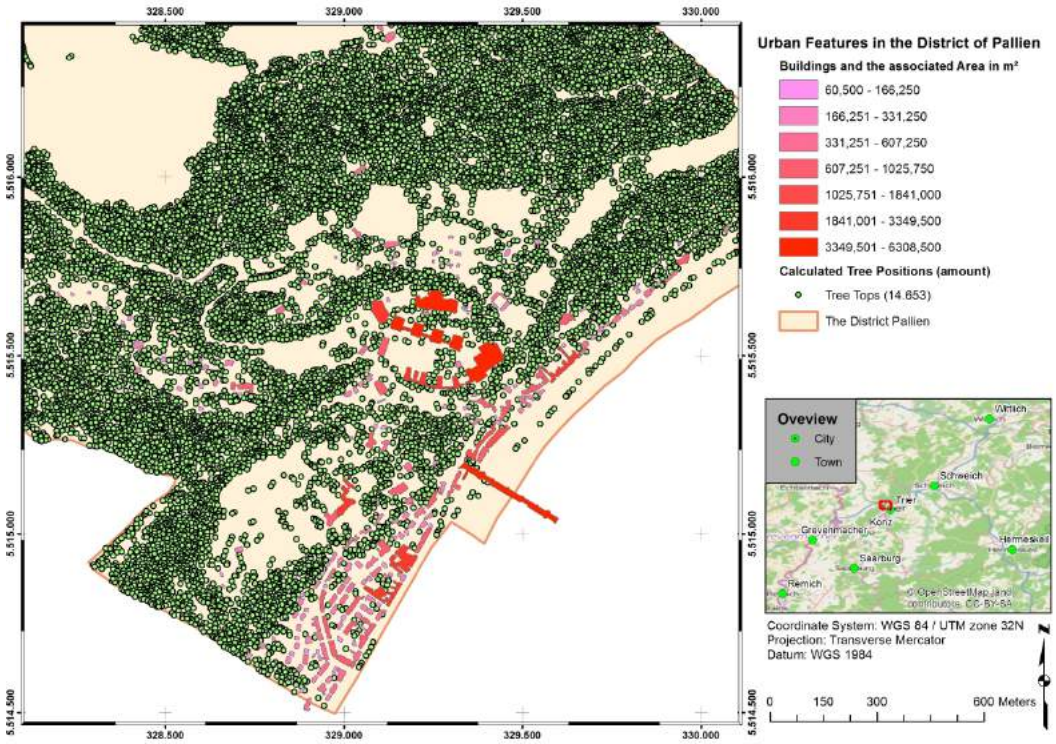



Figure 4.12.: The results of the building and tree top extraction for the district Pallien

To have a closer look on the different aspects of this district, the four maps in figure 4.13 are showing different point of views for this particular side. Three of the four subsets are showing the core of the settlement, while one is situated at outskirts of Pallien.

The upper right map shows an urban part of the district. Alongside the street on the eastern part, the buildings are packed together, which impede the ability to distinguish between the buildings. The outlines of the buildings are well met. The algorithm tends to have some difficulties on the northern edge. The buildings on the East are outside the borders of the district. The trees inside the settlement are hit well. Even on the edges of the urban area, the trees are hit well, if the spacing between them are wide enough. The same holds true for the other three maps.
The upper right map shows a farm with several outbuildings. They are portrait in a good way. An bridge over a street is outlined good. The trees, which have a good spacing, are well hit, the trees alongside the street on the South. A thing, which can be proven on this particular map, is that the clear spots are good covered. So no


Figure 4.13.: A detailed look on the results of the building and tree top extraction for the district Pallien

trees or buildings are set here.
The bridge on the centre of the lower left map is the Kaiser Willhelm bridge. The last map shows the "Hochschule Trier"with some outbuildings alongside. They are represented precisely. Even the small connections between the buildings are outlined. Only the building to north has some parts missing. As stated at the beginning of the chapter, Pallien was the only district, where the rules had to be adjusted in order to enhance the results. This building was one of the main drivers for the poor results. The trees and their amount seems to fit, a tree in the centre of the map has three instead of one tree top. This can be explained by the fact, that this tree has a large canopy and thus the window size was set too small.

The next example, of where errors occur and the reasons for those are the figure 4.14. Here a whole complex of buildings are regarded as false positive (upper right subset) and hence increase the amount of errors regarding false negative and positive areas. This is also reflected by the lowest detection rate with only 73,28 %. The source of this error could be a flaw in the height model for the buildings and a lack of reference for the bridge (a more detailed look can be found in section 5.2).
Although this district has the lowest true detection rate, the results are promising. With the dense forest to the West and North, the amount of tree tops are overesti-


Figure 4.14.: Evaluation of the district Pallien with four detailed maps; Light green = true positive, orange = false negative, purple = false positive

mated but for the urban parts, the results are fairly good. This underlines the fact, that the method is suited for an uneven terrain.

## 4.6. Results for the district Trier West

The last district, which will be portrait is Trier West. It is divided into a western part predominated by forest, and an eastern part, where the core of the settlement is situated. With its 5549 inhabitants, Trier West is the third largest district, But due to its size, it only has a population density of 1787,693 people per *sqkm*. The former military facilities are converted into low price housing. This leads to a social segregation, which is also a reason, why Trier West is part of the earlier mentioned program Soziale Stadt (Trier 2015b).
Trier West faces the same geographic conditions as Pallien does. With the western portion being part of the slate mountains and the East being the main settlement area with the Moselle at its border. It has wide spaces between buildings, especially in the southern part, where large scale buildings are placed. Those are current or former industrial facilities. This partition between the two portions are well pictured in the figure 4.15. Alongside the huge industrial buildings, the residential buildings

are packed together, indicating a dense population with high rise buildings, which holds true for the northern and the southern part of the settlement.

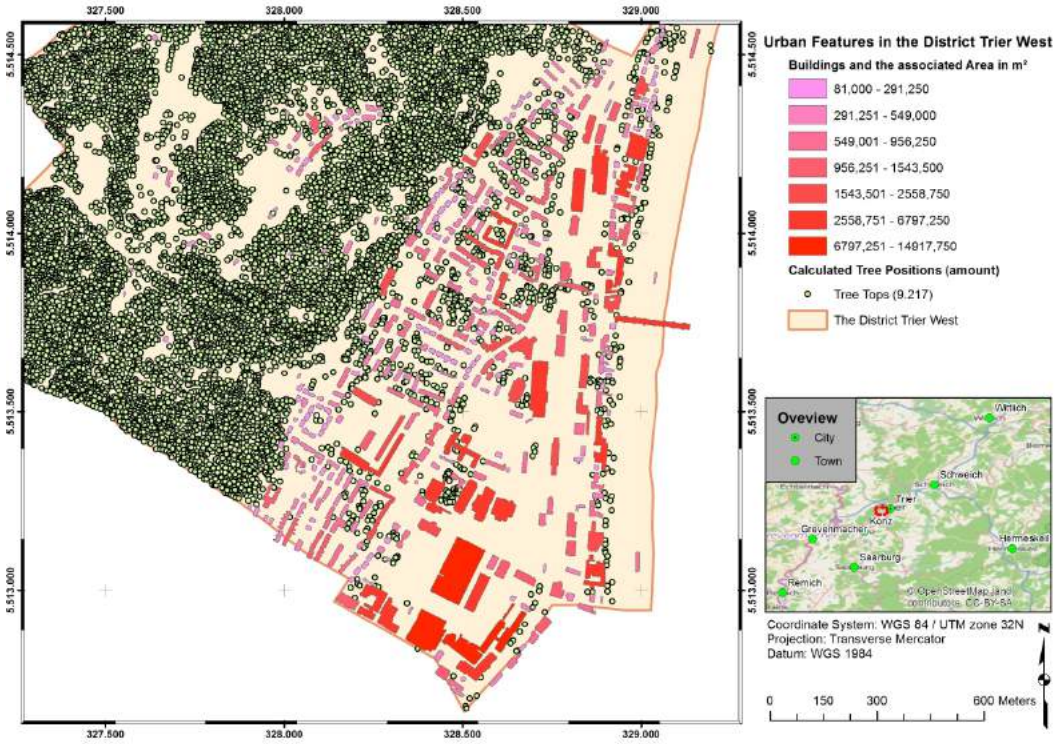



Figure 4.15.: The results of the building and tree top extraction for the district Trier West

The detailed look, which will allow a more sophisticated analysis is presented in figure 4.16. The focus on the examination lays on the urban centre of the district and again on showing the pros and cons of the proposed method.

What all four maps have in common, that the algorithm preserved the shape of objects in a very good way, showing connections between buildings (upper right map) as well as the regular edges (upper left and lower right map). The upper right map shows, that all man-made objects are hit, with even the smaller buildings outlined. Concerning the tree tops, it seems to be again an overestimation, which can be called a systematic flaw, and will be discussed in the following section.
The upper left image shows a snipped of residential buildings, where smaller facilities are missed out (centre of the map). This can be due to the low hight or the surrounding green areas. The amount of trees are met well, with just some overestimations in the north western edge of the map for example.
The lower right map shows some problems, which appears, when a rule is set to strict. The buildings to the West of this subset are low rise allotments. They are too small and too low to be extracted. Alongside the railroads, some objects with white roofs are not detected. Either they face the same problems as the allotments, or they

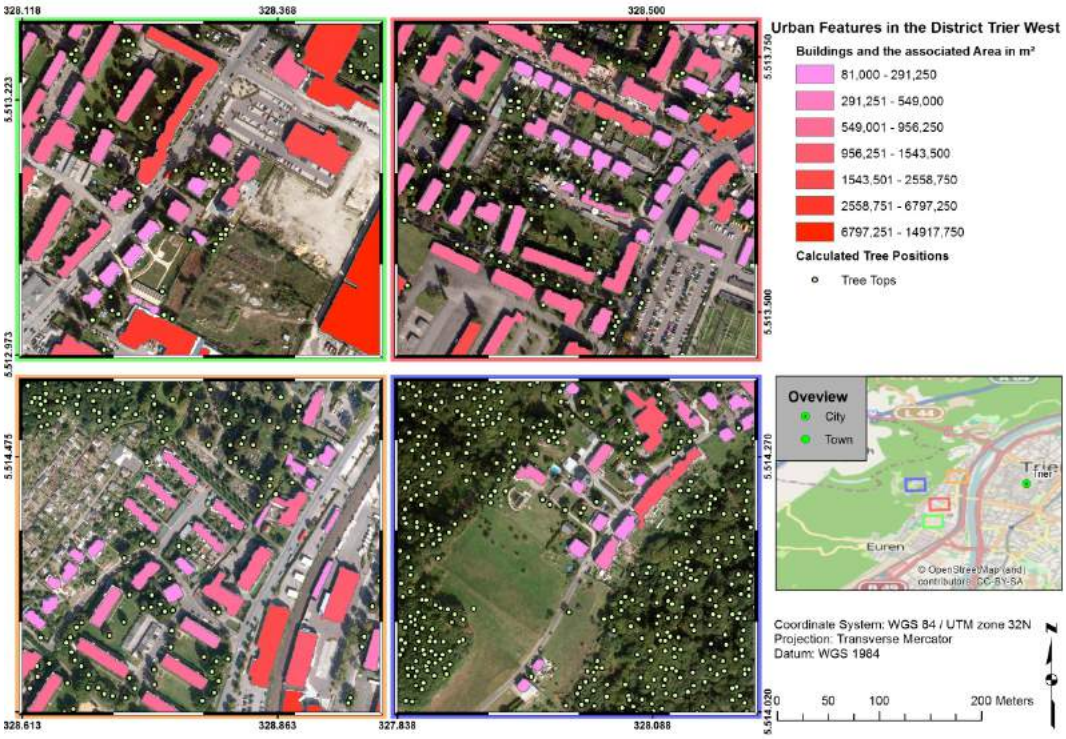



Figure 4.16.: A detailed look on the results of the building and tree top extraction for the district Trier West

are visible on the true colour images but non existent on the *nDEM* and therefore ignored. By looking at the trees, it can be proven that the method works good in this conditions, when trees can be separated and are not packed tightly together.
The last subset shows a residential area near the edge of the settlement, with a surrounding forest. The main buildings are sharply outlined and some outbuildings are missed. Again the reasoning can be, that the rules are set too strict. On the other hand, if the knowledge base would try to compensate this flaws, the amount of false true detections would increase and therefore the overall performance would decrease. The trees are again well met, if they have a certain space between them. This can be proven by looking at the centre of the subset, where the trees alongside the roads and the inside gardens are all detected correctly.
In summary the district Trier West shows, that the method works best in urban areas with buildings and bare grounds. Trees, being inside a forest or a denser stand in a park for example are harder to distinguish because of the different aspects like height, age of the tree and derived from it the area of canopy. To tackle this problems will be theme of the next chapter.

# 5. Discussion

This part of the thesis will explain the flaws as well as the pros of the proposed method, it will line out the things, which went well, and those which need improvement. The three main parts will be discussed, starting with the program, followed by the evaluation of the buildings and finally the position of the trees. The results will be demonstrated with numbers and maps to prove and strengthen the point.

## 5.1. Evaluation of the Program

The first focus will be on the program itself. The way the program works as well as the handling of the data will be discussed. Concerning the results, the following sections 5.2 and 5.3 will cover this subject in the needed depth. As in the section 3.3 shown, the order will remain the same, by starting at the main window and from there to the following tabs and discussing their content and the pros and cons.
The main window as presented in figure 3.3, is the starting point for all other options. The user has to set the input data at first, for having the ability to access the further functions. Following the previous pattern, the section a) could be improved in such a way, that not only height data such as *nDEM* and its preproducts are allowed, but also other forms of data. A combination of a *NDVI* and an *EVI* image as ancillary data could be a sufficient way to extract tree tops from such data. The problem, which occurs here is, that the next steps are based on the assumption, that the input files are height data and an additional file, which is optional, which h as the ability to discriminate against men-made objects and natural things like trees and bushes. The second problem, which can occur is, that the user has to have knowledge about the so called EPSG code (acronym for the European Petroleum Survey Group), which handles the coordinate system in which the data will be projected into. While the argument for this is, that it ensures a maximum of flexibility and the automated use of the current coordinate system, the counterargument can be, that the user can't insert the coordinate system, from another file. The same holds true for the no data value. The standard setting is, that the program uses the one given by the first input file. The section labelled with b) let the user decide, which ancillary data and if he wants to use it. The program checks on this stage, if the cell size is correct, and if the coordinate system are compatible. The options

to create a *NDVI* image could be extended to allow additional indices to be create, like for example the *EVI* or *PVI*. This would include further options and therefore would increase the complexity of the program. For the last section labelled with c), it only be said, that the outputs are only in the *.TIFs* format. This holds true for the input data. Only *.TIFs* data are allowed. This decreases the flexibility but ensures, that the work flow can be handled.
Next on discussing the program, the tab for the buildings options are analysed. By referencing the figure 3.4, the discussion can be put into perspective. By using a knowledge engineering approach, the user needs to have at least a decent inside in the field of geoinformatics as well as urban planing. If this is knowledge is not present, then the results will lack in precision and validation. The user can set an unlimited amount of assumptions, with the logical rule, that they does not contrasting each other. Here is the first downside the program faces. It lacks a error-reduction option, which can identify such rules and gives the user a feedback. The filter options are self explaining. The values for the x- and y-axes are disabled, when the checkbox is not activated. The information, what those filters do, can be taken from the *Help* tab. The last option ensures, that false detection is reduced and can be either used or ignored. An argument can be, that only one rule is applied, but no range can be set, when only residential buildings should be considered. To summarise this tab, It could be good, if the feedback loops would be implemented, but all at all the functionality is straight forward and allows to make fast decisions.
The next part is dedicated to the tree filter tab. As demonstrated in figure 3.5, the user needs to have the same abilities as in the earlier mentioned building tab. The knowledge based assumptions following the same pattern and have therefore the same pros and cons. The only filter, which can be applied here is the closing filter. The other filters could be implemented as well but they do not fulfil any need for this kind of task. As the previous noted value for the minimum value for the building size, the critics can be said about the options in the tree filter tab. The values allowed no data and critical height could be a range instead of a threshold value. The window size looks at the number of pixels and not at the unit of the image (metre or centimetre).
The last tab is the is the help section of the program. Here the user can have information about the functionality of the program. This information contains only of text. By adding images, the help could be more precise. Another improvement could be, that hyperlinks to sites could be used. In conclusion, the program can handle the data provided (see 3.1), it can lead the user through the workflow and prevent to a certain degree, that errors are made. Those feedback loops are controlled by text field on the bottom of the program. By adding background-colours to indicate, if the values are logic and are allowed, the user could get track down errors more straight forward. All in all, the program is a base, on which further developments

can be taken place. The results, which it produces are promising, as the following sections will prove and therefore an extension of the program could be reasonable.

## 5.2. Evaluating the Buildings

The chapter 4 deals with the results, while ignoring the exact amount of falsely detected buildings. This section will focus on the results and will name the sources of errors and ways, how those can be prevented. In addition to this, some maps will focus on the different aspects of false detection and will line out the reason, why the algorithm might not have worked correctly.
The evaluation will be guided from the methods, which are used by Khoshelham et al.. They used seven different metrics, to asses, if their algorithm worked. By testing different setups, the author could peal out the method, which worked the best. In the case of this theses, the choice of method is already done and thus only appropriated ratios will be used, which reduces the number of metrics to five.
All The numbers and conclusions, which will be presented here, are drawn from the table in the appendix A. The abbreviations are used in all formulas. So TP stands for true positive,TN is the abbreviation for true negative, FN for false negative and FP for false positive. The first ratio, which will be used is the detection rate(DR). It is defined as:

$$DR = \frac{TP}{TP + FN} \tag{5.1}$$

The ratio basically controls, whether a pixel is correctly assigned as building pixel or as background. In the original formula from Khoshelham et al., they had the variable UP, which stands for unclassified positive. This is left out, because in this model, there are either buildings or non buildings. The next ratio, which allow to see, if a pixel, which is classified as pixel, is acutely a building. This metric is called reliability or user's accuracy. It is defined as:

$$R = \frac{TP}{TP + FP} \tag{5.2}$$

This equation ensures, that the results have a certain precision. The next formulas are the False negative and false positive rates, abbreviated as FNR (see 5.3)and FPR (see 5.4).

$$FNR = \frac{FN}{TP + FN} \tag{5.3}$$

$$FPR = \frac{FP}{TN + FP} \tag{5.4}$$

Those metrics are a measurement for the pixels, which are wrongly classified as background (FNR) or buildings (FPR). Those ratios allows to see, if the algorithm has problems of distinguishing between different types of uses (either buildings or non buildings).
The last ratio, presented in the formula 5.5, is the wildly used one. It shows, how good a method can discriminate men-made objects against other objects such as trees or shrubs.

$$OA = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.5}$$

Those five metrics allows a more specific view on the results. If one wants to compare different methods, this approach ensures, that the evaluation has a strong basis, and that all aspects can be analysed (Khoshelham et al. 2010). All five metrics are multiplied by 100, to get the percent values. From this start point, the thesis draw the view on the results, presented in the table 5.1.

| District | **R** | **DR** | **FNR** | **FPR** | **OA** |
|---|---|---|---|---|---|
| Altstadt | 94,075 | 82,826 | 5,925 | 7,273 | 90,241 |
| Feyen | 83,315 | 78,113 | 16,685 | 1,550 | 97,343 |
| Gartenfeld | 82,300 | 83,501 | 17,700 | 0,359 | 99,250 |
| Maximin | 84,914 | 82,595 | 15,086 | 2,291 | 95,759 |
| Pallien | 84,157 | 73,280 | 15,843 | 3,118 | 95,219 |
| Trier West | 89,260 | 88,329 | 10,740 | 0,918 | 98,244 |
| Average | 88,814 | 82,833 | 11,186 | 1,821 | 97,058 |

Table 5.1.: Evaluation of the results by different metrics, values in percent

The table proves, how good the overall performance of the algorithm is. The OA metric has an average value of 97,058 %, with 99,250 % as maximal value and 90,241 % as lowest value. The reliability is with 88,814 % very good and in combination with an very low average FPR value of 1,821 % implies, that the reliability of the results produced by the method is high (Khoshelham et al. 2010). An downside of the methods seems to be amount of false negative detection, which is too high with 11,186 %. The detection rate is with 82,833 % okay but leave space for improvements. The overall accuracy is satisfying.
The relative large amount of false positive, which was detected in figure 4.3, can be explained in two ways. First, the closing hole filter was set too strong and therefore some yards too narrow are regarded as part of the building which surrounds it (see all subsets of 4.3 expect the lower right). Second, some buildings are not present in

the validation dataset, because they weren't build, when the validation was done. This is the case for a building on the north western part of the lower left subset, although this district has the lowest FNR and FPR rates (see table 5.1).
For the district of Gartenfeld, portrait in figure 4.9, the findings stated above remain the same. Due to lack of enclosed yards, the false negative detection is the lowest of all six researched districts (only 0,359 % FPR). The algorithm had problems with the building detection, which resulted in the highest false negative rate (17,7 %). The reasons are, that residential buildings are dominating this area, which results in smaller objects and thus the removal of those buildings seeming too small. The figure 4.9 shows again, why data consistency is an important issue. The lower right map shows, that several buildings are missing, which lead to the conclusion, that those buildings are missing in the *nDEM* or *NDVI* images. Another source of errors may be a rule set too strict or too loose, causing an increase of either false negative or false positive.
To summarise the findings, three reasons for errors can be identified. First, the filter window are set too wide and therefore regards enclosed yards as buildings. Second, errors in the input datasets, for example wrong hight informations. Third, too strict assumptions for the extraction rules, and last and most important, the inconsistency of data, by the means of temporal consistency.
So what can be drawn from this findings? The first conclusion is to find data, which is temporal consistent. This allows to conduct time-series and reveal changes in the structure of the buildings and shows, where heavy development is taking place. The second thing, which one have to keep in mind is, that rules and the filters largely affects the results and therefore have to be chosen carefully. The fields of improvement for this method is wide. A first approach can be the addition of further filters to give the user a wider range of choices. The second option is to add user crafted filters, which remove foreground pixels inside a sliding window. This could be done by a similar approach as presented in section 3.3.3. The amount of background pixels has to identified and then the program will decide, if all foreground pixels will be switched to background. Other forms of filters, which detect certain forms and smooth them, in order to get sharper edges and better building shapes could also be an improvement. An algorithm, which identifies jumps in the height could a possibility to distinguish between buildings build next to each other. The final solution could be a change from a pixel approach to an object based one. This would need a complete makeover of the program and should be the last option.

## 5.3. Evaluating the Tree Positions

This section will deal with evaluation of the tree positions. Due to the lack of validation data, the results produced by the proposed methods have to be evaluated by hand measured reference data. Those were taken on the 4. of February 2015, and includes 42 trees in the district of Altstadt. An hand-held GPS by Garmin is used to determine, where the trees are. With the devise, it is possible to access the GPS and GLONASS satellites. This ensures, that the localisation of points is handled appropriate. The points are measured in geographic coordinates. The devise allows to take several samples from the same spot. The results are than middled, which increases the accuracy (Garmin 2011).
To conduct the evaluation, the measured points are compared to the ones automatically created. This is done by two different means. The first is to check, how near the points are. If they lay perfectly on top of each other, than this value should be zero. To test this, the near function from *ArcMap* is used. It produces a table with column *NEAR_DIST*, which contains the nearest distance to a reference point. This measurement shows, that in the mean, the trees tops are 3,054 metres away from the ground truth data, with a standard deviation of 2,28 metres.
To illustrate this bias, the figure 4.4 uses the similarity search from *ArcMap*, to rank the points according to their similarity. The output contains a field called *SIMINDEX*, which represents the sum of squared values and quantifies, how similar each solution is to the target feature. The mean is 2,096, with a standard deviation of 1,803, indicating a good result.
The reasons for inner-city misses can be named as following, with the distinguishing two main sources. The first source is the data, and can be tracked back to the same as reasons, as identified in the section 5.2. The temporal mismatch of the height and the ancillary data produces errors in such a way, that in the height model a tree is present, but on the *NDVI* image, the ground is bare soil. Another could be, that a tree was present both, in the height and the ancillary data, but was cut down before the ground truth data was measured. Finally, a reason can be, that due to the change of seasons, a tree already lost its leafs and is therefore not recognizable in the ancillary data.
The second source may be the downsides of the method itself. This can be poorly chosen assumptions, which will lead to either over- or under-representation of foreground pixels and thus to too much or too less detected trees tops. It can be also a stand, which is too densely packed and thus make it hard to distinguish different trees. A wrong choosing of the window size (see 3.3.3), which can be seen as a misinterpretation of the age distribution of the trees, can cause problems as well.
To compensate those downsides, different approaches can be chosen. The most obvious one is to evaluate the knowledge engineering approach and adjust the rules

according to the findings. This means in detail, that the rules for choosing the thresholds has to be checked. The window size largely influence the amount of detected trees and therefore has to be chosen carefully. Depending on the age of the stands, the window size will affect the accuracy. The amount of allowed no data can be a factor, on tree tops, whit a small footprint, if the value is set too strict. In dense stands, it has nearly no effect. An problem can be occur, when dealing with small trees, when the critical height is chosen too high.
While the thresholds may be drawn from the literature, the other values are depending as well on the data as on the exposition and the place of the study side. Fitting values can only be drawn from experience and trial and error. This may be a downside of this method, but can also regarded as pro argument, because the results will improve by repeating using the program and thus gaining experience. An improvement of the method could be the usage of changing window sizes, driven by the maximum value inside a window. This could reduce the errors of a window size chosen too small. The program would increase or decrease the window size if the highest value is in a certain range. This range would be again provided by the user. Although the method proved to be robust and can detected trees in urban areas very accurate (see 4), the space for further improvements is present and advances can be made in future studies.

# 6. Conclusions

The last chapter will summary the thesis and checks, if the hypotheses, conducted in section 1.3 are met and will finally give an outlook for future works which could be conducted in this field of study.
The study area was chosen by the availability of data, which imitate the choices to the cities of Rhineland Palatinate. Due to familiarity with the city, Trier was an obvious choice. To reduce the amount of data and due to other the limitations, the next step is to look for six districts, which differ by the means of statistical data. Those data are the values for the housing prices and population and two districts with low, mean and high values are chosen each. From here the data will be reduced and fit to the borders of the six chosen districts. Than, the methods are applied to the remaining data. The first one is the knowledge engineering, which is used for both, the extraction of buildings and the determination of tree positions. The next steps differ according to the different results, which they will produced. The extraction of buildings will use additional filters, which reduce the errors. Than the rasters will be converted into vectors and remaining polygons, which are smaller than a certain threshold will be removed. The tree position will determined by the combination of filters and a moving kernel approach.
The results, which are produced proved, that the program and the method used are capable of extracting the building and the trees with a good precision. The downside can be seen, when trying to determine the position of trees in dense forest stands. The problems for buildings can be seen in the mismatch of the height and the ancillary data regarding the temporal consistency. The program *CityEX* proved to guide the user through the processes and therefore is a good tool, compared to using only the single scripts.
The hypotheses could be proved just in partition. For proving the main hypotheses, that cities can be compared by the means of remote sensing, the problem to prove this idea is, that data is missing for other cities. Thus a comparison between Trier and other cities, with similar population and extend couldn't be conducted. The work around for this was, to compare different districts. This could prove, that the amount of trees does not have a influence on the housing prices, because districts, whit an nearby forest biased the results. The density could not prove the hypotheses either. It showed, that it not an indicator for housing prices. The best example for this is the district Altstadt. With a portion of 38,61 % it has only the third highest

ground prices. The next hypotheses, which says, that s there are statistical and physical variables, which can determine the extend of a city, proved to be true. The statistical values extracted are the mean ground prices, the population density and the total number of population. The physical values are the amount of tree tops, the area of green and the area of buildings inside a district. The third hypotheses, asking, if trees and buildings can be distinguished between other urban objects could be proven as well. The results from chapter 4 and the evaluation of those findings in chapter 5 prove, that the methods used performed well. The last hypotheses, could be proved. It stated, that different district, which can be distinguished with the means of statistical analyses are also separable with the tools of remote sensing. In conclusion, the aim of measuring different cities failed due to the lack of data, but the other aims could be proven. The framework of the whole thesis was set by the program and the evaluation was carried out by statistical approaches.
Future approaches should focus on the uses rather then on the way to create the data. This uses can be a temporal stack of information, which allow to detect changes and thus make it able to guide those changes into directions, which urban planers, business owner and politician, want them to change. The program, the approach and the combination of data allows a fast and coast efficiency decision making processes. Those are important in countries, where no plans are available, for example in slums. Because those countries often lack in infrastructure and institutions to monitor the development of their cities. The aim of further development has to be, to tackle those problems and improve decision making processes by providing straight forward data products.

# Bibliography

Bilal Al Momani, Sally McClean, and Philip Morrow. Using dempster-shafer to incorporate knowledge into satellite image classification. *Artificial Intelligence Review, 2006, Vol.25(1), pp.161-178*, 25(1):161, 2006.

R Attarzadeh and M Momeni. Object-based building extraction from high resolution satellite imagery. *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 1:57–60, 2012.

M Awrangjeb, M Ravanbakhsh, and CS Fraser. Automatic detection of residential buildings using lidar data and multispectral imagery. *Isprs Journal Of Photogrammetry And Remote Sensing, 2010, Vol.65(5), pp.457-467*, 65(5):457, 2010.

Mourad Bouziani, Kalifa GoÃ¯ta, and Dong-Chen He. Automatic change detection of buildings in urban environment from very high spatial resolution images using existing geodatabase and prior knowledge. *ISPRS Journal of Photogrammetry and Remote Sensing, 2010, Vol.65(1), pp.143-153*, 65(1):143, 2010.

Guo Cao, Xin Yang, and Zhihong Mao. A two-stage level set evolution scheme for man-made objects detection in aerial images. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 474–479. IEEE, 2005.

Q Chen, D Baldocchi, P Gong, and M Kelly. Isolating individual trees in a savanna woodland using small footprint lidar data. *Photogrammetric Engineering And Remote Sensing, 2006, Vol.72(8), pp.923-932*, 72(8):923, 2006.

Hans Czap and Wolfgang Nedobity. *Terminology and knowledge engineering*. Frankfurt/M. : Indeks-Verag., 1990.

Open Source Geospatial Foundation. Gdal/ogr info sheet, 2015. URL `http://www.osgeo.org/gdal_ogr`.

Garmin, 2011.

Valentin Haenel, Emmanuelle Gouillart, and Gaël Varoquaux. Python scientific lecture notes, release 2013.2 beta, 2013. URL `https://scipy-lectures.github.io/index.html`.

Renée Heilbronner and Stephen David Barrett. *Image analysis in earth sciences*. Springer, Heidelberg [u.a.], 2014.

Johannes Heinzel, Holger Weinacker, and Barbara Koch. Prior-knowledge-based single-tree extraction. *International Journal Of Remote Sensing, 2011, Vol.32(1), pp.69-84*, 32(17):4999, 2011.

MJ Huang, SW Shyue, LH Lee, and CC Kao. A knowledge-based approach to urban feature classification using aerial imagery with lidar data. *Photogrammetric Engineering And Remote Sensing, 2008, Vol.74(12), pp.1473-1485*, 74(2):1473, 2008.

Yan Huang, Bailang Yu, Jianhua Zhou, Chunlin Hu, Wenqi Tan, Zhiming Hu, and Jianping Wu. Toward automatic estimation of urban green volume using airborne lidar data and high resolution remote sensing images. *Frontiers of Earth Science, 2013, Vol.7(1), pp.43-54*, 7(1):43, 2013.

Ralph Jätzold. Der trierer raum und seine nachbargebiete. *Trierer Geographische Studien Sonderheft*, 6:360, 1984.

Brian Johnson and Zhixiao Xie. Classifying a high resolution image of an urban area using super-object information. *ISPRS Journal of Photogrammetry and Remote Sensing, 2013, Vol.83, pp.40-49*, 83:40, 2013.

K Khoshelham, C Nardinocchi, E Frontoni, A Mancini, and P Zingaretti. Performance evaluation of automated approaches to building detection in multi-source aerial data. *Isprs Journal Of Photogrammetry And Remote Sensing, 2010, Vol.65(1), pp.123-133*, 65(1):123, 2010.

Jungrack Kim and Jan-Peter Muller. Tree and building detection in dense urban environments using automated processing of ikonos image and lidar data. *International Journal of Remote Sensing, 2011, Vol.32(8), p.2245-2273*, 32(8):2245, 2011.

XL Meng, L Wang, and N Currit. Morphology-based building detection from airborne lidar data. *Photogrammetric Engineering And Remote Sensing, 2009, Vol.75(4), pp.437-442*, 75(4):437, 2009.

Jean-Matthieu Monnet, Eric Mermin, Jocelyn Chanussot, and FrÃ©dÃ©ric Berger. Tree top detection using local maxima filtering: a parameter sensitivity analysis, 2010.

Andrew Niccolai, Aaron Hohl, Melissa Niccolai, and Chadwick Dearing Oliver. Decision rule-based approach to automatic tree crown detection and size classification. *International Journal Of Remote Sensing, 2011, Vol.32(1), pp.69-84*, 31 (12):3089, 2010.

Weng Qihao. *Remote sensing and GIS integration.* New York: McGraw-Hill, New York, 2010.

Franz Rottensteiner, John Trinder, Simon Clode, and Kurt Kubik. Building detection by fusion of airborne laser scanner data and multi-spectral images: Performance evaluation and sensitivity analysis. *ISPRS Journal of Photogrammetry and Remote Sensing, 2007, Vol.62(2), pp.135-149*, 62(2):135, 2007.

Gunho Sohn and Ian Dowman. Data fusion of high-resolution satellite imagery and lidar data for automatic building extraction. *ISPRS Journal of Photogrammetry and Remote Sensing, 2007, Vol.62(1), pp.43-63*, 62(1):43, 2007.

Deutschland Statistisches Bundesamt Deutschland. *Statistisches Jahrbuch Deutschland und Internationales.* Statistisches Bundesamt, Wiesbaden, 2014.

Jan Tigges, Tobia Lakes, and Patrick Hostert. Urban vegetation classification: Benefits of multitemporal rapideye satellite data. *Remote Sensing of Environment, 2013, Vol.136, pp.66-75*, 136:66, 2013.

Pooman Tiwari and Hina Pande. Lidar remote sensing applications in automated urban feature extraction, laser scanning, theory and applications„ 2011. URL `http://www.intechopen.com/books/laser-scanning-theory-and-applications/lidarremote-sensing-applications-in-automated-urban-feature-extraction`.

Stadt Trier. Bevölkerungsstruktur der stadt trier, 2015a. URL `http://www.trier.de/Rathaus-Buerger-in/Trier-in-Zahlen/Bevoelkerungsstruktur/`.

Stadt Trier. Soziale stadt, 2015b. URL `http://www.trier.de/Bauen-Wohnen/Stadtplanung/Soziale-Stadt/`.

Quan Wang, Samuel Adiku, John Tenhunen, and André Granier. On the relationship of ndvi with leaf area index in a deciduous forest site. *Remote sensing of environment*, 94(2):244–255, 2005.

Qihao Weng. *Urban remote sensing.* CRC Press, 2007.

# Declaration of authorship

With this statement I, Andreas Löwe, matriculation number 1002405, declare, that I have independently completed the above master thesis entitled with:

> *Extraction of urban features with knowledge based engineering and local maxima filtering Demonstrated for the city of Trier*

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.

Trier, January 28, 2016

Andreas Löwe

# A. Appendix

| District | True Positive | False Positive | False Negative | True Validation | Result Build up |
|---|---|---|---|---|---|
| Altstadt | 601554,8 | 112099,0 | 37884,3 | 646044,0 | 684374,0 |
| Feyen | 111889,9 | 31350,8 | 22407,8 | 134297,7 | 143240,8 |
| Gartenfeld | 97287,7 | 17568,4 | 17196,6 | 118387,7 | 114175,8 |
| Maximin | 339044,5 | 66888,5 | 54897,7 | 399279,5 | 396756,3 |
| Pallien | 78964,0 | 28792,9 | 14865,2 | 93829,2 | 97363,8 |
| Trier West | 278145,0 | 36752,2 | 33468,7 | 311613,7 | 310682,5 |
| Sum | 1506885,9 | 293451,9 | 180720,3 | 1703451,8 | 1746593,0 |

Evaluation of the results for the six districts, units are square metre

| District | True Positive | False Positive | False Negative |
|---|---|---|---|
| Altstadt | 93,1 | 19,3 | 5,9 |
| Feyen | 83,3 | 23,3 | 16,7 |
| Gartenfeld | 82,3 | 16,3 | 17,7 |
| Maximin | 84,9 | 17,9 | 15,1 |
| Pallien | 84,2 | 30,7 | 15,8 |
| Trier West | 89,3 | 11,8 | 10,7 |
| Sum | 88,5 | 18,3 | 11,1 |

Evaluation result of the results for the six districts in Percent