

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

*Distributed
Computing*



Tesselating Switzerland of the Past

Bachelor Thesis

Sebastian Keller

`sekeller@student.ethz.ch`

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:

Manuel Eichelberger
Prof. Dr. Roger Wattenhofer

February 28, 2018

Acknowledgements

I thank my supervisor Manuel Eichelberger for the assistance during the project by giving helpful hints, asking critical questions and reading this paper. I further thank Prof. Dr. Wattenhofer for his encouraging comment, that there must exist a simple solution to this complex task that has just not been found yet. I also thank my family and all my friends for their patience with me, talking to them about the ongoing work.

Abstract

This project continues to improve an existing framework for positioning historical aerial images on the current map. The framework consists of three main parts, an image fetching step, an image matching step and an exporting step, that converts the processed aerial images in to the needed format to view them on the included interactive web map.

The main focus was laid on improving the image matching step to get more accurate results, especially at the borders of the images. A slight improvement was achieved by respecting the geometrical constraints, such as angles and distances, between points. Another approach was considered, in which the matching quality gets improved by displacing, rotating and scaling small image parts until they fit best.

The framework was entirely written in Python, but to improve the performance of the matching step, the software was rewritten in C++. From the rewriting results a speed-up up to 100 % for the matching step. Considering that the rewritten matching step uses multiple feature matcher, instead of just one, the speed-up is not negligible. The overall runtime to process one image is however about four times slower since an additional part which applies the matching step recursively to subpatches of the image was introduced.

Moreover the software works with OpenData images which have no copyright. Therefore, the resulting images shown on the interactive map can be made publicly accessible.

Contents

Acknowledgements	i
Abstract	ii
1 Introduction	1
1.1 Previous Work	1
1.2 Goals	1
1.3 Related Work	2
2 Method	3
2.1 Existing Method	3
2.2 Improving Feature Based Image Matching	4
2.2.1 Using Multiple Detectors	5
2.2.2 Image Pyramids	6
2.2.3 Filtering Matches	8
2.2.4 Geometrical Path Construction	9
2.3 Recursive Search for Matches	15
2.4 Pixel Data Based Matching	16
3 Implementation	18
3.1 Programming Language and Libraries	18
3.1.1 Multithreading	18
3.1.2 WMTS Layer Creation	19
3.2 Using OpenCV	19
3.2.1 Feature Based Image Matching	19
3.2.2 Estimating Homographies	19
3.3 Program Components	21
3.4 Implementing Pixel Data Based Matching	21

4	Results	25
4.1	Feature Image Matching	25
4.1.1	Matching Quality	25
4.1.2	Older Aerial Images	26
4.1.3	Performance Comparison	28
4.2	Pixel Data Based Matching	29
4.2.1	Optically vs. Mathematically Appealing	29
5	Conclusion	35
5.1	Outlook	35
5.1.1	Further Improving Pixel Data Based Matching	35
5.1.2	Image Matching Using Multiple Data Sources	35
5.1.3	Accelerating WTMS Tile Generation	36
	Bibliography	37
A	Appendix	A-1
A.1	Using <i>aerialImageMatcher</i>	A-1
A.1.1	Installing Prerequisites	A-1
A.1.2	Building	A-1
A.1.3	Image Downloading Scripts	A-2
A.1.4	Usage	A-2

Introduction

Today aerial and satellite images are used in most online map services but their goal is to provide the most current data only. To review the development of an area over time, historical aerial image data has to be taken into account. There is a large amount of aerial images provided by Open Government Data Zürich [1] and the Swiss Federal Office of Topography (swisstopo) [2] dating back to the year 1926. However, there is no easy way to look at larger areas or to compare images captured at different dates, as the images are only provided individually.

This project tries to ease the process of looking at different dated aerial images and compare them to any other given point in time by providing a web based map viewer similar to other map services.

1.1 Previous Work

This project is the third thesis to deal with the subject of temporal maps. The series started with the semester thesis of Florian Zinggeler [3], in which the basic framework with the image matching back end and a web based front end was developed. Markus Roth continued the series with his bachelor thesis [4], where he optimized several aspects of the matching process, placing the focus on storage space and computing time reduction as well as increasing matching accuracy.

1.2 Goals

Despite the optimisations worked out by Markus Roth, the matched images still lack precision at the border of the processed image, such that there are clearly visible distortions like roads that are not aligned. On the one hand, we try to optimize the feature matching approach and on the other hand introduce a second matching stage to calculate a best fit for small tiles of the matched image to further improve accuracy. To speed-up the matching process even more, we decided to reimplement the software in C++ instead of Python. The given web front end is reused with some small changes.

1.3 Related Work

Feature based image matching is researched since the early 1980's. Therefore many different strategies to find good features have been developed. A detailed overview and comparison between the different strategies is described by Usman Muhammad Babri et al. [5]. They compared *Speeded Up Robust Features* (SURF) [6], *Features from Accelerated Segment Test* (FAST) [7] and the *Scale Invariant Features Transform* (SIFT) [8]. Although SURF and FAST are faster than SIFT, they lose precision in the matching. Therefore, SIFT was the choice for this work.

The FLANN-based matcher developed by Muja and Lowe [9, 10], provides fast matching of features between images, without compromising the result quality compared to linear matching, which is the textbook solution to find the best match by comparing every found feature to each other.

Yingjie Xia et al. [11] propose an algorithm to find discriminative structures in aerial images by creating region connected graphs (RCG) of the aerial image. This approach is most useful in urban regions, since it requires clearly defined segmentation in the image to find the borders of the extracted regions. They tested it on two datasets but included only the results for one of them, which is indeed completely within urban areas. The idea to search for geometrical structures in images has influenced the development of the geometrical path construction algorithm described in Section 2.2.4.

With regard to aerial images there exist approaches that combine different data sets to get a good georeferencing quality or content discretisation. The used additional sources are digital surface/terrain models ([12, 13, 14]), LIDAR imagery ([14, 15]) and four channel images with near infrared spectral data ([13]). Another technique used by Karel et al. [12] is to combine the gathered information of multiple aerial images and store them into a database, against which each new referenced image can be compared.

Method

In this chapter we present the existing method and then show our changes and improvements.

The term *aerial image* is used in the following chapters to refer to an image captured by plane somewhen in the past, where as the term *base image* refers to the most recent satellite image of the region the aerial image was taken in.

2.1 Existing Method

The existing method from previous work uses a collection of Python scripts. This collection contains scripts to download the aerial images from swisstopo [2] and the base image from the *Web Map Service*¹ server of the canton of Zurich [17]. Furthermore, there are scripts to automatically determine the geographic coordinates of the aerial images with respect to the base image's coordinate reference system, in brief called *georeferencing* the aerial image, and to add the resulting image to an interactive web map.

To georeference the aerial image, the provided *bounding box* from swisstopo, which is defined by the top-left and the bottom-right geographic coordinates, is used as an initial placement of the aerial image on the base image. Then feature based image matching is applied in multiple steps to further refine the positioning of the aerial image. A more detailed overview can be found in Figure 2.1.

The feature based image matching part uses low resolution versions of the aerial image and the base image to only find features that are clearly visible in both images. This approach was chosen since it diminishes the influence of fine-grained structures, such as street crossings or trees, on feature image matching. For these fine-grained structures exist too many possible candidates in an image, which can lead to imprecise matches being reported by the feature detector. To further improve the matching, a new bounding box based on the found features is calculated. With the more precise base image, a second feature matching step is

¹A Web Map Service (WMS) is a standard specified by the Open Geospatial Consortium for serving georeferenced map images over the Internet [16].

performed, in which the image is divided into patches and the process described before, is repeated for the whole image with every single patch. Then the matches of those patches are combined to calculate the final, relatively precise bounding box.

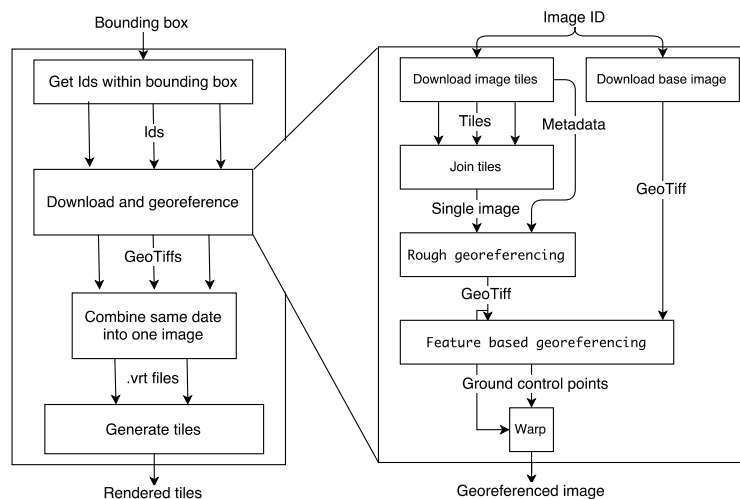


Figure 2.1: Overview of the existing method by Florian Zinggeler. Markus Roth optimized the process in terms of storage space requirements and execution speed but did not change the sequencing of the method. (Image from [3])

2.2 Improving Feature Based Image Matching

The new method presented in this work uses a *SIFT keypoint detector*² to extract features from the images. The keypoints and descriptors returned by the SIFT keypoint detector must be further processed with a feature matcher, to know which features in the first image correspond to the ones in the second. The resulting matches should be filtered to obtain a set of good matches. An overview over the filtering techniques applied in this method can be found in Figure 2.2. After filtering, the good matches can be used to calculate a perspective transform between the two images.

In computer vision this perspective transform is called a *homography* and is mathematically described by a 3x3 matrix. This matrix is used to map the homogeneous coordinates of every point from the source to the destination plane (with a possibly different coordinate system). An in-depth explanation can be found in [18].

²A detector that uses the Scale-Invariant Feature Transform algorithm described by Lowe [8] to find keypoints in the given images

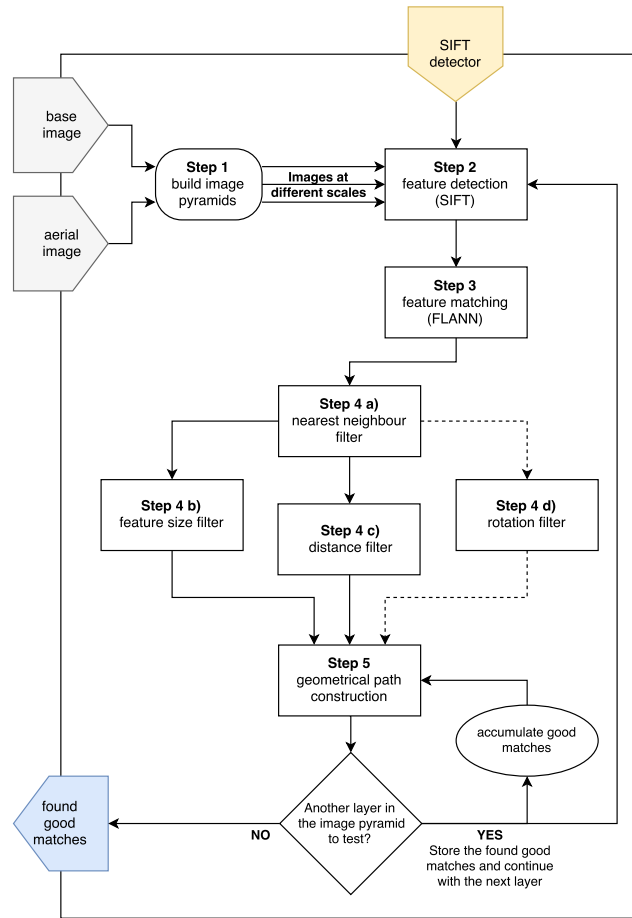


Figure 2.2: Overview over all the steps involved in the feature based image matching. These steps are performed for each of the differently configured SIFT keypoint detectors.

2.2.1 Using Multiple Detectors

A feature detector computes an abstraction of the provided image to find locations of special interest within the image. Such locations of special interest can be corners or edges of objects or other structures that can easily be recognised, like a single tree in a grass field.

To improve the number of found features, we decided to use multiple SIFT keypoint detectors (Figure 2.2 Step 2). Each of them is configured in a different manner. The variable parameters are the number of octave layers created by SIFT keypoint detector, the contrast threshold, the edge detection threshold and the parameter σ_{start} of the *Difference of Gaussian* (DOG) method used by the SIFT keypoint detector.

The number of octave layers determine how many times the input image gets downsampled by a factor of two to detect local extrema in the image. Each octave consists of multiple DOG image layers, which are generated by taking the difference of two adjacent Gaussian blur smoothed input images at the octaves current scale (see Figure 2.3). Each image in the set of blurred images is calculated using a different value for the standard deviation $\sigma_i (= \sigma_{start} + \delta_i)$ of the Gaussian blur function, which corresponds to varying the intensity of the blur.

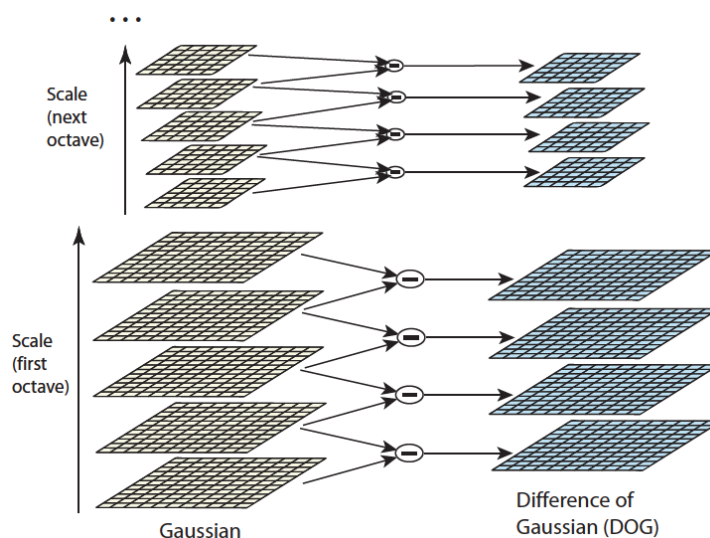


Figure 2.3: The creation of octave layers in the SIFT keypoint detector. On the right are the sets of blurred Gaussian images, each blurred with a different value of σ , and on the left the resulting *Difference of Gaussian* layers. (Image from [8])

To derive good keypoints from the detected extrema, the keypoints having a too low contrast, namely below the given contrast threshold parameter, are rejected, because these keypoints are too sensitive to noise in the image. The edge threshold is used to reject false-positive extrema that lie on edges in the image. This filtering needs to be applied because the DOG method gives large values around edges and therefore reports extrema along edges, despite it might be no extremum at all or the exact position of the extremum along the edge can not be clearly determined because it is biased by the high value of the DOG around the edgy structure.

2.2.2 Image Pyramids

The provided aerial images have a high resolution, of up to 20.000 by 20.000 pixels, so we need to scale them down to get matches in an acceptable amount

of time. As in the previous work to improve the number of matches, we build image pyramids (see Figure 2.4) of both images and then match each layer of one image pyramid against the corresponding layer of the other image pyramid (Figure 2.2 Step 1). This way, we can detect features that are only detectable at a certain resolution. Despite the fact that the SIFT keypoint detector already creates image pyramids by its own to extract feature descriptors, the use of image pyramids increases the number of matches found (see Figure 2.5).

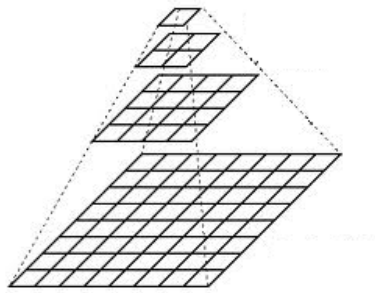


Figure 2.4: Image pyramids are generated by repeatedly sampling down the bottom layer by a constant factor to get the next higher layer. A factor of $\sqrt{2}$ is used in this method. (Image from [19])

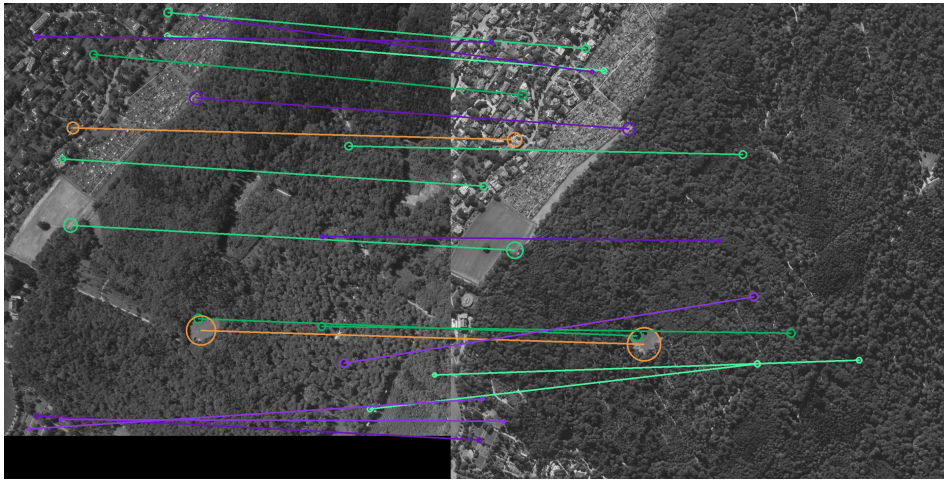


Figure 2.5: The matches are drawn in a color depending on the layer of the image pyramid. (Left image from [1], right image from [17])

2.2.3 Filtering Matches

A matcher decides which keypoint in the first image belongs to another keypoint in the second image by comparing the extracted image feature descriptors.

To match the keypoints we use a *FLANN-based matcher*³ (Figure 2.2 Step 3). This matcher was chosen since it is faster than a simple linear search for the best match. The FLANN-based matcher can also be configured to internally use different indexing approaches. The used index influences the speed and the precision of the matches. Since we find complex image structures in aerial image and aim to have precise matches, the choice was made for the k-means index [10]. The use of this matcher allows us to speed-up the matching process but still get good precision compared to the *brute force matcher*⁴ used in previous work.

Nearest Neighbours The matcher provides us with an interface to extract the two nearest neighbours of any found feature. This allows us to test the best match of every found feature in the set of matched features \mathcal{M} against the second best match (Figure 2.2 Step 4 a)).

When the best two matches have about the same distance according to the distance function used by the matcher, both of them are either equally likely to be the correct match or both of them are presumably a wrong match. But when the best match has a far lower distance than the second best match, we know that there is no better match for this feature. Therefore we only add matches to the set of accepted matches \mathcal{M}_{dist} when the distance of the best match is more than a factor l lower than the second best match.

$$\mathcal{M}_{dist} = \{x, x \in \mathcal{M} \wedge \left(\frac{distance(x.snd)}{distance(x.best)} \right) - 1 < l\}$$

The factor l is chosen to initially be 0.5, but if more than two thirds of all matches are discarded, it will be reduced in small steps to at least 0.1. This way, we always get the best possible matched features relative to all other found matches but still discard features of the aerial image that cannot clearly be assigned to a *single* feature in the base image.

To further refine the matches, additional filtering steps are performed, which are explained next.

Feature Size The filtering based on the feature size (Figure 2.2 Step 4 b)) compares the size of the found feature located at a keypoint in the aerial image

³A matcher that uses a fast approximation for the nearest neighbour problem as described by Muja and Lowe [9]

⁴A matcher implementing a linear search approach by comparing every keypoint to all others to find the two best matching keypoints.

to the size of the feature in the base image. A match in the set \mathcal{M}_{dist} only gets added to the set \mathcal{M}_{pt} if the size difference is below a chosen threshold p .

$$\mathcal{M}_{pt} = \{x, x \in \mathcal{M}_{dist} \wedge \frac{|size_{aerial}(x) - size_{base}(x)|}{max(size_{aerial}(x), size_{base}(x))} - 1 < p\}$$

$|a|$ denotes the absolute value of a and $max(a, b)$ returns the bigger value of a and b .

The threshold p is also varied between 0.1 and 0.35 to prevent rejecting all matches.

Distance After applying the nearest neighbour filter, the keypoints may all have a good ratio to their second best match but compared to other points in the set they may have a large distance. To reject keypoints that match worse relative to other keypoints in the set, which is indicated by a large distance relative to the minimum distance. Therefore, the minimum distance $dist_{min} := \min_{x \in \mathcal{M}_{dist}}(distance(x))$ is identified and then all keypoints with a distance greater than $d \cdot dist_{min}$ are rejected. (Figure 2.2 Step 4 c)

$$\mathcal{M}_{min} = \{x, x \in \mathcal{M}_{dist} \wedge \left(\frac{distance(x)}{dist_{min}} \right) < d\}$$

The factor d is also chosen in a range to still get more than just the minimal point, when all but one point have a distance $\gg dist_{min}$.

Rotation After finishing the first step of calculating a more precise bounding box for the aerial image, the rotation of the aerial image compared to the base image is approximately known. Therefore, in any further processing we assume that the found matches also must not be significantly rotated. So from this step on another filter is added (Figure 2.2 Step 4 d) , that ensures that the matched features are located in about the same region in the image. To achieve this, the location of the keypoints is normalized to the range $[0, 1]^2$, with respect to the dimension of the aerial image or the base image, respectively. The distance between the two keypoints in this normalized space must be below a threshold to be added to the set \mathcal{M}_{rot} .

A keypoint is considered a good match if it is at least in two of the three sets \mathcal{M}_{pt} , \mathcal{M}_{min} and \mathcal{M}_{rot} .

2.2.4 Geometrical Path Construction

To get more stable homographies, all matching features found are tested with respect to whether the keypoints form the same geometrical shape in the aerial image as in the base image.

Algorithm 2.1 Calculate the Homography error and the coverage of the inliers

Input: \mathcal{P} := set of points in path, H := homography matrix, $area_{image}$, $displacementThreshold$

Output: $error_{proj}$, $coverage$, \mathcal{I} := inliers, \mathcal{O} := outliers

```

1:  $error_{proj} \leftarrow +\infty$ 
2:  $\mathcal{I} \leftarrow \{\}$ 
3:  $\mathcal{O} \leftarrow \{\}$ 
4: if  $|\mathcal{P}| \geq 4$  then
5:    $\mathcal{E} \leftarrow \{\}$  // Array of all errors
6:   for all  $p_i$  in  $\mathcal{P}$  do
7:      $a \leftarrow$  Homogeneous coordinates of the point  $p_i$  in the aerial image
8:      $b \leftarrow$  coordinates of the point  $p_i$  in the base image
9:      $h \leftarrow H \cdot a$ 
10:     $h' \leftarrow h$  converted back to Euclidian coordinates
11:     $err \leftarrow \|b - h'\|$  // Euclidian distance between  $b$  and  $h'$ 
12:    if  $err \leq displacementThreshold$  then
13:       $\mathcal{I} \leftarrow \mathcal{I} \cup p_i$ 
14:       $\mathcal{E} \leftarrow \mathcal{E} \cup err$ 
15:    else
16:       $\mathcal{O} \leftarrow \mathcal{O} \cup p_i$ 
17:    end if
18:  end for
19:   $numInliers \leftarrow |\mathcal{I}|$ 
20:  if  $numInliers \geq 4$  then
21:     $errMed \leftarrow$  median of  $\mathcal{E}$ 
22:     $errAvg \leftarrow$  average of  $\mathcal{E}$ 
23:     $error_{proj} \leftarrow (errMed + errAvg) / 2$ 
24:     $area_{\mathcal{I}} \leftarrow$  area of the convex hull of  $\mathcal{I}$ 
25:     $coverage \leftarrow area_{\mathcal{I}} / area_{image}$ 
26:  end if
27: end if

```

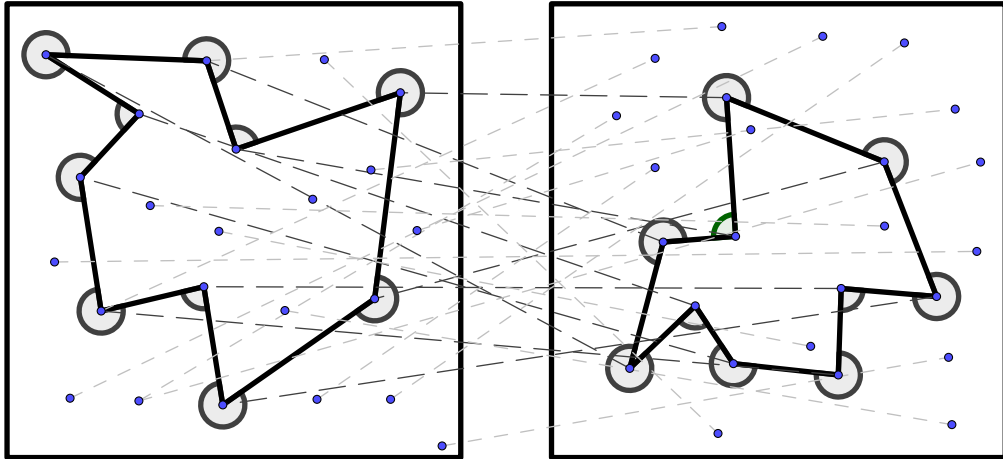


Figure 2.6: On the left side are the image features found (dots) in the aerial image, on the right side the ones found in the base image. In both point sets a path is found in order that the right one is a scaled and rotated version of the left one. The dashed lines show which image features were matched.

To test for the same geometrical shape, a path is constructed through each of the two point sets, one belonging to the feature locations in the aerial image and the other to the feature locations in the base image, so that all triplets of points approximately have the same angles in between on both paths and that the ratio of the line lengths between all lines in the two paths is approximately constant up to a scaling factor (see Figure 2.6).

From the good matching keypoints obtained from the previous filtering (Figure 2.2 Step 4) a *min-heap* is constructed. A min-heap is a data structure that allows insertion and deletion of items to be performed in logarithmic time and always keeps the minimal element at the root. To build the heap, a partial order on the elements contained needs to be specified. The order is defined by adding state information to each matched keypoint about the number of times it was included or discarded in a path candidate. If a matched keypoint was included less than three times in a path and discarded less than a hundred times, the order value is set to the number of times it was included. Otherwise, the order value is the sum of the number of times it was included and discarded. This way, it is ensured that matched keypoints with a low included count get tested more often and matched keypoints that were discarded many times get tested less frequently, as they have a longer distance until they moved to the root of the heap.

The values 3 and 100 were chosen arbitrarily, but there is a reasoning behind: if a matched keypoint is rejected more than a hundred times, the probability that

its contribution to a new path candidate will lead to a usable path is converging to zero. Also if a matched keypoint which is already used in more than two candidate paths is added to a new one, the new path candidate will most likely resemble the two previous ones. By using the previously sketched partial ordering and by reinserting the points to the heap as long as their order value is less than 200, we have assured that every point gets tested at least once.

The procedure starts searching for path candidates by removing the minimal element of the heap and choosing two other randomly selected matching keypoints. For this start triple the angle between the three points and the ratio between the two line segments in both images are compared. If they have about the same value, the search is continued among the remaining matched keypoints. Each of these matching keypoints is tested on fulfilling those conditions and therefore can be appended to the current path candidate. No matter if a path candidate was found or not, the matching keypoint is then added back to the heap with an updated included and discarded count.

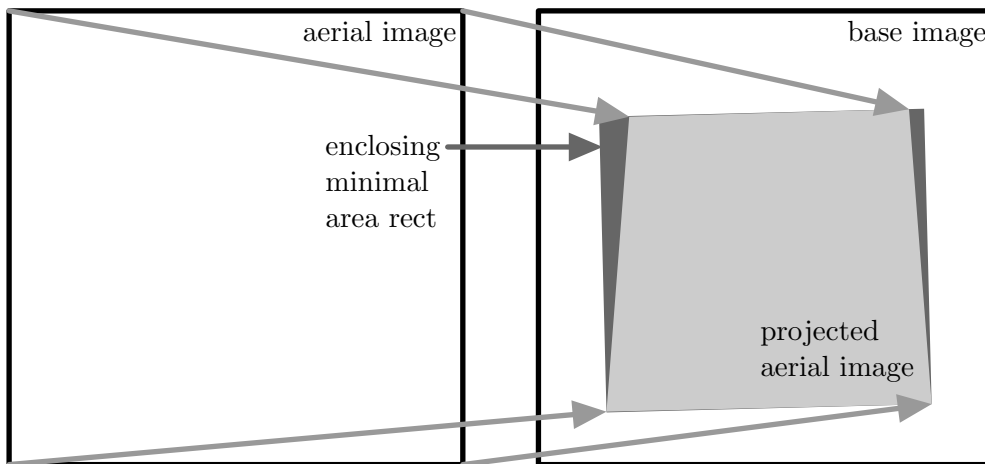


Figure 2.7: The difference quotient of the area of the (dark grey) enclosing minimal area rectangle divided by the area of the (light grey) projected aerial image is defined as the area error.

Using this technique we might get different paths, therefore to each of the found paths an error value (e_{path}) is assigned. The total error value is the weighted sum of the average error of the angle and line ratio differences (e_{geom}) along with the projection error (e_{proj}) (calculated as shown in Algorithm 2.1) of the resulting homography using only the points in the path candidate. Also

the difference quotient of the areas (e_{area}) of the projected image quadrilateral⁵ and the minimal enclosing rectangle of the before mentioned quadrilateral (see Figure 2.8) is added to the total error value.

To calculate a homography between two bases, at least 4 points are needed. If the calculated projection error of a path contains 4 points, its error value is going to be zero, since every point will match exactly the projected one. A homography calculated from a path with more than 4 points is an approximation. In Homographies estimated from many points, every point contributes with a small error but a better projection results than using a homography that is estimated using just 4 points.

To take this into consideration also the *coverage* factor (b_{cov}) is calculated in Algorithm 2.1 alongside the projection error. This factor is the ratio of the matching image's area and the area that is covered by the convex hull of the points used to calculate the homography. A higher coverage means a better distribution of the points in the image plane, which leads to lower aberrations at the borders of the image. When the different found path are compared, the coverage factor serves as a bonus in the ranking.

$$\mathbf{W} = \begin{bmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 4 & 2 & 2 & 2 & 2 & 2 & 2 & 4 \\ 4 & 2 & 1 & 1 & 1 & 1 & 2 & 4 \\ 4 & 2 & 1 & 0 & 0 & 1 & 2 & 4 \\ 4 & 2 & 1 & 0 & 0 & 1 & 2 & 4 \\ 4 & 2 & 1 & 1 & 1 & 1 & 2 & 4 \\ 4 & 2 & 2 & 2 & 2 & 2 & 2 & 4 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{bmatrix}$$

Besides the coverage, we also want to know about the distribution of the points inside the image since two different distributions of points can lead the same coverage factor. Therefore the image gets divided in 64 sectors and each sector is marked if a matching point lies inside. Then the associated weight of each marked sector is summed up and divided by the value of a perfect distribution to get a second bonus value (b_{distr}). The weights, shown in the matrix \mathbf{W} , are chosen in such a way, that points further away from the center lead a higher contribution in the bonus.

⁵A four-sided figure

The total error of a path is then calculated as following:

$$e_{path} = \frac{\left(1 - (1 - e_{area})^2\right) + 4 \left(\sqrt{4 \cdot (1 + e_{geom})} - 2\right) + e'_{proj}}{3} - 3(b_{cov} + b_{distr}) \left(\sqrt{1 + e''_{proj}} - 1\right)$$

where $d := displacementThreshold$,

$$e'_{proj} := \begin{cases} e_{proj} & e_{proj} < d \\ \sqrt{\frac{e_{proj}^3}{3}} & d \leq e_{proj} \end{cases} \quad \text{and} \quad e''_{proj} := \begin{cases} e_{proj} & e_{proj} < d \\ d & d \leq e_{proj} \end{cases}$$

The functions applied to each error component influences its contribution to the total error in a characteristic way. The area error is bounded within the interval $[0, 1]$ and after applying the function, it grows towards 1 quicker than linear (see Figure 2.8). The geometrical error grows slower than linear after applying the function, but the value is multiplied by a factor of 4 to keep the influence on the total error (see Figure 2.9). The projection error is treated in two different ways. Values below the *displacementThreshold* grow linear but all values greater than the *displacementThreshold* are made to grow faster than linear that to too high errors get sorted out quickly (see Figure 2.10).

A projection defined by only 4 points gets no bonus at all. The values of e''_{proj} are bounded above by the *displacementThreshold* (see Figure 2.11), such that the size of the bonus becomes more and more irrelevant as e_{proj} and therefore also e'_{proj} grows. The bonus values are both bounded within the interval $[0, 1]$ and therefore, the maximal bonus given is $6 \left(\sqrt{1 + displacementThreshold} - 1\right)$.

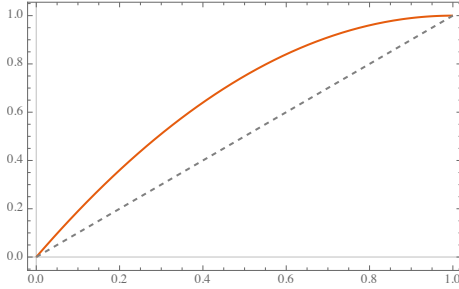


Figure 2.8: The red plot shows the function $y := \left(1 - (1 - x)^2\right)$ applied to the area error term e_{area} . The dashed plot shows the linear contribution as a comparison.

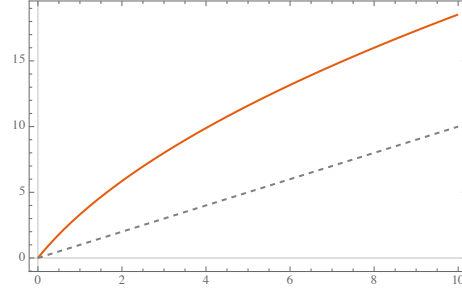


Figure 2.9: The red plot shows the function $4 \left(\sqrt{4 \cdot (1 + x)} - 2\right)$ applied to the geometrical error term e_{geom} . The dashed plot shows the linear contribution as a comparison.

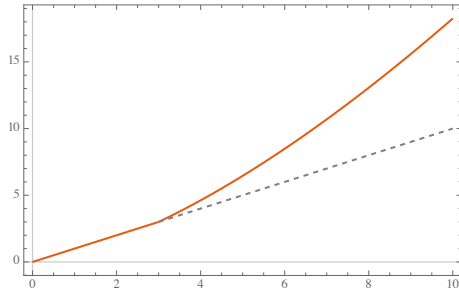


Figure 2.10: The partially defined function applied to projection error e'_{proj} plotted in red shows the penalty for values greater than the *displacementThreshold* which is fixed at a value of 3 in the plot.

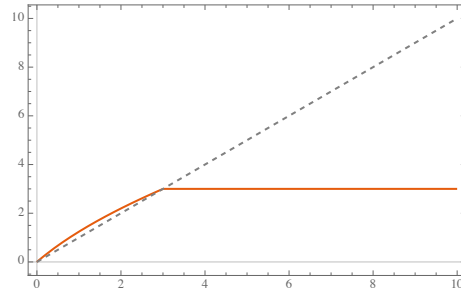


Figure 2.11: The red plot shows the function $y := (\sqrt{1+x} - 1)$ applied to e''_{proj} , which is clipped to the *displacementThreshold*. The *displacementThreshold* is fixed at a value of 3 in the plot.

2.3 Recursive Search for Matches

After finishing the initial georeferencing step, the rotation of the aerial image compared to the base image is known and the images are more or less aligned. Therefore, we can use this information and restrict the search space for the paths in the image to those which have about the same orientation in both images. Additionally the thresholding distance *displacementThreshold* used to differentiate inliers from outliers is reduced (see in Algorithm 2.1).

Then, the matching process is continued with the goal to increase the total number of matched points. This is done by splitting up the image into 4 patches. In each of these patches the procedure described in Section 2.2 is reapplied.

Since it is now searched for matches only in a certain part of the image, chances increase for some matches to survive the filtering step described in Section 2.2.3 although they were discarded before. Either they were rejected because the nearest neighbour was reported in another part of the image and the match would therefore not meet the requirements to be added to a path or there were more similar features in other parts of the image and therefore the nearest neighbour filtering would have rejected the point.

If the number and quality of the matches is above some predefined threshold or the patch size gets to small, the recursion is stopped. Otherwise, the patch is again split in 4 smaller subpatches and the matching continues with these subpatches.

When the recursion terminates, all found points are combined and used to estimate the final perspective transform between the aerial image and the base image.

2.4 Pixel Data Based Matching

To augment the matching quality further, especially around the borders of the image, an approach to find the best positioning of the image by using a template matching based approach working deterministically on the images pixel data has been considered.

The standard template matching approach only searches for the optimal 2D displacement of the *template* (in our case the aerial image patch) against the *reference* image (in our case the base image) inside a rectangular window [20]. In addition to that we also want to evaluate the best scaling and rotation of a patch. We achieve this by iteratively applying the template matching to scaled and rotated versions of the template. Since we already have an approximation of the scale and rotation of the whole aerial image after georeferencing, only small scaling and rotation factors are considered. In concrete terms a scaling between 90% and 110% of the original patch size and a maximal rotation of ± 2 degrees are tested.

To reduce the influence of noise in the image and to minimize the effects of different lightning conditions between the two images, an edge image representation of the aerial image and the base image are generated. Then the original and edge images are used simultaneously in the matching process. Both images are used to overcome the problem that template matching will give bad results when the template or the reference image have too less diversity in the image data.

The used template matching approach provides several different methods to calculate a weighted sum over all pixel of the template image but only two of them support an image mask. We need this functionality since we need to mask out the black borders around the image which are unavoidable when rotating an image as shown in Figure 2.12.

The two usable methods are the *Normalized Sum of Squares Difference* (NSSD) method and the *Normalized Cross Correlation* (NCC) method. The NSSD method is computationally simpler than NCC since it only involves addition and multiplication operations, but the NCC method is more robust to lightning changes between the template and the reference image. We will therefore use the NCC method to match the single patches.

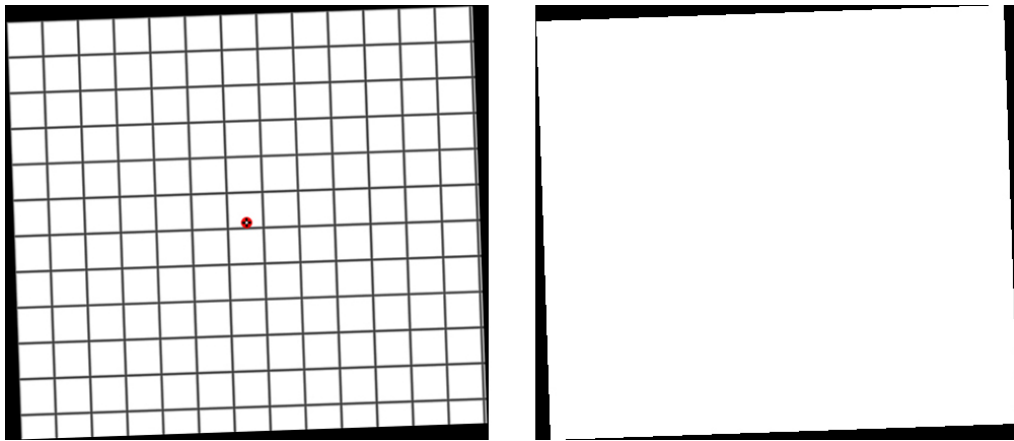


Figure 2.12: On the left side of the Figure is a test image rotated about 2 degrees anticlockwise. The black border around the rotated image results from crop free the rotation applied to the image. On the right side the corresponding mask used to only match inside the white area is shown.

Implementation

3.1 Programming Language and Libraries

The tool is written in C++ using the C++14¹ version of the language. To read and manipulate images with associated geospatial information, like the GeoTIFF file format, or to create virtual geospatial information files referencing images that cannot have associated data, GDAL, the Geospatial Data Abstraction Library [21] is used. The OpenCV library [22] and its contribution libraries [23] are used to do image manipulation tasks, the feature based image matching and template matching. For auxiliary tasks, such as parsing program arguments, reading and writing JSON and XML files, parts of the boost library [24] are used.

3.1.1 Multithreading

C++ provides an interface to create threads via *std::thread*. In a first approach when entering a section that can be parallelized, a bunch of *std::threads* were created and then released right after leaving said section. Using this approach up to over 1,000 threads were created and destroyed for matching a single image. This introduces an unnecessary overhead for creating and destroying the threads. To fix the number of created threads a thread pool is used. The thread pool creates a bunch of worker threads, to which workloads can be submitted, but which are not destroyed after finishing a single workload. Since the C++ standard library does not provide an implementation of a thread pool, an open source header-only thread pool implementation from Progsch and Zeman [25] is used with small modifications to support prioritized tasks.

¹ISO/IEC 14882:2014 <https://www.iso.org/standard/64029.html>, <https://en.wikipedia.org/w/index.php?title=C++14&oldid=822817064>

3.1.2 WMTS Layer Creation

To display an image on the web interface, the image has to be split up according to the Web Map Tile Service (WMTS) standard [26]. The Open Source Geospatial Foundation provides a Python script (*gdal2tiles.py*²), which is directly called from within C++ via the Python C API [27].

3.2 Using OpenCV

3.2.1 Feature Based Image Matching

When using any feature based image matching in OpenCV, the library exposes the *Keypoint* class to deliver information about a matched point. This class has an attribute *angle* which is the *computed orientation of the keypoint* according to the documentation³.

Given this information, we tried to estimate the orientation of the aerial image compared to the base image. The difference of the angles between two matching keypoints was first stored in a list and naively sorted by angle. Since a histogram produced from the sorted data did not reveal any clear peak, a 1D k-means clustering algorithm was applied to the gathered data. Neither using random initial centers nor setting the initial centers based on the histogram helped finding a converging solution. Therefore the idea to determine the rotation of the aerial image directly from the keypoint data was dropped.

3.2.2 Estimating Homographies

To estimate the projection between two images, the OpenCV framework provides three different methods which all can be invoked over the single interface *cv::findHomography*. The regular method estimates a homography by solving a system of linear equations using all provided points. The second method uses a least median square (LMEDS) approximation. The third method uses the random sample consensus approach (RANSAC) [28]. Both methods start with a subset of 4 points, estimate a first homography and then add more points that fit the estimation, either by comparing the median of the projection error (LMEDS) or by thresholding on the distance between the points real position and the estimated position (RANSAC). The latter two methods are called robust methods, because they can handle a certain number of outliers.

²<https://github.com/OSGeo/gdal/blob/123adbbe121c175d1721b531cfe48f3b4ad5b6b2/gdal/swig/python/scripts/gdal2tiles.py>

³https://docs.opencv.org/3.3.1/d2/d29/classcv_1_1KeyPoint.html#a4484e94502486930e94e7391adf9d215

Each method has its benefits and drawbacks. The regular method should only be used when there are no outliers in the provided point set, but it returns a homography that approximates all given points and so respects the distribution of the points on the plane. LMEDS works best if not more than 50% of the points in the set are outliers. RANSAC works with any ratio of outliers but is conservative with the points it adds to the set of inliers, which can be seen in Figure 3.1. Many homographies calculated by the RANSAC method are only determined by the minimum of 4 points required to calculate a homography.

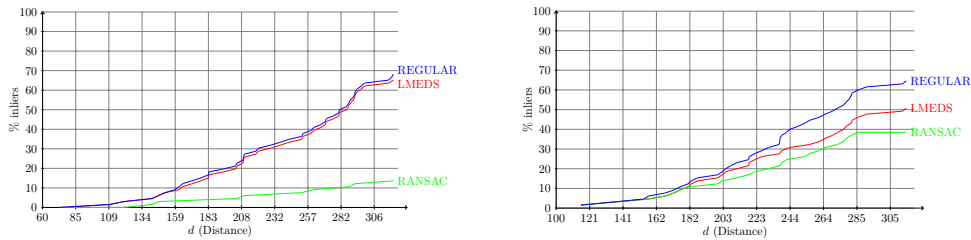


Figure 3.1: The fraction of inliers (points used in homography vs. points selected in the geometrical path construction) compared to the distance returned from the image feature matching phase. Slightly outstanding example on the left where RANSAC considers a *lot* less points as inliers. The image on the right shows a more common case.

The points in a constructed path (see Section 2.2.4) are considered to contain only a small number of outliers. The homography is estimated initially using the regular method. After calculating the projection error (see Algorithm 2.1), it is compared to the projection errors using LMEDS and finally RANSAC method. The homography with the least projection error and the best distribution will then be selected. Thereafter, we reestimate the homography using only the inliers but this time using the regular method, which yields a better estimation if more than 4 points are considered inliers.

Under bad circumstances even recalculating the homography will not improve the quality of the estimated transform. Such a situation can be seen in Figure 3.3, where all the found matches lie nearly on a line or clustered to near together. Therefore, the distribution bonus is taken into account in the path finding algorithm described in Section 2.2.4.

The OpenCV library does not provide an direct way to get the projection error of a homography estimation, wherefore Algorithm 2.1 was implemented.

3.3 Program Components

The structure of the program, its required input data and the generated output data are shown in Figure 3.4. The *border cropping*, *georeferencing* and *WMTS tiles* phases are executed after each other, but they can also be invoked separately by specifying the corresponding program argument. The program also keeps track of its progress, so it can be stopped and restarted at will and resumes from the last good known state.

The program accepts two different types of input file formats. Either an already georeferenced GeoTIFF format or any image format supported by OpenCV⁴ together with a metadata file. Multiple input files can be specified at once.

3.4 Implementing Pixel Data Based Matching

For the implementation of the pixel data matching phase the *cv::matchTemplate* method of the OpenCV library was used. This method implements different types of matching algorithms, as already explained in Section 2.4. Since the library version 3.0 the method allows to specify a mask to specify which regions should be used from the *template* to match against the *reference*.

Random forests, as proposed by P.Dollar et al. [29], are used to detect edges in the aerial image and the base image. To be able to use random forests for edge detection, a model file containing basic information about edges and corners is required. The already trained model available from the OpenCV Github repository⁵ is used in this work.

Since the results of the template matching with the generated edge images were poor in the beginning, the idea to mask all pixel values greater than a threshold was tested. Since the edge images only contain grayscale color values it was a relatively simple task to extract a thresholded intensity mask using the OpenCV library. The returned values using this intensity mask, which should lie between -1 for a total dis-match and 1 for a total match, were greater than 1 . Therefore this idea was abandoned. Further testing showed that the provided mask must be a continuous connected plane to get results in range.

The bigger the difference between the size of the template and the reference image is, the more meaningful results are returned. Hence we split the aerial image into small patches and provide a corresponding tile of the base image which has double the area than the aerial image patch as shown in Figure 3.2. Then all possible rotations between plus and minus two degrees are matched at

⁴https://docs.opencv.org/3.3.1/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fcea07b3bbd3a56

⁵https://github.com/opencv/opencv_extra/blob/master/testdata/cv/ximgproc/model.yml.gz

different scales. The results are then collected and statistical analysis is applied. We calculate the mean \bar{x} and the standard deviation σ of the returned results and only further process those results which are at least two times the standard deviation away from the mean.

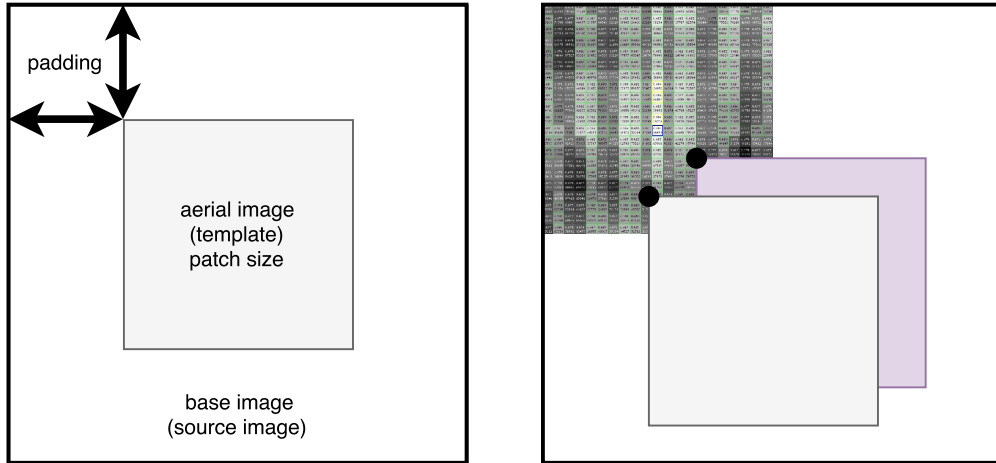
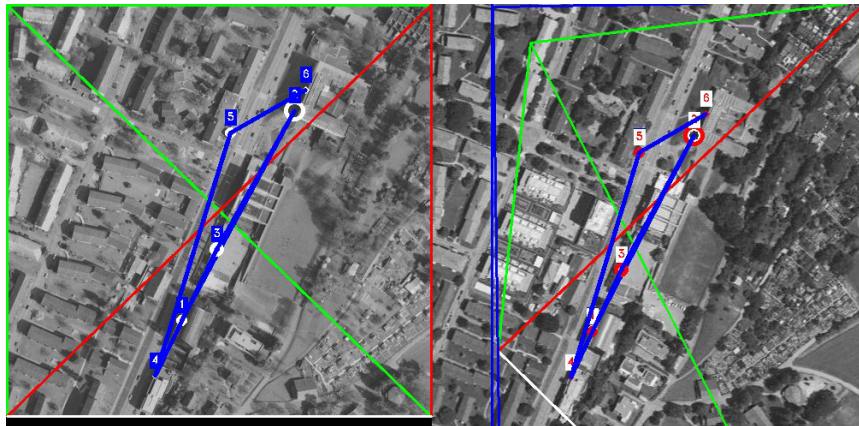


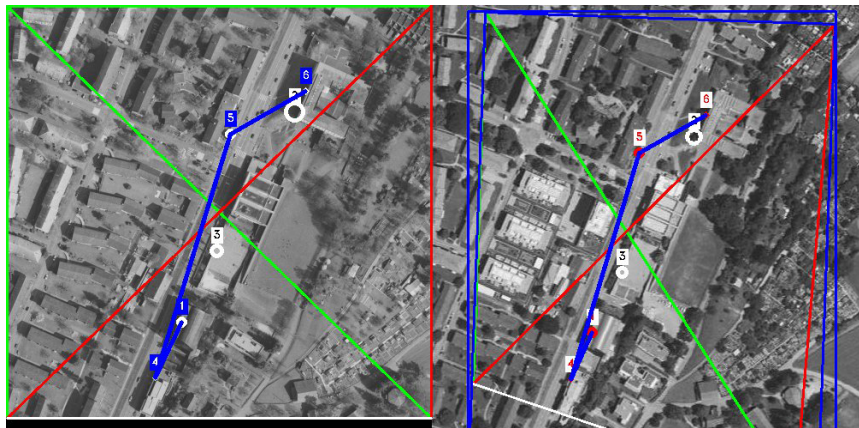
Figure 3.2: The constructed patch pairs are shown on the left. The base image patch is created with additional padding compared to the aerial image patch. This padding is needed, that the template matching algorithm can move the *template* around. The black dots in the right image show which values of the resulting matrix are calculated for two given displacements of the *template*.

We apply the template matching to the normal aerial image patches and the edge image patches simultaneously to get matches from both types. Unless there are clearly visible, some pixel wide edges in an image patch, the results from template matching with edge images are distributed less than once the standard deviation around the mean. Therefore, the results from the edge image patches consisting of only forest or other regions, where edge detection fails to extract clear edges, are unusable and discarded.

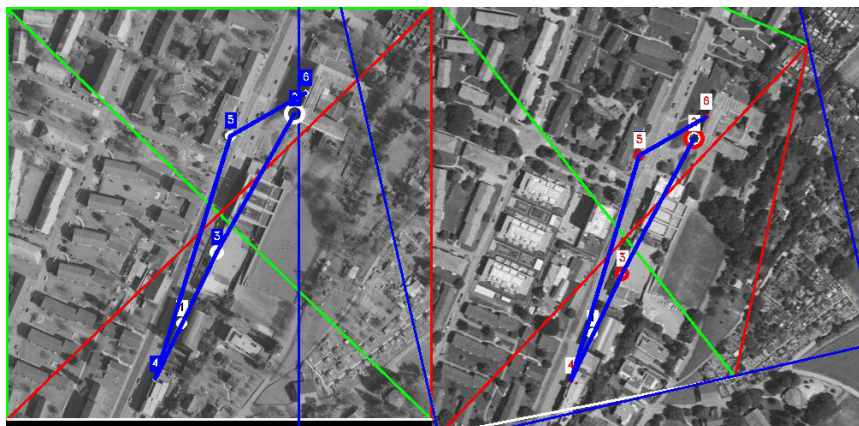
The current implementation is incomplete as it just gathers the data but will not apply any correction to the aerial image since the resulting data is still too noisy. By too noisy is meant that the scales, rotations and displacements gathered with this approach are a mix of all possible values, apart from a few exceptions like images consisting of roads or urban areas. Even when comparing two neighbouring patches, for which the template matching reported clearly significant data, one is estimated to have a rotation of 2 degrees while the other one is estimated to have -2 degrees of rotation. Therefore, further filtering and sorting must be applied to the results to handle cases with totally different resulting values for the scale, rotation or displacement.



(a) regular method



(b) LMEDS



(c) RANSAC

Figure 3.3: Comparison of the different homography estimation methods. On the left side is the aerial image, the base image is on the right. The aerial image has a red-green-white marker cross overlaid to track where it is mapped in the base image. Neither of the three methods can perform a proper estimation of the transform in the case where the points are nearly lying on an a straight line.

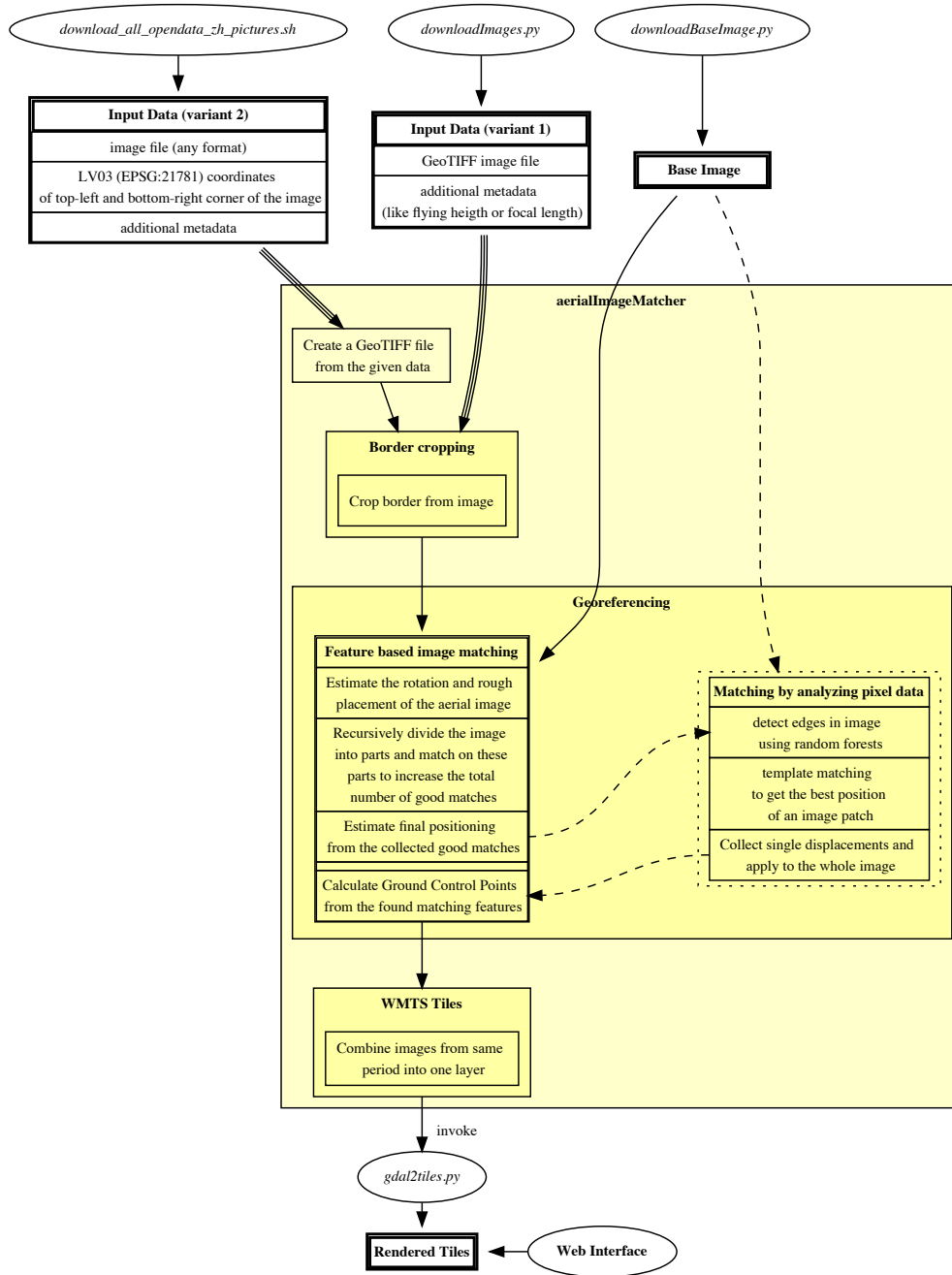


Figure 3.4: Overview of the current implementation. (The dotted box indicates a partial implementation, see Section 3.4)

Results

4.1 Feature Image Matching

4.1.1 Matching Quality

The matching quality greatly depends on the structure of the ground. When an image lies completely in an urban area, like the city center of Zurich, then the program finds dozens of good matches and achieves good precision (see Figure 4.1).

In more rural areas or more generally spoken in areas with less structured content, such as grass, lakes, fields or forests the matching quality diminishes (see Figure 4.2). The same is true for areas where a pattern is repeated multiple times, like many railway lines next to each other, the platforms at railway stations or garden plots.

For the less structured rural areas, the problem is that matches are likely to be rejected by the filtering described in Section 2.2.3. A patch without a clear structure inside it, for instance a patch inside a grass field, which is even optically really hard to distinguish from another randomly selected patch of grass, contains no meaningful data to be used in feature based image detection. The SIFT keypoint detector and the FLANN-based matcher rely on the structural content of the image patches to classify and match them with each other. The FLANN-based matcher is not able to clearly distinguish two features both lying in unstructured patches of the given image and therefore assigns about the same distance to the these features, if the features even get detected by the SIFT keypoint detector as an image feature in the first place.

For areas with repeating patterns inside of them the matches can be clearly assigned to a specific region, but due to the highly repetitive pattern the correct location of the match within that region can only be estimated by the detector. Therefore this points will be treated as outlier or impair the quality of the estimated homography. See Figure 4.4 for examples.

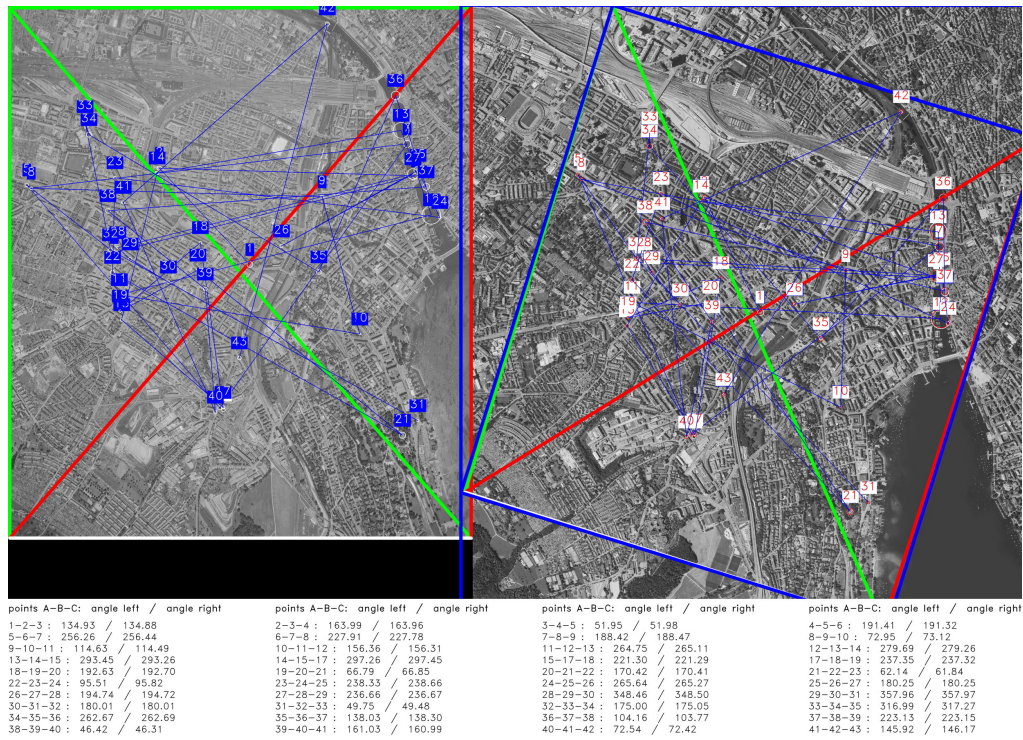


Figure 4.1: A pretty decent amount of images features are detected in the center of a city. The Figure shows the output of the geometrical path finding algorithm (see Section 2.2.4). The found matches are numbered and the fine blue lines show the found path. Below the image the angles between any three points on the path are listed. The green-red-white marker cross helps keeping track of the applied perspective transform. The thick blue lines around the marker cross in the base image on the right show the minimal area rectangle around the projected aerial image and the containing upright bounding rectangle.

4.1.2 Older Aerial Images

In past these old images were a big problem for the feature based image matching approach implemented there. With the techniques implemented in this work it depends less on the age of the image as on the difference of the structural data in the images. The image shown in Figure 4.1 is from the year 1957 but the matcher has no problems to identify enough features to get a good projection.

Even an image from the year 1931 has been tested multiple times and in one run an initial placement was possible. But the further processing stage was not performed, because the match uses too few points and covers a too small area. The found initial placement is shown in Figure 4.3.

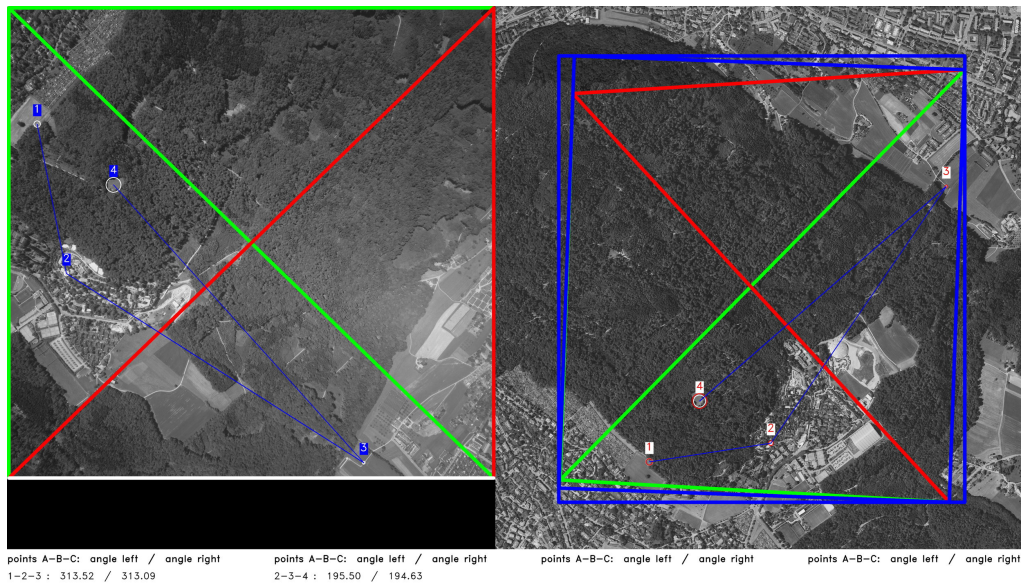


Figure 4.2: When most of the image is in a rural area, only a few good matches are found. Also the minimal area rectangle and the aerial image projection do not match up since the projection is under-determined in the area covered completely in forest.

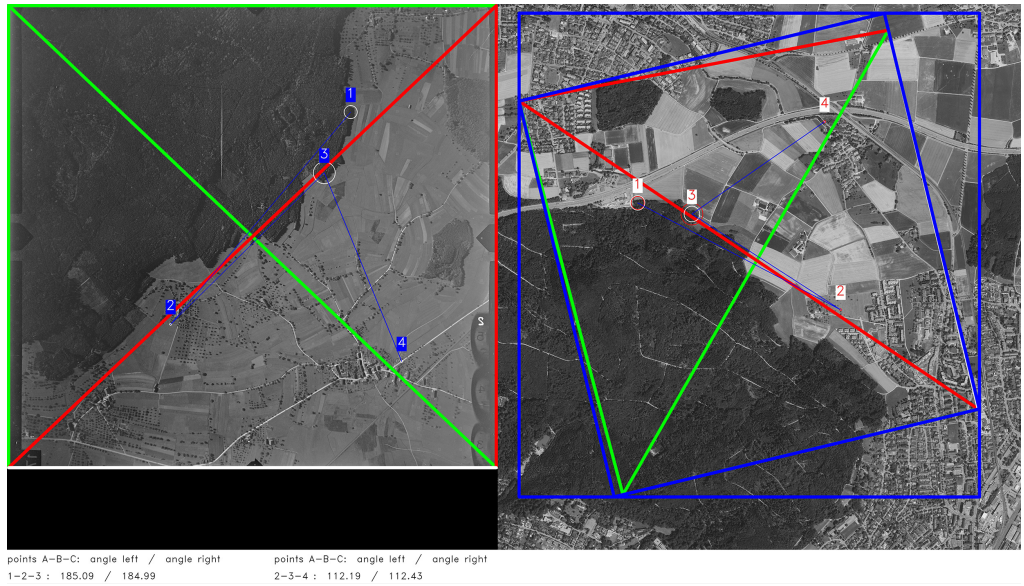


Figure 4.3: Visually good matching result with an image from the year 1931.

Nevertheless, the changes in the structure of the image grow with the age of an image, but the age of an image is not the main reason causing bad matching results.

4.1.3 Performance Comparison

Both implementations were tested using five different images. Multiple runs were performed with each image deleting all processed data between the runs. The values shown in the table below are the averages of the performed tests.

year	previous work		this work		
	georeferencing	applying final positioning	georeferencing	recursive search	applying final positioning
1931	46 s	n / a	495 s*	n / a	n / a
1955	16 s	n / a	113 s*	813 s	15 s
1978	63 s	99 s	64 s	653 s	12 s
1990	500 s	610 s	90 s	897 s	60 s
1998	28 s	n / a	228 s*	854 s	19 s
2005	122 s	150 s	97 s	1485 s	126 s

* The current implementation adapts its internal filtering state as described in Section 2.2.3 when no matches are found instead of directly aborting. But the additional effort pays off as a match is found for the images of the years 1955 and 1998, where the old implementation fails.

		previous work	this work (all steps)	this work (excluding recursive search)
averaged total	over all runs	179 s	1004 s	220 s
averaged total	only successful test runs	329 s	1106 s	165 s
minimum	only successful test runs	190 s	731 s	77 s
speed-up	average of successful test runs		-70 % (\approx 4 times slower)	99 % (\approx 2 times faster)
speed-up	minimum of successful test runs		-74 % (\approx 3.5 times slower)	147 % (\approx 2.5 times faster)

The matching step in previous work used only one SIFT keypoint detector while in this work seven differently configured SIFT keypoint detectors are used. This means there is seven times the work performed per execution of the matching step. Also accounting for this, the speed-up of about 100 % in run time can be compared to a speed-up of roughly 700 % in terms of performed calculations.

However, the total run time is approximately 4 times slower. This is due to the newly introduced recursive search step, that applies the matching step multiple times to only parts of the image.

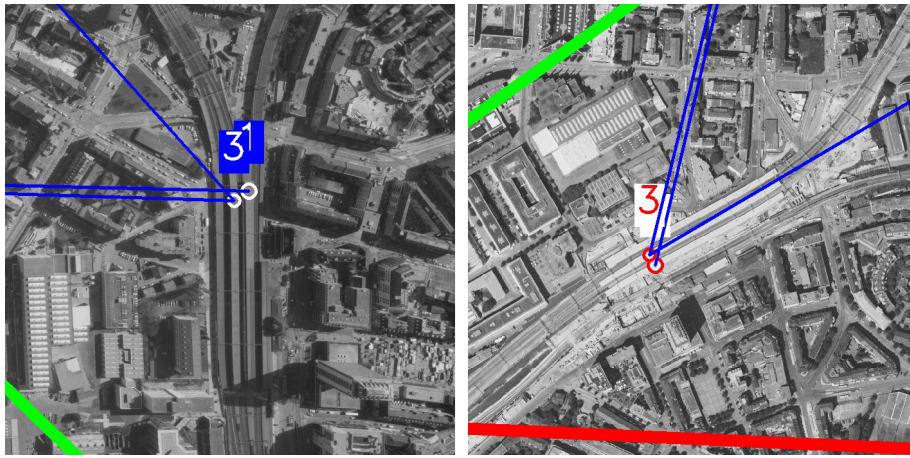
4.2 Pixel Data Based Matching

The pixel matching approach is not yet finished, although some partial quite promising results can be presented. The same problems are faced in the template matching approach as with feature base image matching when the ground does not have any clearly defined structural elements. Since template based matching “somewhat merge[s] the feature detection step with the matching part” of feature based image matching, as stated by Guido Bartoli in his survey about different image matching techniques [20], the two techniques are clearly related to each other and therefore when one fails on an image the other has a bad chance to deliver good results.

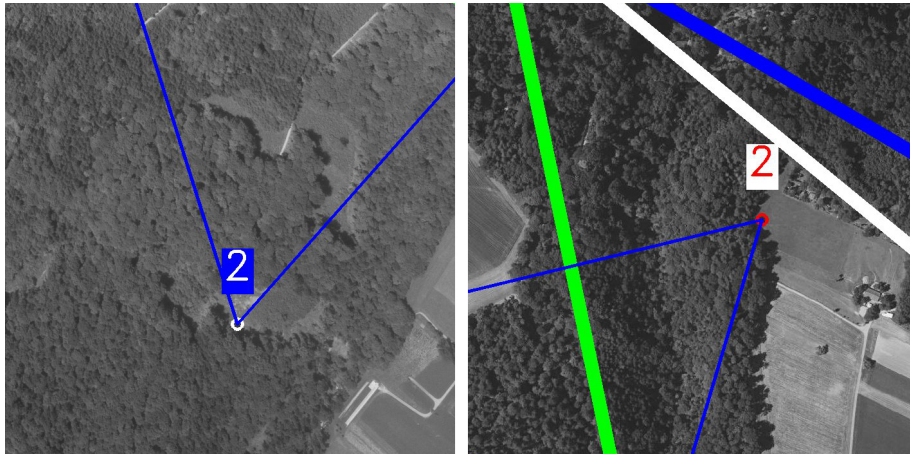
4.2.1 Optically vs. Mathematically Appealing

Despite implementing statistical filtering using the standard deviation, results which have a high correlation value can be placed far away from the correct position and results with a smaller correlation value are nearer to the ‘optical truth’ but rejected due to the small correlation value. Therefore the current approach needs to be refined to take this irregularities in the results into account.

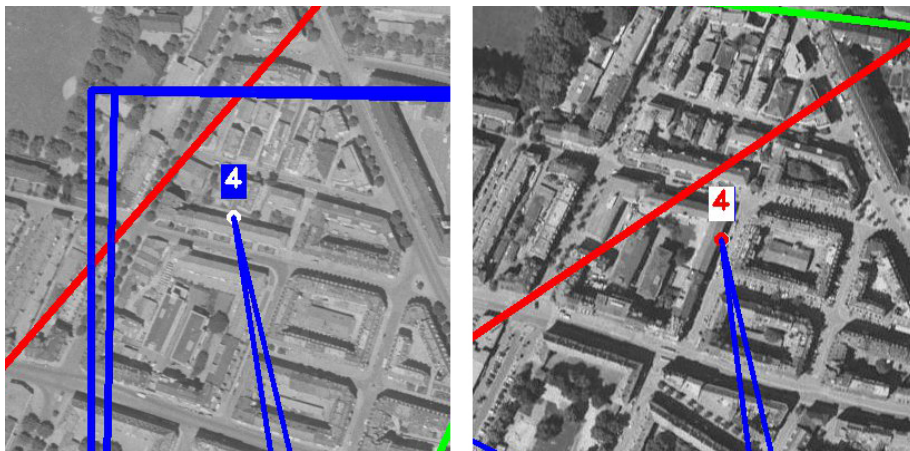
In Figure 4.5 the visualisation of the calculated date is shown. There is a white spot around the maximum value in the results considered a good match. The Figure 4.6 shows the results of template matching a single aerial image patch and the Figures 4.7 and 4.8 show the results for two different aerial images.



(a) An area with a repetitive pattern which causes the matches to be slightly displaced.

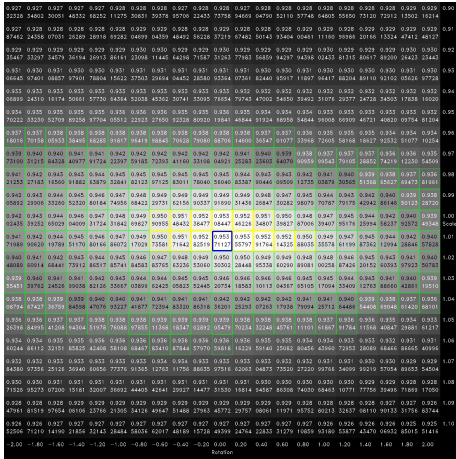


(b) An area with similar structural texture which causes matches to be in totally wrong places.

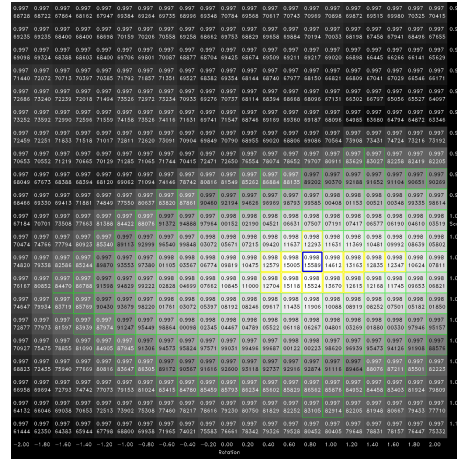


(c) The two images were taken at two different times of the day, therefore the shadow is cast in other directions.

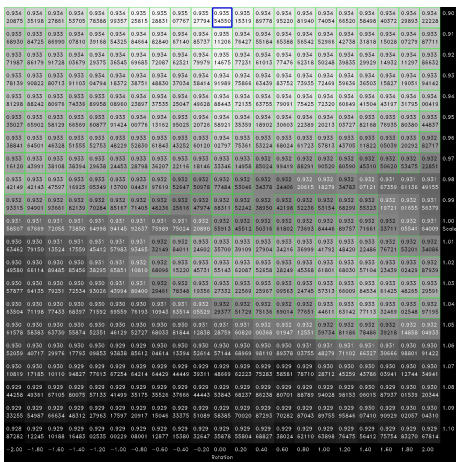
Figure 4.4: Examples of patterns that cause the feature based image matching to fail.



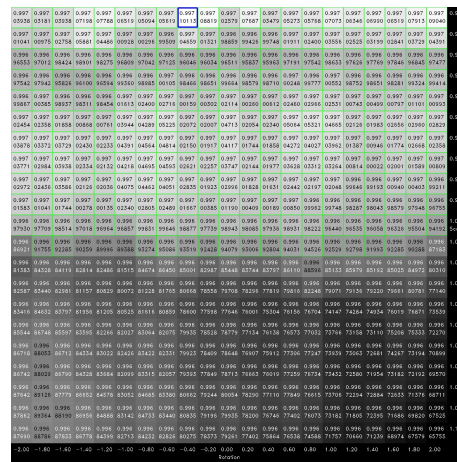
(a) good results by template matching the aerial image



(b) good results by template matching the edge image

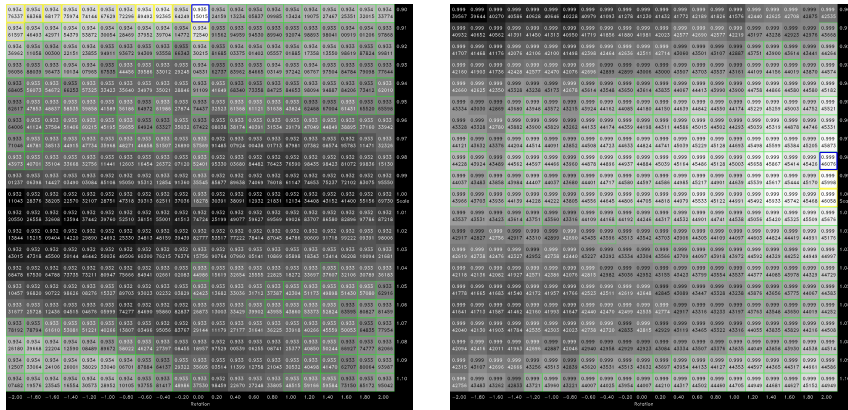


(c) discarded results by template matching the aerial image



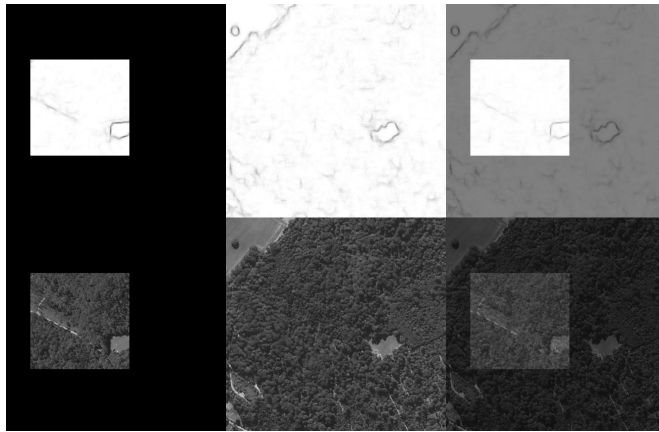
(d) discarded results by template matching the edge image

Figure 4.5: The visualized data returned by template matching. The blue square denotes the maximum value, the yellow ones are greater than $\bar{x} + 2\sigma$ and the green squares are above the median. In Figure 4.5a and 4.5b a clearly visible bright spot indicates good results.

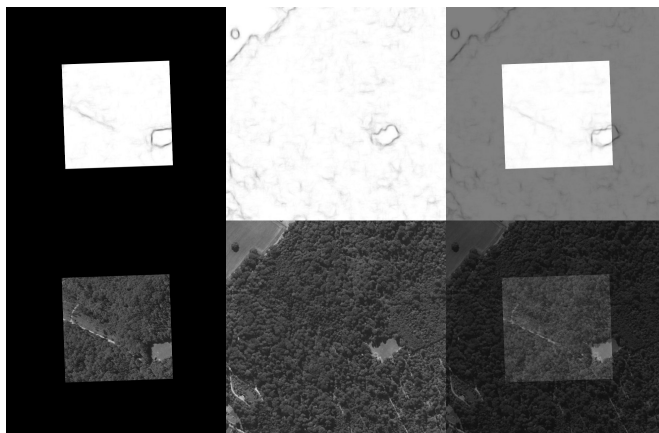


(a)

(b)



(c)



(d)

Figure 4.6: This single patch shows the advantage of an edge image to match the patches, when there are enough structures to detect dominant edges in the image. The Figure 4.6a shows the results from template matching for the aerial image and Figure 4.6b for the edge image. Below in Figure 4.6c, the best match placed onto the base image patch is shown using the aerial image as the template image and in Figure 4.6d for the edge image used as the template image respectively.

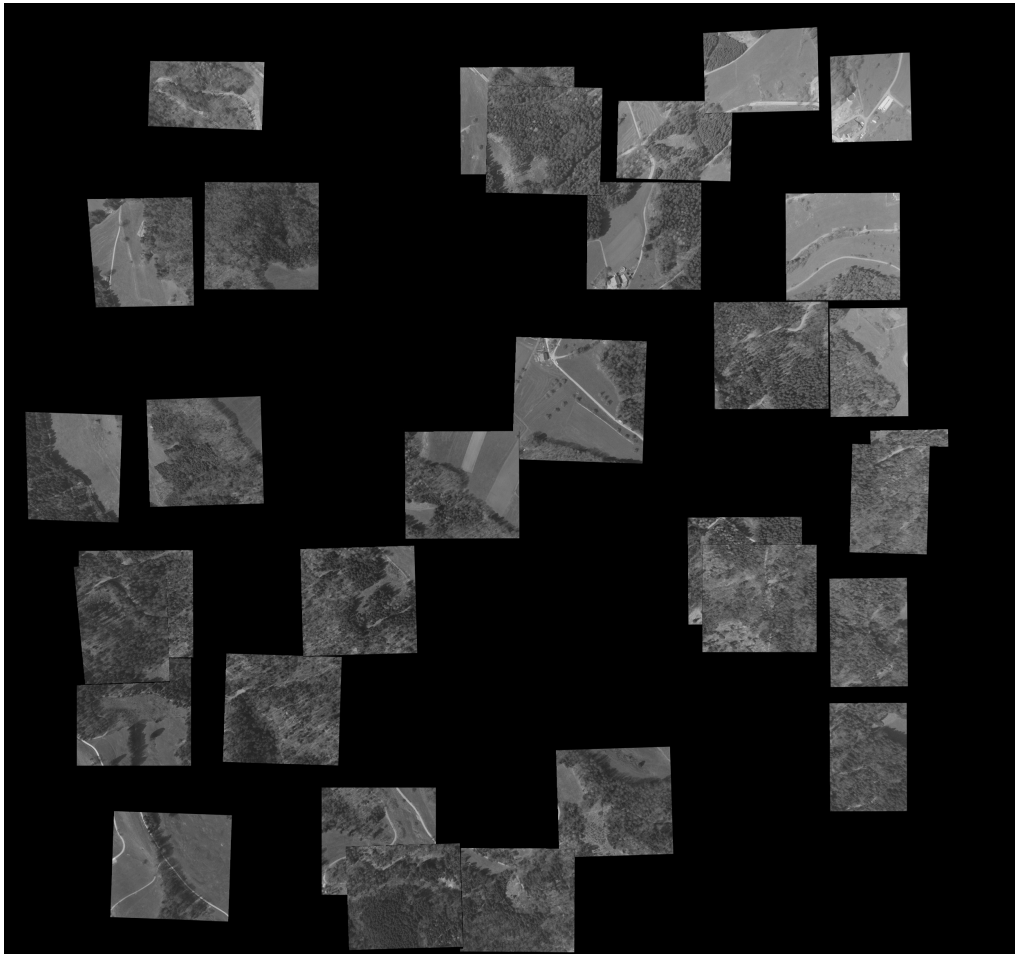


Figure 4.7: The image contains a lot of forest and from this results a rather bad placement of the matched tiles.



Figure 4.8: The best result of all tested images. The tiles in nearly the whole image are well aligned.

Conclusion

The goal of this work was to improve the image matching part of the previously written framework. To reach this goal the whole image matching part was on one hand rewritten in C++ on the other hand we started implementing new approaches like the path constructing algorithm or the pixel matching approach. We were able to slightly improve the level of accuracy and speed of the matching phase. Although the improvements made, aerial imagery taken from unfavourable terrains make the process still fail.

5.1 Outlook

Despite the effort put in this project, it is not yet complete. We use this section to wrap up open problems and give an outlook on what could be done to further improve this work.

5.1.1 Further Improving Pixel Data Based Matching

The pixel matching approach introduced in this work is a promising start to overcome some of the accuracy problems still faced. To overcome the weaknesses of the current approach, tools used for panorama stitching might be helpful. Furthermore, instead of using a pre-trained model for edge detection, a new model could be trained using only aerial images to achieve better edge extraction. The guide available from the OpenCV website¹ is unfortunately incomplete and partially outdated.

5.1.2 Image Matching Using Multiple Data Sources

The information provided by a single 2D image is limited and the most promising available methods to process a single image are by now implemented. Instead of analyzing just a single image at the time, multiple neighbouring images could

¹https://docs.opencv.org/master/d2/d59/tutorial_ximgproc_training.html

be analyzed together. Using such a technique regions with poorly defined image features could then be approximated with the information gathered from the neighbouring images. Also completely different data sources, for instance digital terrain models, LIDAR data or infrared imagery, could be used to improve the matching quality.

5.1.3 Accelerating WMTS Tile Generation

The WMTS tile generation is done by a helper script called *gdal2tiles.py*. Despite the recently added support to split up the workload onto multiple processes, the generation of an image layer still takes an excessive amount of time. Maybe there exists another solution to generate WMTS tiles more efficiently. Or if not, the WMTS tile generation process could be implemented in C++ following the WMTS standard [26].

Bibliography

- [1] Geographisches Informationssystem des Kantons Zürich (GIS-ZH): Luftbilder 1981-2000 (HistLuftZH). <http://geolion.zh.ch/opendata/> / <http://maps.zh.ch/?topic=HistLuftZH> / <https://opendata.swiss/de/dataset/luftbilder-zh-1981-2000>
- [2] Bundesamt für Landestopografie swisstopo: Swisstopo lubis viewer. https://map.geo.admin.ch/?topic=swisstopo&layers=ch.swisstopo.lubis-luftbilder_schwarzweiss&bgLayer=ch.swisstopo.pixelkarte-farbe&layers_timestamp=99991231&lang=en&catalogNodes=1430
- [3] Zinggeler, F.: Temporal map of switzerland. (June 2016)
- [4] Roth, M.: Accurate temporal map of switzerland. (April 2017)
- [5] Babri, U.M., Tanvir, M., Khurshid, K.: Feature based correspondence: A comparative study on image matching algorithms. *International Journal of Advanced Computer Science and Applications(IJACSA)* **7**(3) (2016)
- [6] Bay, H., Tuytelaars, T., Van Gool, L.: Surf: Speeded up robust features. In: *Computer Vision – ECCV 2006*, Springer Berlin Heidelberg (2006) 404–417
- [7] Rosten, E., Drummond, T.: Machine learning for high-speed corner detection. In: *Computer Vision – ECCV 2006*, Springer Berlin Heidelberg (2006) 430–443
- [8] Lowe, D.G.: Distinctive image features from scale-invariant keypoints. (January 2004)
- [9] Muja, M., Lowe, D.G.: Fast approximate nearest neighbors with automatic algorithm configuration. In: *International Conference on Computer Vision Theory and Application VISSAPP'09*, INSTICC Press (2009) 331–340
- [10] Muja, M., Lowe, D.G.: Scalable nearest neighbor algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **36** (2014)
- [11] Xia, Y., Chen, J., Li, J., Zhang, Y.: Geometric discriminative features for aerial image retrieval in social media. *Multimedia Systems* **22**(4) (Jul 2016) 497–507

- [12] Karel, W., Doneus, M., Verhoeve, G., Bries, C., Ressel, C., Pfeifer, N.: Oriental – automatic geo-referencing and ortho-rectification of archaeological aerial photographs. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* **II-5/W1** (2013) 175–180
- [13] Salehi, B., Zhang, Y., Zhong, M., Dey, V.: Object-based classification of urban areas using vhr imagery and height points ancillary data. *Remote Sensing* **4**(8) (2012) 2256–2276
- [14] Nebiker, S., Lack, N., Deuber, M.: Building change detection from historical aerial photographs using dense image matching and object-based image analysis. *Remote Sensing* **6**(9) (2014) 8310–8336
- [15] Korpela, I.: 3d treetop positioning by multiple image matching of aerial images in a 3d search volume bounded by lidar surface models. (05 2012)
- [16] de la Beaujardiere, J.: OpenGIS Web Map Server Implementation Specification, v. 1.3.0. Open Geospatial Consortium Inc. (2006)
- [17] Geographisches Informationssystem des Kantons Zürich (GIS-ZH): Luftbildkarte des Kantons Zürich. <http://wms.zh.ch/OrthoZHWS?service=WMS&request=GetMap&version=1.1.1&layers=orthophotos&srs=EPSG:21781&format=image/tiff&width=w&height=h&bbox=xmin,ymin,xmax,ymax>
- [18] Dubrofsky, E.: Homography estimation. Master’s thesis, The University of British Columbia (March 2009)
- [19] OpenCV team: OpenCV documentation. https://docs.opencv.org/3.1/d4/d1f/tutorial_pyramids.html
- [20] Bartoli, G.: Image registration techniques : A comprehensive survey. (2007) 16–18
- [21] Open Source Geospatial Foundation: GDAL - Geospatial Data Abstraction Library. <https://www.gdal.org>
- [22] OpenCV team: Open Source Computer Vision Library. <https://opencv.org>, <https://github.com/opencv/opencv>
- [23] OpenCV team: Open computer vision library - patented and non-free modules. https://github.com/opencv/opencv_contrib
- [24] Boost.org team: Boost C++ Libraries. <http://www.boost.org>
- [25] Progsch, J., Zeman, V.: A simple C++11 Thread Pool implementation. <https://github.com/progschj/ThreadPool> (2012)

- [26] Masó, J., Pomakis, K., Julià, N.: OpenGIS Web Map Tile Service Implementation Standard, v. 1.0.0. Open Geospatial Consortium Inc. (2010)
- [27] van Rossum, G., Python Dev Team: Python 3.6 C API. Samurai Media Limited (12 2016) Available also online in the Python 3.6 documentation: <https://docs.python.org/3/c-api/>, <https://docs.python.org/3/extending/>.
- [28] Fischler, M.A., Bolles, R.C.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM* **24**(6) (June 1981) 381–395
- [29] Dollár, P., Zitnick, L.: Structured forests for fast edge detection. In: *Proceedings of the IEEE International Conference on Computer Vision*. (12 2013)

APPENDIX A

Appendix

A.1 Using *aerialImageMatcher*

All paths referred to in this section are relative to the *project root directory* further called `$PROJ_ROOT`.

A.1.1 Installing Prerequisites

The software comes with an install script to install all prerequisites on an *Ubuntu Linux*. To invoke the install script, execute the following line in a bash compatible shell:

```
cd $PROJ_ROOT && ./INSTALL-UBUNTU.sh
```

This script installs the required dependencies including *GDAL*, the latest release of *OpenCV 3*, *npm* and sets up and starts an *nginx* web server instance to run the web interface.

A.1.2 Building

To build the main binary, the *CMake* build environment is used. There are two prepared scripts to either build a debug or a release version. The scripts can be run from any bash compatible shell:

```
source $PROJ_ROOT/code/cpp/aerialImageMatcher/buildRelease.sh
# or
source $PROJ_ROOT/code/cpp/aerialImageMatcher/buildDebug.sh
```

After a successful build operation the path to the executable is copied to the shell variable `$AIM_PATH`.

Build Flags The build of the executable can be influenced by the build flag `DRAW_IMAGES`, which enables all graphical output of the program. The purpose of this build flag is to visualize the progress of the matching phases. Use this build flag only for debug executables as a massive amount of image data is produced.

A.1.3 Image Downloading Scripts

Download the Base Image To download the base image from the WMS server of the canton of Zurich [17], edit the file `$PROJ_ROOT/code/python/downloadScript/downloadBaseImage.py` and adjust the region of interest (the variable is called 'bbox'). Also the hardcoded output path has to be changed if the default location `../../data/base/` is unavailable. Then execute the script:

```
/usr/bin/env python $PROJ_ROOT/code/python/downloadScript/downloadBaseImage.py
```

Download Aerial Images from OpenData Zurich In the directory `$PROJ_ROOT/OpenData_Zurich/scripts/` there is a download script located, which downloads all available images from the OpenData Zürich data set. The script can be used directly but it is not recommended to do so.

Instead use the *PHP* script located in `$PROJ_ROOT/code/php/generateDLBashScript.php` together with a filtered version of the *CSV* file downloadable from [1] to generate a new download script. This way the amount of data downloaded can be controlled. The whole image set is about 2 TB in size!

A.1.4 Usage

The program can either be invoked directly or via a helper script.

Processing Images from OpenData Zurich The *PHP* script located in `$PROJ_ROOT/code/php/generateDLBashScript.php` also generates a script to process all (previously) downloaded files in one go. The script will be written to `$PROJ_ROOT/OpenData_Zurich/scripts/`.

Program Options

```
Usage: aerialImageMatcher [--v|--vv] [-o] [-t maxNumberOfThreads]
                        [--pyProcesses maxNumberOfSpawnedPythonProcesses]
                        [-g] [-l logFile] [--save_storage_space]
                        [-c] [-r [--pixel_matching]]
                        [-w --web-root serverRoot --web_layer layerName]
                        [-b] baseImageDirectory
                        [-a] aerialImageFile [[-a] aerialImageFile2 [...]]
```

```
Usage: aerialImageMatcher --version
```

```
Usage: aerialImageMatcher -h
```

Options:

```
--version          print version
-h [ --help ]     print this help message
-v [ --v ]        verbose logging
--vv              extra verbose logging (WARNING: slows down program
                  execution drastically!)
-o                overwrite existing cropped, extracted or referenced
                  files created by an earlier run of
                  aerialImageMatcher.
-l arg            use given file as log file (new log entries will be
                  appended).
-g               search for a file <aerialImageFilename.ext.georef>
                  to initially georeference the aerial image
```

The file must be a text file containing the top-left and bottom-right corner coordinate in LV03 (EPSG:21781) in the following format:

```
top-left east / x coordinate\n
top-left north / y coordinate\n
bottom-right east / x coordinate\n
bottom-right north / y coordinate\n
```

Example:

```
600000.000\n
230000.000\n
620000.000\n
200000.000\n
```

```
--save_storage_space delete all unnecessary files to save space.
-c                  If specified, execute the image border cropping
                  step. If none of -c, -r or -w are specified, all
                  steps are performed.
-r                  If specified, execute the georeference step. If
                  none of -c, -r or -w are specified, all steps are
                  performed.
--pixel_matching   If specified, the georeference step also performs
                  pixel matching.
-w                  If specified, execute the WMTS layer creation step.
                  Needs --web_layer and --web_root to be specified.
                  If none of -c, -r or -w are specified, all steps
                  are performed.
```

<code>--web_layer arg</code>	Specify the date of the image (series) [YYYYMMDD] (to store the resulting WMTS layer).
<code>--web_root arg</code>	use given folder as web server root directory (to store the WMTS tiles).
<code>-b arg</code>	The directory containing the base image to compare the aerial images to. Directory must contain a "merged.vrt" file.
<code>-a arg</code>	Path to an aerial image. Should be a GeoTIFF file, otherwise the <code>-g</code> option must be specified.
<code>-t [--threads] arg</code>	set the capacity (number of threads) of the thread pool.
<code>--pyProcesses arg</code>	the number of processes that python is allowed to spawn.