Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

# Accurate Temporal Map of Switzerland

Bachelor Thesis


Markus Roth

maroth@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

**Supervisors:**
Manuel Eichelberger
Prof. Dr. Roger Wattenhofer


April 24, 2017

# Acknowledgements

I thank my supervisor Manuel Eichelberger for his support and ideas, András Bódis-Szomorú from the Computer Vision Lab for important inputs concerning feature matching and computer vision. I express my gratitude to Florian Zinggeler for his code and insights from his past thesis.

# Abstract

In this project, we improved an existing method that creates an interactive map of Switzerland. This map allows users to easily view aerial pictures from different years. This provides a chronological display of the changes that happened over the years and how both urban and rural areas developed. The map gets created automatically, using historical aerial pictures as inputs.

We improved the existing method in various aspects, such as runtime and used storage space. Computer vision approaches, like guided matching, were applied, for a more exact positioning of the picture.

# Contents

Chapter 1

# Introduction

Open Government Data Zurich [1] and the Federal Office of Topography, Swisstopo [2], released a large amount of aerial pictures dated from 1926 to 2007. The pictures are accessible on a website, but the user has to select each one by one. If one wants to see a larger area, this approach is rather time consuming and cumbersome.

The idea is to provide a better tool for users to observe larger areas, similar to Google Maps. Since we have a lot of data from different years, we also want to provide a journey through time, allowing users to see how a place has changed over time.

## 1.1 Goals

This project follows a semester thesis written by Florian Zinggeler [3]. He already developed a tool and method for what is described above. In his solution, however the matching process is not very accurate and fails completely on older image data. On the basis of the existing method, the objective is to improve the processes in different ways. Since his front-end application works well, our focus is on the back-end implementation. The main goal is to improve the automatic georeferencing of the aerial pictures by using a guided feature matching approach. That way we are able to provide more data to the front-end application and therefore enhance the user experience. We also try to optimize the method in terms of speed and storage space since large amounts of data have to be processed.

## 1.2 Related Work

As already mentioned above, our goal is to improve an existing method that was developed by Florian Zinggeler [3]. His approach is explained in Chapter 2.1.

Feature matching is a common technique in computer vision. Barbara Zitova

et al. [4] present in their paper a good survey over all different image registration methods and all necessarily steps needed for the matching process.

Jae Sung Kim et al. [5] used the Harris corner detector for feature matching. But their solution did not work well on images with a wide range in the flying height and is therefore not suitable for our method. Cléry et al. [6] proposed a solution that extracts line segments from raster images and matches them with vector data. As their solution sounds promising, it needs vector data, what we did not have.

There also exists work about guided feature matching by Ting Feng et al. [7] who use this approach to reduce the amount of wrong matchings. This approach sounds promising, but has not been tested on aerial pictures.

Chapter 2

# Optimizing Existing Method

In this chapter we explain our optimizations of the existing method by Florian Zinggeler [3]. We were given his code and are developing our version on top of his.

## 2.1 Overview of Existing Method

In the existing method, a script downloads all existing aerial pictures within specified area using a web crawler. The aerial images get downloaded in many small tiles which have the size of 256 x 256 pixels. In the following step, they get composed to the original image as it is stored on the Swisstopo server. After that, additional metadata like the flying height of the airplane and the center coordinates of the image are downloaded and stored along with the image. This data is not exact, as we are going to explain further in Chapter 2.5, but it helps to compute a good estimate of the image's coordinates. At the same stage, an already georeferenced base image, covering about the same area as the historical image, gets downloaded. This images's metadata contains its exact geographical location. The base image was shot in the year 2016 which means that big differences between it and the historical images may exist, as one can see in Figure 5.1.

After this step, the historical image gets automatically georeferenced using a feature matcher. This involves detecting features on both images, the base image and the historical one. Then, the features from both images are compared and get matched. This result is nicely visualized in Figure 4.1. The found matches can be used to assign coordinates to pixels of the historical image, as the coordinates of the corresponding features in the base image are known. Then, the coordinates and the corresponding pixels are used to warp historical image so, that it fit nicely together with all other georeferenced images. All images shot on the same date get rendered into one image. In the last step, tiles get generated to provide smaller data parts for the front-end application. A nice overview of the whole process is given in Figure 2.1.

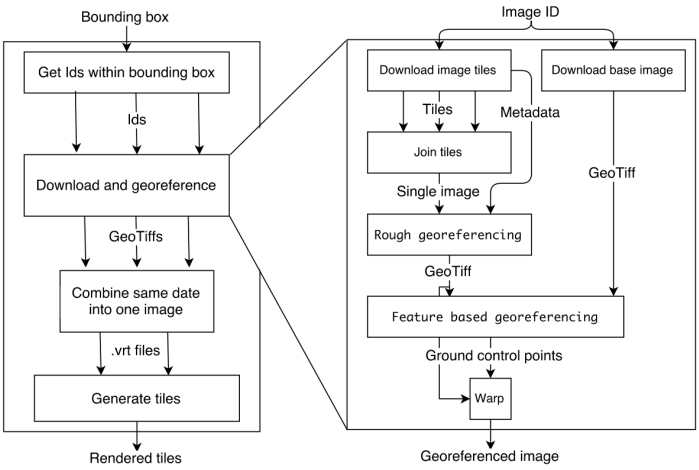



Figure 2.1: Diagram of the existing process (By Florian Zinggeler [3])

## 2.2 Optimizing Storage Space

In Florian Zinggeler's method [3], the GeoTiff format [8] is used to store all georeferenced images. Along with the raster image, the GeoTiff format can store large amounts of useful metadata like coordinate system, georeferenced points and map projection. This increases the compatibility with other GIS software, such as QGIS, which could be used for manual georeferencing. The drawback of this format is the large file size, as all the pictures are commonly stored uncompressed.

We get the image tiles as JPEGs from our our crawler, thus using lossy data from the beginning. Therefore, converting them later to a much larger GeoTiff file needs a lot of unnecessary space without improving the image quality. However, the loss of all the metadata would involve a lot of code refactoring and implementation of a system which stores metadata. GDAL [9], the geospatial library that is used to handle GeoTiffs, allows storing the images in JPEG and the geographical metadata in an auxiliary XML file.

The images in Figure 2.2 were cropped and enlarged so that it is easier for the reader to compare the quality. As one can see above, the quality of the pictures looks the same for the human eye, and a factor of approximately 20 in terms of used storage space is gained. Using this approach also helps with debugging, as we can access the meta data in an XML file via a simple text editor and without relying on other software.

(a) The 677 Megabyte GeoTiff version (b) The 33 Megabyte JPEG version

Figure 2.2: Comparison of image quality between GeoTiff and JPEG (Images from [2])

## 2.3 Optimizing Computation Time

As most aerial images are rather big (up to 16,000 x 16,000 pixels), processing them may be time consuming. Composing the downloaded tiles takes more time than the more important georeferencing. Florian Zinggeler's method [3] uses imagemagick [10], an open-source software for converting and composing images. This approach seems to work very slow with big inputs, as up to 4,500 tiles have to be joined for a single aerial picture.

To improve this step we decided to implement the join procedure by ourselves. This was done using the computer vision library OpenCV [11] and Numpy [12]. Our solution is about five times faster and is possibly even faster with more parallelization, since the current implementation only uses a single core.

## 2.4 Optimizing Feature Detectors and Descriptors

In the previous project [3], several image feature detectors and descriptors have been evaluated. Florian Zinggeler chose to use ORB [13], as it has a fast runtime and returns more matches than most other detectors. The SIFT [14] detector and descriptor provided even better results, however, had a much longer running time. Compared to SIFT, ORB is not patented and available in the standard OpenCV library. As our main goal is to improve the number of matchings, we decided to use SIFT.

## 2.5 Georeferencing Using Guided Matching

### 2.5.1 Overview

The low amount of correct matches was a big problem in the Florian Zinggeler's approach [3]). It can bes assumed that one reason for this, is that big aerial pictures have a lot of similar looking features such as crossroads, trees and corners of buildings as shown in Figure 2.3.

Figure 2.3: Visualization of the ORB feature descriptors (Image from [2])

In cities like Zurich, similar features can be distributed over the whole area of the image. Florian Zinggeler used a brute-force matcher provided by the OpenCV library. This matcher compares one feature with all features in the set of the other image and returns the closest match [15]. As the base and the historical image always contain variation, having a lot of similar features increases the probability of wrong matches.

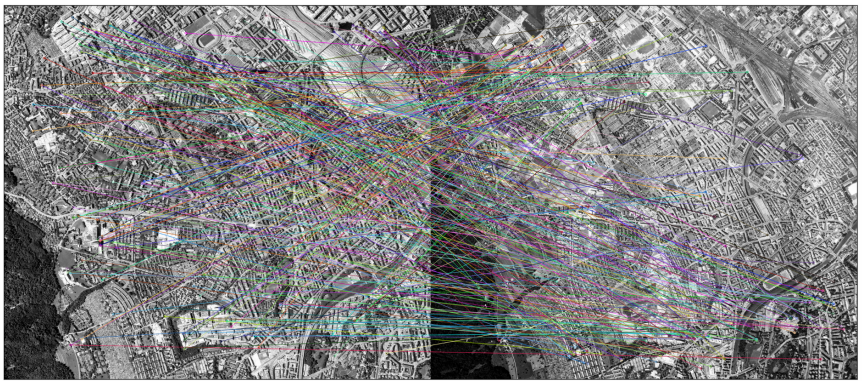
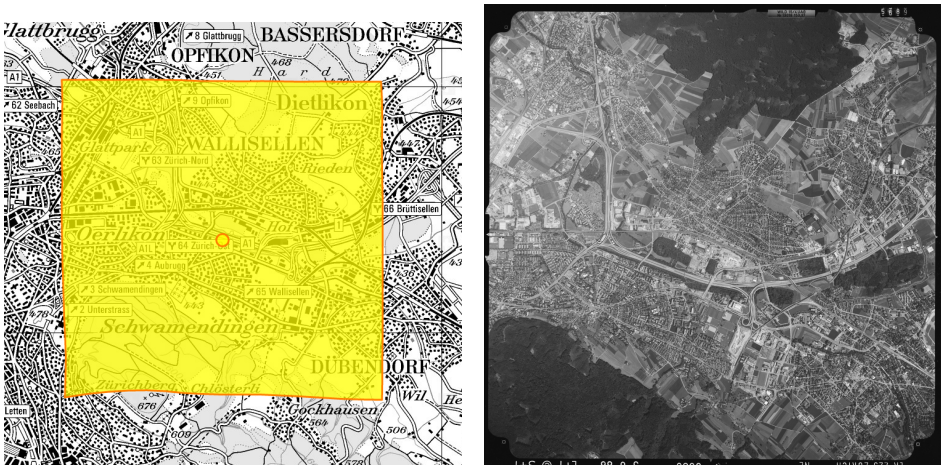


Figure 2.4: Unfiltered matches of ORB features using a brute-force matcher, using fewer features than in the final implemented method for better visibility (Images from [2])

Our objective is to fix this by using a guided matching approach that allows matching features only in a smaller neighborhood. With this approach, we are able to reduce the number of wrong matchings. To be able to calculate a good correct and neighborhood, both pictures need to cover about the same area.

We tested if the rough georeferences from Swisstopo were good enough. The problem was, that the bounding box, downloaded from the Swisstopo REST API [16], does not provide accurate data, as shown in Figure 2.8. The upper right corner shows a much larger area compared to the bounding box.

[!tbp]




(a) The bounding box according to the API data

(b) The aerial picture

Figure 2.5: Comparison between bounding box of an aerial picture and the actual image (Images from [2])

Other images provided more accurate data, but it was impossible to decide at the download time which data was exact and which data was not.

For calculating more exact neighborhoods we decided to split the matching process into multiple stages:

1. Feature matching on a low image resolution is performed. This way the number of wrong matches can be reduced as there are not many similar looking features left. The found georeferences are used to calculate precise bounding box coordinates of the historical aerial picture.

2. A new base image gets calculated, based on the bounding box from the previous step. The base image and the historical now cover almost the same area. This is where guided matching can be applied, using an image mask which allows matching features only in specific parts of the pictures. In this step we use a higher resolution of the image than in the first step for detecting more features.

3. If there are still not enough matched features after the second step, the matching may be computed again in an even smaller neighborhood.

### 2.5.2 Calculating the Correct Base Image

To be able to apply the image mask in the second step, the historical image and the georeferenced image need to have the same rotation and to cover approximately the same area. We can use the stored bounding box of the historical picture and crop a fitting base image with this information.

As shown in Figure 2.5, bounding boxes describe the corner coordinates of a georeferenced image. If the image is rotated, the corner coordinates are chosen so that the whole rotated image fits inside the box, as shown in Figure 2.6.

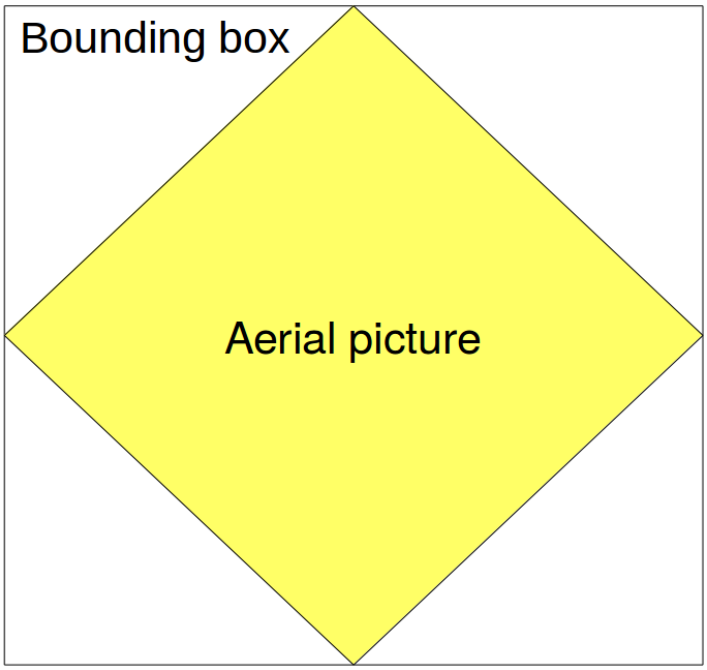



Figure 2.6: A rotated picture and its corresponding bounding box

Additionally to the rough bounding box, Swisstopo also provides information about the rotation of the aerial pictures. If there is no rotation, getting a fitting base image is trivial, as we can just crop an image from the already georeferenced base image. This can be done by the GDAL library, using the corner coordinates as input.

In the case of rotation, the problem is harder. First, we need to decide if the base or the historical image has to be rotated. We do not want to waste any picture information from the historical data by cropping too much of the picture or rotating it. Therefore, we decide to rotate the base image as it can cover a much bigger area and we do not lose visual information.

We start with an image that has the size of the bounding box and then, we rotate the image in the reverse direction of the historical image's rotation (Figure 2.8b). After this step, both pictures, the base and the historical, have the same orientation. Now, we can calculate the corresponding corner pixels of historical image. Figure 2.7 shows the current situation. For calculating the desired point (mx, my), we are using the parametric equations for the vectors $\overrightarrow{OM}$ (from (0, 0) to (mx, my)) and $\overrightarrow{AB}$ (from (ax, ay) to (bx, by)). The point (mx,my) can be expressed with the two following equations:

$$(ax, ay) + t * \overrightarrow{AB}$$
$$(0, 0) + s * (-y/x)$$

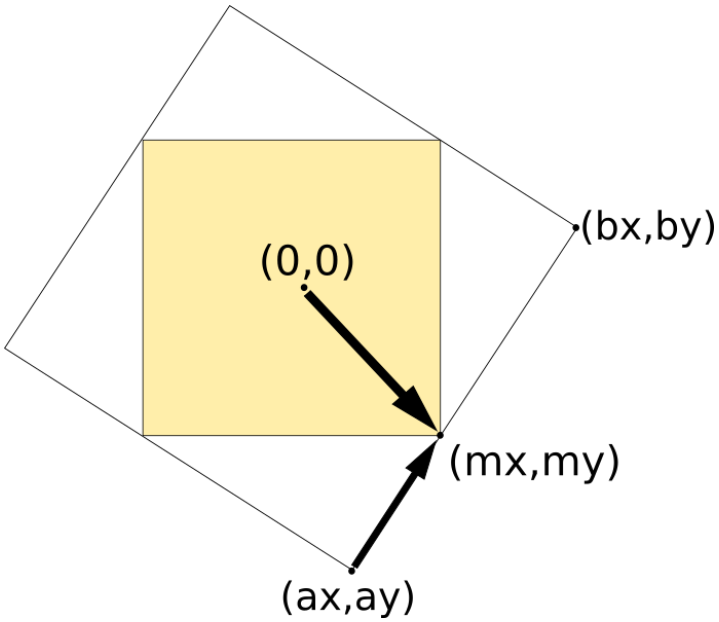
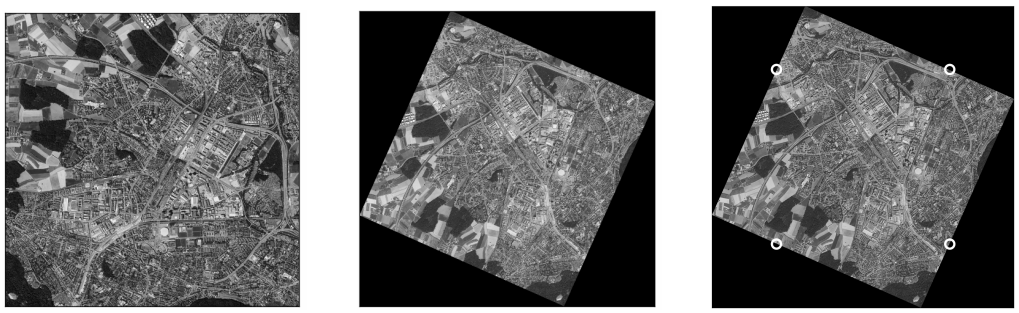



Figure 2.7: The rotated base image (white) with the historical picture inside. The arrows show the parametric equations with the desired parameters which get calculated in our method

where x is the width of the historical aerial picture and y its height.
This can be formulated as an linear equation system:

$$ax + t * (bx - ax) = s * x$$
$$ay + t * (by - ay) = -s * y$$

In our method, the values for variables s and t get calculated by Numpy. The upper left part can be calculated by using the parametric equation for $\overrightarrow{OM}$ and negating s. Having the pixel coordinates for M and the opposite corner, we are finally able to crop the proper base image.



(a) The cropped base using the bounding box's coordinates

(b) After the rotation

(c) The calculated points

Figure 2.8: The implemented calculation of the bounding box (Images from [2])

Chapter 3

# Implementation

## 3.1 Used Programming Languages and Libraries

We kept most parts of the implementation from the code developed by Florian Zinggeler. His code was written in Python, using multiple additional libraries. For geospatial data, the GDAL library [9] is used. It provides many useful features such as the handling of GeoTiff files. We can also use it to warp aerial pictures based on georeferenced points. This is used to remove the perspective distortion from the original image. For computer vision tasks, the OpenCV library [11] is used, as it provides many of features for feature matching and is well documented.

## 3.2 Adding More Robustness to the First Matching Step

All the aerial pictures not only differ from the years they were shot, but also from a lot of other different aspects as flying height and size of the covered area. To give our algorithm more robustness, we decided to apply the first step from our guided matching algorithm in Chapter 2.5 multiple times, using different resolutions. This is implemented because we were testing an ideal resolution for the first matching step. Due to differences among the pictures, there was no ideal resolution. Some images already had good matches with a height of only 150 pixels, others needed a height of approximately 300 pixels for a good estimate.

Our final implementation uses a loop that that iterates in steps of 40, starting from a height of 120 pixels up to 600 pixels. We only store the result of an iteration if it contains more than 9 matches, as a lower number of results often contain wrong matches. At the end of the loop, all results with more than 9 matches are combined and are then used for the calculation of the more precise bounding box.

Chapter 4

# Results

## 4.1 Increased Number of Matches

Using SIFT instead of ORB and the guided matching approach significantly increased the number of found matchings during the automatic georeferencing step.

In Figure 4.1 we can see the found matches from Florian Zinggeler's method [3]. All 40 matches are concentrated in the center of the picture.

In one iteration of the first stage, our approach already produces much better results as illustrated by Figure 4.2. Compared to Figure 4.1, our matches are much better distributed over the whole picture.

Figure 4.3 shows the accumulated matches over all defined neighborhoods. After that we still have several wrong matches that need to be filtered. We use the RANSAC algorithm [17] to accomplish this, as RANSAC detects and eliminates outliers.

After applying RANSAC, the valuable matches in the corner of the images disappear, as one can see in Figure 4.4. This leaves us with the same problem that already existed in the old method, although we have much more matches. Additional to our final matches we could also use all the matches from the previous stage, as they are distributed more evenly. The georeferenced points assign a coordinate to a pixel, low-resolution georeferences, since the ones from the first stage, lack accuracy. However, since the matches from the first stage have a better distribution, we decided to add them to the ones from the second stage and use them for the final warping of the historical aerial picture.

One reason for the unsatisfactory result at the edges of the image could be that the cropping after the second stage does not seem to be exact. Unfortunately, We did not manage to identify the source for this mistake.

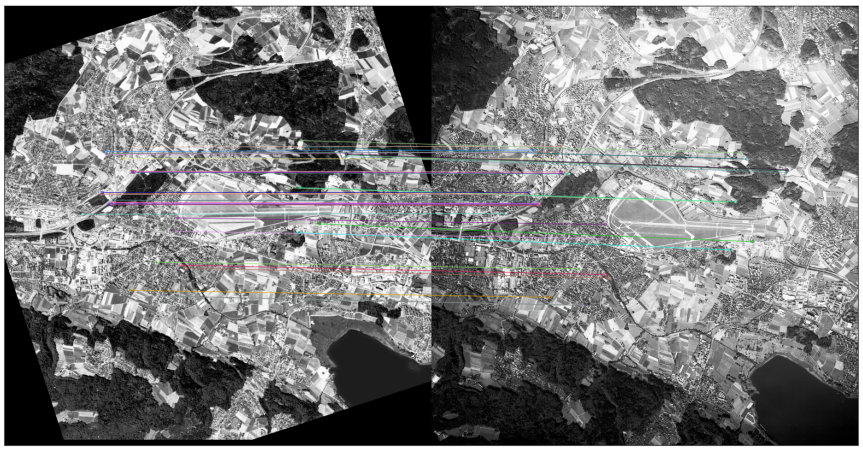


Figure 4.1: The found matches in Florian Zinggeler's method [3] (Images from [2])

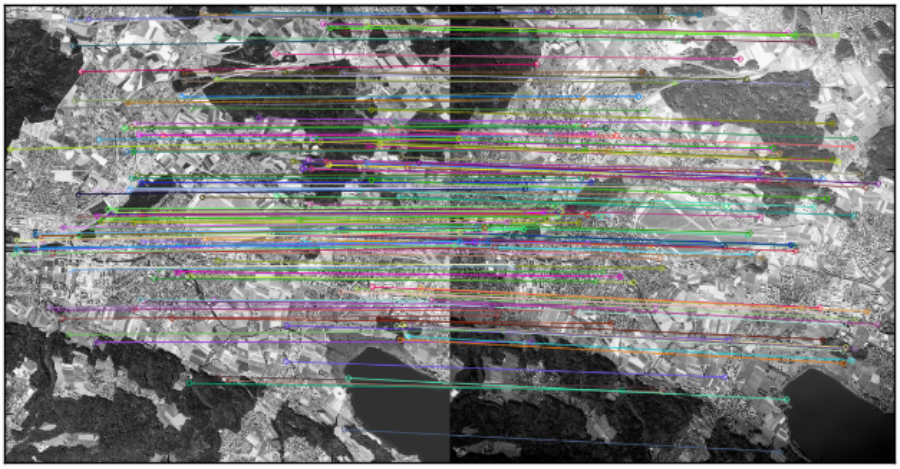


Figure 4.2: Matches in one iteration of the first stage (Images from [2])

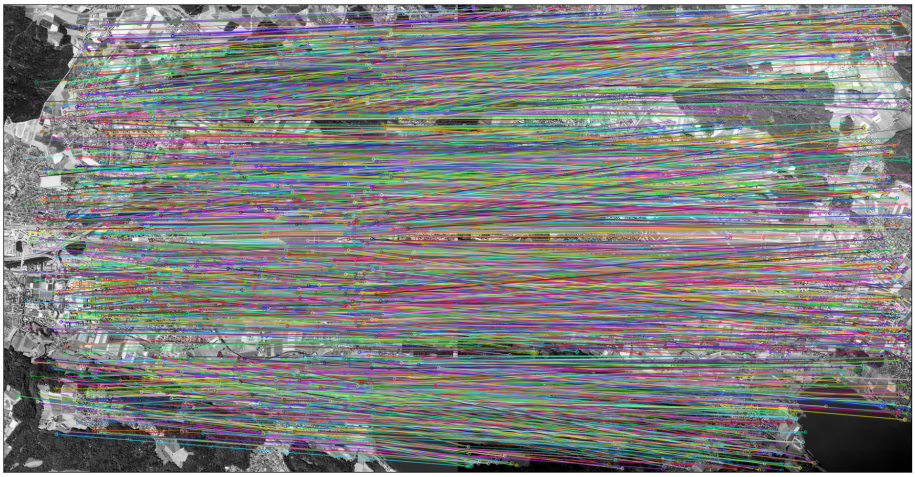


Figure 4.3: Unfiltered matches produced in the second stage, using the masks (Images from [2])

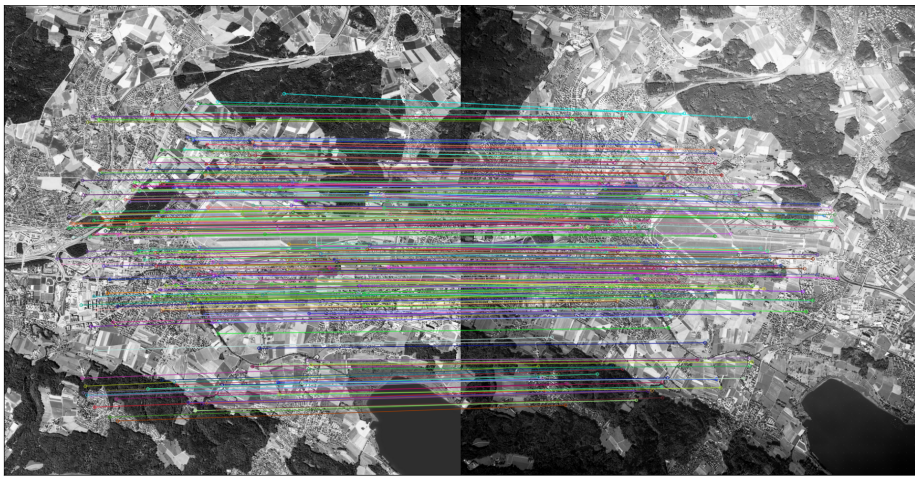


Figure 4.4: The matches after applying the RANSAC algorithm (Images from [2])

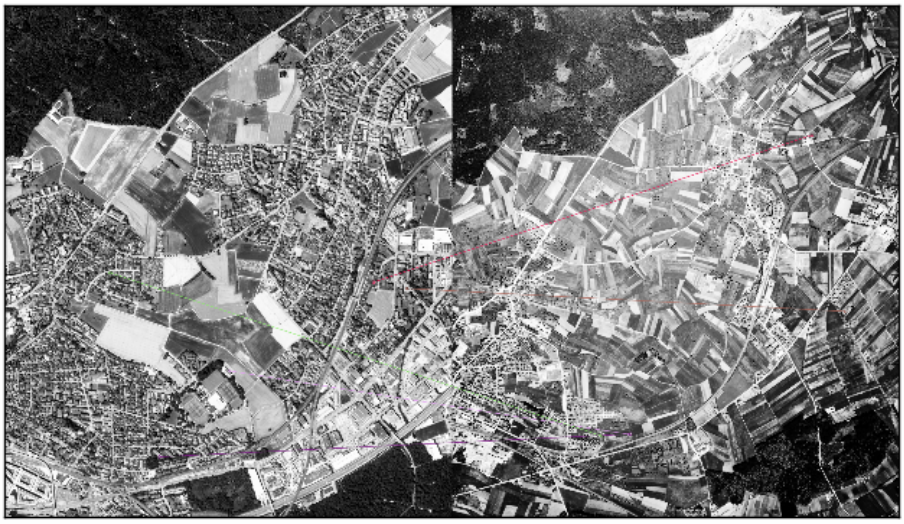


Figure 4.5: Wrong matches in Florian Zinggeler's method [3] (Images from [2])

## 4.2 Performance on Older Image Data

With the higher amount of matches we were hoping to be able to match more older images, since the old method was unable to produce a good result, as one can see in Figure 4.5. The picture on the right dates back to the year 1951 and a lot of changes have happened since then.

We were hoping that our more robust guided matching implementation is able to detect more correct features. Unfortunately, our implementation already failed during the first stage. In none of the different iterations from the first stage, enough features were found. An example is shown in Figure 4.6, higher and lower resolutions performed similarly.

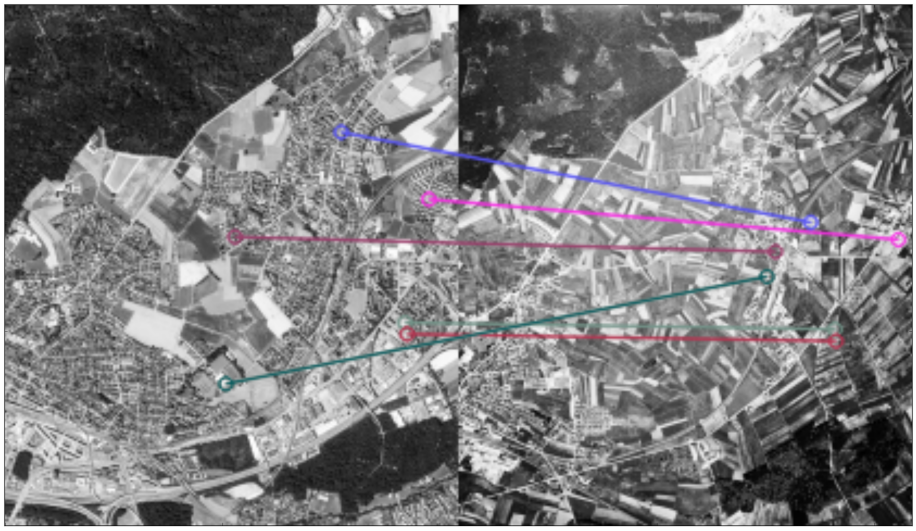


Figure 4.6: Wrong matches with our approach during the first stage (Images from [2])

Chapter 5

# Conclusion and Outlook

We presented multiple improvements to the existing method of Florian Zinggeler [3]. Ranging from performance improvements to improvements regarding the used storage space. We developed an approach that results in a much higher amount feature matches when compared to the previous solution. Despite these improvements, the automatic georeferencing algorithm often fails on old images, dating from the 1960s and before.

## 5.1 Outlook

Although we were able to increase the amount of matches, our algorithm still often fails on old image data. In this chapter we give an outlook what could be done for further improvements.

### 5.1.1 Using Graphics Cards for Better Performance

Our multistage approach involves multiple matching processes. As the image resolution grows bigger in every step, the computational time for the matching algorithm grows in time too. OpenCV provides a CUDA library [18] which could increase the performance of this process by a factor of 5.

### 5.1.2 Matching Street Data

The course of streets does not change as often as buildings or other features. Matching streets may, therefore, be a good approach, especially for older data. This was done by Cléry et al. [6]. They extracted line segments in historical pictures and matched them with vector data from streets. The implementation of this would either require a high-resolution vector map of the Swiss street network or a robust algorithm which is able to extract street data from aerial pictures.

Figure 5.1: Comparison between the base image and a aerial picture of the same area from 1931 (Images from [2])

### 5.1.3 Georeferencing Back in Time

Matching a current aerial picture with an image from 1931 is challenging, even for humans, as one can see the many changes in Figure 5.1. We could solve this issue by first georeferencing the newest aerial picture and then using the already georeferenced ones for older data. With this approach, the changes between the images would be much smaller and a feature matching algorithm should be able to find much more matches. The influence of errors would have to be observed in this approach, as the propagation of them could lead to wrong georeferences in older pictures.

# Bibliography


[1] "Gis-zh: Open government data." http://maps.zh.ch/?topic=HistLuftZH. Online; accessed April 24, 2017.

[2] "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. Online; accessed April 24, 2017.

[3] F. Zinggeler, "Temporal map of switzerland," June 2016.

[4] B. Zitova and J. Flusser, "Image registration methods: a survey," *Image and Vision Computing*, vol. 21, pp. 977–1000, 2003.

[5] J. S. Kim, C. C. Miller, and J. Bethel, "Automated georeferencing of historic aerial photography," *Journal of Terrestrial Observation*, vol. 2, no. 1, p. 6, 2010.

[6] I. Cléri, M. Pierrot-Deseilligny, and B. Vallet, "Automatic georeferencing of a heritage of old analog aerial photographs," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 2, no. 3, p. 33, 2014.

[7] T. Feng and J. Yuan, "Feature point detection and matching of wide baseline image based on scale space theory and guided matching algorithm," in *Multimedia Technology (ICMT), 2011 International Conference on*, pp. 538–542, IEEE, 2011.

[8] "Geotiff file format." https://trac.osgeo.org/geotiff/. Online; accessed April 24, 2017.

[9] "Gdal - geospatial data abstraction library." http://www.gdal.org/. Online; accessed April 24, 2017.

[10] "Imagemagick library." https://imagemagick.org/. Online; accessed April 24, 2017.

[11] "Opencv library." http://opencv.org/. Online; accessed April 24, 2017.

[12] "Numpy." http://www.numpy.org/. Online; accessed April 24, 2017.

[13] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," 2011.

[14] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[15] "Opencv brute-force matcher." http://www.docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html. Online; accessed April 24, 2017.

[16] "Swisstopo api rest services." https://api3.geo.admin.ch/services/sdiservices.html. Online; accessed April 24, 2017.

[17] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, pp. 381–395, June 1981.

[18] "Opencv cuda library." http://opencv.org/platforms/cuda.html. Online; accessed April 24, 2017.