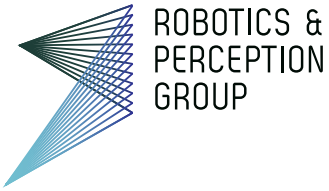
University of Zurich
Department of Informatics



Fabian Weiersmueller

Automated Order Retrieval Of Ultra Thin Brain Sections

Bachelor Thesis

Robotics and Perception Group
University of Zurich

Supervision

Thomas Templier
Prof. Dr. Richard Hahnloser
Prof. Dr. Davide Scaramuzza

December 2015

Contents

Abstract

Numerous scientific fields in neuroscience rely on high resolution maps of brain regions. One hopes to understand the basic principles of memory, learning and many diseases of the brain. The traditional way to create such brain maps consists of producing hundreds of consecutive, ultra thin sections of the given region of interest. The Hahnloser Group at the Institute of Neuroinformatics is currently developing a new method to collect these ultra thin brain sections in a fully automated manner. This new method has many advantages over traditional methods, however, it comes at a cost: we are losing the information about the sections' position inside the stack.

The purpose of this thesis is to present the pipeline we developed to retrieve the sections' true order. This retrieval is not as straight forward as it seems, since one has to deal with many associated challenges, such as missing image alignment and file sizes up to several gigabyte per section image. The pipeline we developed is able to deal with all those challenges, using various methods of image processing such as SIFT features, image transformations and normalized cross correlation. The first big goal is to compute the similarities between all pairs of sections. Once this list is available, one has to find the best possible way to stack the sections. This is done by solving a travelling salesman problem instance.

Additionally to just the presentation of the pipeline, this thesis explains the observations, experiments and decisions that led to this pipeline and the methods and algorithms used in it. Finally, the paper documents our successful attempt of retrieving the order of our own dataset.

Nomenclature

Acronyms and Abbreviations

EM	Electron Microscopy
LM	Light Microscopy
MDE	Mean Displacement Error
NCC	Normalized Cross Correlation
SIFT	Scale Invariant Feature Transform
SOR	Section Order Retrieval
TSP	Travelling Salesman Problem

Chapter 1

Introduction

1.1 General overview

Connectomics is an emerging discipline of neuroscience that aims to reveal and decipher the wiring diagrams formed by interconnected brain cells. It carries the hope of understanding the building units of memory, learning processes, diseases of the brain and might eventually help the development of treatments for diseased brains. A main prerequisite is the creation of high resolution maps of brain regions, and we concentrate on a part of this process in this thesis.

A current approach to produce high resolution imagery of brain tissue consists of producing hundreds of consecutive ultra thin sections of a given region of interest. This is called serial section microscopy. Typically, these sections have a thickness between 20 and 50 nm. After collection of the sections on an appropriate substrate, the sections are scanned using high resolution imaging methods such as high resolution light microscopy (LM) or electron microscopy (EM). Further, related neural structures have to be identified on each section, so that one can track objects over multiple sections. In the end, the sections can be put back in order, and all structures in the examined block of brain can be reconstructed, providing finally a wiring diagram of the interconnected brain cells.

The Hahnloser laboratory is currently developing a method to cut and collect these brain sections in a fully automated fashion that circumvents many drawbacks from existing methods. It comes however at the cost that it is not possible to keep track of the order of the sections during the section collecting process. The goal of this Bachelor's Thesis is to retrieve the exact order of the sections based on information contained in their imagery using methods from computer sciences and image processing.

Our starting point is the end product of the section collection and imaging pipeline: high resolution imagery of each of the collected sections without any information about the order of the sections. As a rough estimate, the image size of one section is about 35'000 x 35'000 pixels ($35'000^2$ px), that is, about 1 GB on disk. They can, however, reach up to more than $100'000^2$ px and sizes up to

more than 30 GB. Each section is made of a mosaic, built up from many smaller patches that are stitched together to form a complete section. Figure 1.1 shows such an image and highlights the subpatches.

1.2 Image Similarity Principle

Three options were considered in the laboratory to retrieve the order of the sections that is lost during the collection process:

- Video tracking of the sections during the entire collection process from the sectioning of each section until it is placed onto its definitive position on the substrate
- Incorporation of marks with a laser into the sectioned block of tissue
- Order retrieval based on acquired imagery of the sections

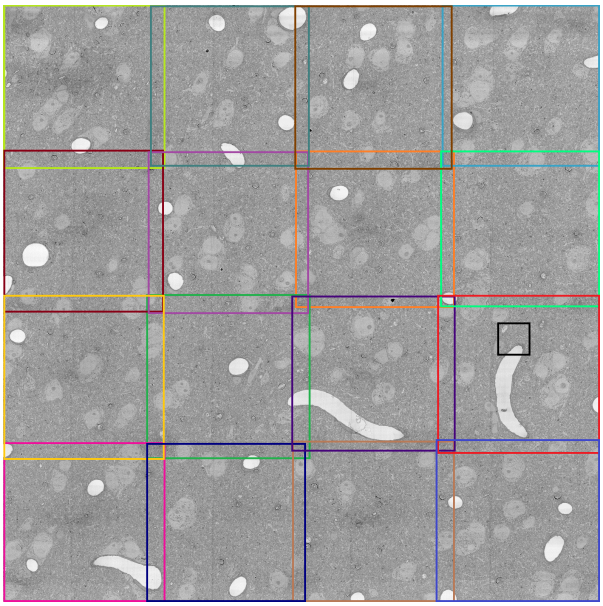
In this thesis we have explored the third item. We want to find the exact position of a section in a stack of images solely based on its appearance and that of all other sections.

Our starting point is the following simple observation: close neighbouring sections look similar while distant sections look dissimilar. Is this observation true for all sections, so that we can use it to retrieve their order?

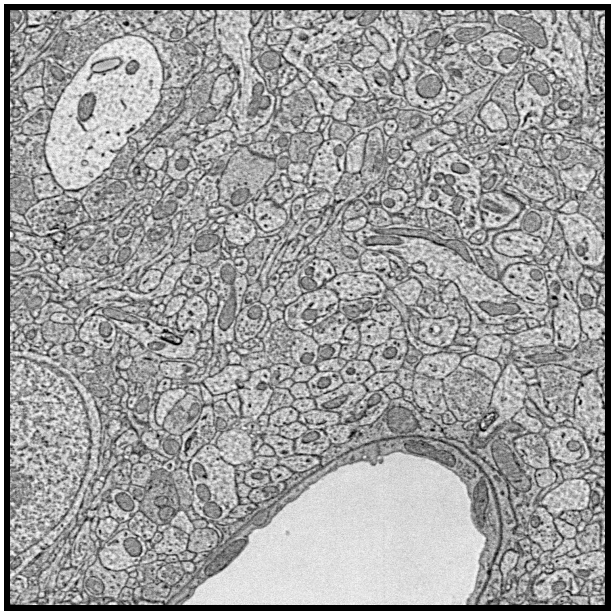
For a first assessment of the problem we computed an image similarity measure between pairs of sections based on normalized cross correlation (NCC), which is later described in detail in the Methods 2 section. A convenient way to notate this is in the form of a matrix. We define the similarity matrix $\mathbf{M}$ to be the matrix of all similarity values between pairs of sections or images $\mathbf{s_{a,b}}$, where a and b are the indices of two sections with $0 \leq a < m, 0 \leq b < m$ and m is the number of sections.

$$\mathbf{M} = \begin{pmatrix} 1 & \mathbf{s}_{0,1} & \cdots & \mathbf{s}_{0,\text{m-1}} & \mathbf{s}_{0,\text{m}} \\ \mathbf{s}_{1,0} & 1 & \cdots & \mathbf{s}_{1,\text{m-1}} & \mathbf{s}_{1,\text{m}} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{s}_{\text{m-1},0} & \mathbf{s}_{\text{m-1},1} & \cdots & 1 & \mathbf{s}_{\text{m-1,m}} \\ \mathbf{s}_{\text{m},0} & \mathbf{s}_{\text{m},1} & \cdots & \mathbf{s}_{\text{m,m-1}} & 1 \end{pmatrix} \tag{1.1}$$

The matrix has a number of interesting properties. First, notice that the main diagonal entries refer to the similarity value of a section to itself. Second, it is useful to normalize the similarity values to lie between 0 and 1. In such a case, the similarity of a section to itself is by definition 1. Third, note that $\mathbf{M}$ is symmetrical, since the similarity of sections A and B is the same as the similarity of B and A. Last, we would like to point out that all similarity values of neighbouring sections lie on the first lower and upper subdiagonal of $\mathbf{M}$.



(a)



(b)

Figure 1.1: (a) The image of one section from our own dataset 2.1.2. Highlighted in different colors are the individual tiles that form the section. Each tile has a size of $8'500^2$ px, leading to a section image of around $32'000^2$ px. The white structures are blood vessels, the lighter gray ones are cell bodies of neurons. The space between is filled with various other cell types, cell organelles, as well as axons. (b) A subpatch of 1500^2 px. It shows the tip of a blood vessel and various different cell types and cell organelles at high resolution. Its location on image (a) is indicated by the small, black square.

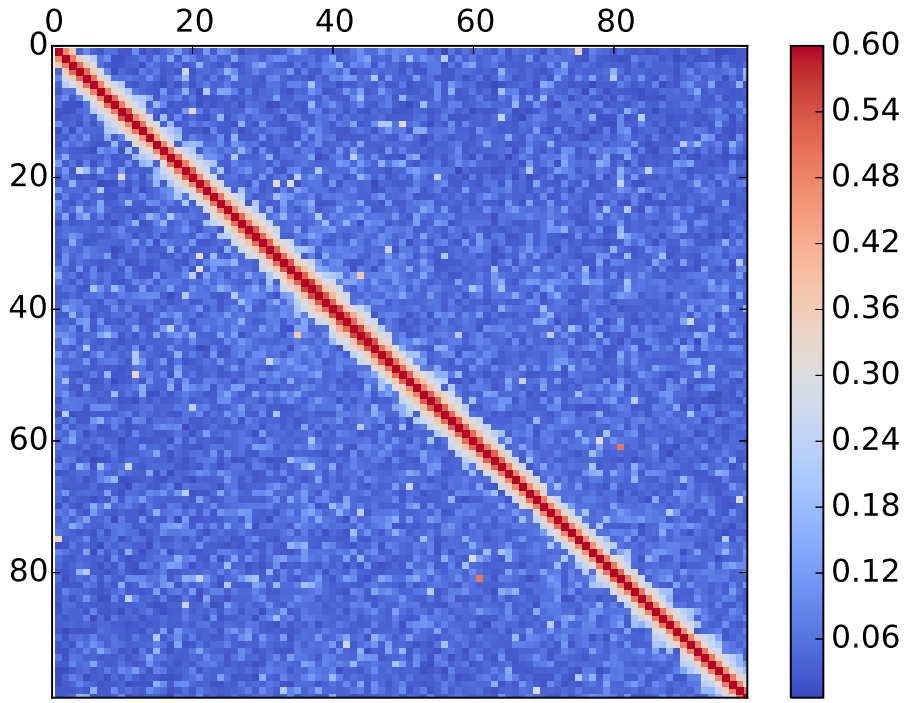


Figure 1.2: This figure shows the similarity matrix of 100 sections. The values were obtained using normalized cross correlation on subpatches of 1500^2 px size. The sections came from the Kasthuri et al. dataset 2.1.1. The high values along the diagonal imply high similarity among neighbouring sections, giving us a reasonable chance of identifying their order with high accuracy.

We empirically measured the similarity distribution of 1000 sections from which we already knew the correct order. Of special interest were the similarity values of neighbouring sections, e.g the values of the first upper diagonal. Figure 1.2 presents the results for a subset of 100 consecutive sections, since all properties can be seen and explained better on a smaller subset. The similarity values of most of the section pairs lie between 0.05 and 0.1. Only the values close to the main diagonal are significantly higher, lying between 0.25 and 0.5. This is true on average up to a 4-neighbourhood, meaning sections that are at most 4 sections away from each other. This observation is the foundation of the approach presented in this thesis. It confirms the assumption that consecutive sections are more similar than non-consecutive ones, and that this can be measured. The purpose of this thesis is to present different methods that all use image similarities to accomplish the goal of high accuracy section order retrieval (SOR).

1.3 Order Retrieval

Once a similarity matrix is created, we are faced with the problem of finding the best possible way to stack the sections. A natural way of thinking about the problem is in terms of 'distances'. In this interpretation, two similar sections are "close" to each other, while dissimilar sections are "far" away from each other. The question of "what is the correct order of those sections ?" therefore becomes "what is the shortest way to connect those sections that maximizes the sum of the similarities of consecutive sections ?". This is the famous problem of theoretical computer science called the Travelling Salesman Problem (TSP): Given a number cities, what is the shortest path to visit all cities exactly once? In our case, the "cities" are the sections, and their distances are the similarities between them. The TSP is a well studied problem that is difficult to solve, and to only cite one reference, Papadimitriou et al. [20] proved the NP-completeness of this problem.

For an introduction in the Travelling Salesman Problem, as well as its numerous solution approaches, we point the reader to the overview of Laporte et al. [18] and Hornik et al. [14].

There are many methods available to solve the problem, exact and approximative. We tested a number of solution approaches, and our results are presented in Section 3.2.3.

1.4 Related Work

Little work has been done so far for the retrieval of the order of consecutive sections, as all serial section microscopy techniques can preserve the order of sections during section collection and imaging. The closest contribution comes from the work of Hanslovsky et al. [13]. In their paper, the authors deal with the problem of automatically estimating the thickness of sections. Acquiring ultra thin sections is a challenging task, and the true physical thickness of a section may deviate greatly from the value set on the sectioning machine. Hanslovsky et al. used NCC as a measurement of section similarity and developed a method to simultaneously estimate the thickness of a section as well as dealing with image artifacts. They were also able to recognize single missing sections as well as correcting swaps in the sections' order. These swaps, however, were only locally in a 4-neighbourhood. Our method is improving this greatly, being able to fully retrieve the section order over hundreds of sections, ignoring section thickness variations in the process.

1.5 Problems Of Size and Alignment

In addition to the central problem of missing order information, we face two problems that are less obvious, but nonetheless crucial for the SOR to succeed.

1.5.1 Size

Naturally, we acquire brain imagery in regions encompassing meaningful neuronal circuitries. At the same time, image resolution should also be as high as possible, so that synapses and other subcellular structures are visible. This comes at the cost of growing data sizes, up to a point where even a single mosaic subimage is too big to be processed in one pass. The method presented in this thesis needs to take this into account, since we can never process the whole section image in its full size at full resolution. We can, however, reduce one of those characteristics either by downsampling or by taking subpatches. Both is done at different stages of the final method we present.

1.5.2 Alignment

Image alignment describes the process of matching corresponding regions on different sections, and placing them exactly above each other. Examples of aligned and non-aligned images are given in Figure 1.3 to give a better impression.

Alignment is required before one can take any subpatches from any two sections, since we have to guarantee that the same features are visible on both subpatches. Additionally, the section images need to be aligned before one can proceed with other processing steps, such as creating a 3D model of the cellular structures. Again, the sheer size of the sections makes this a challenging task, as we need to find a way to align the images without ever seeing the whole section in full resolution.

1.6 Goals Of This Thesis

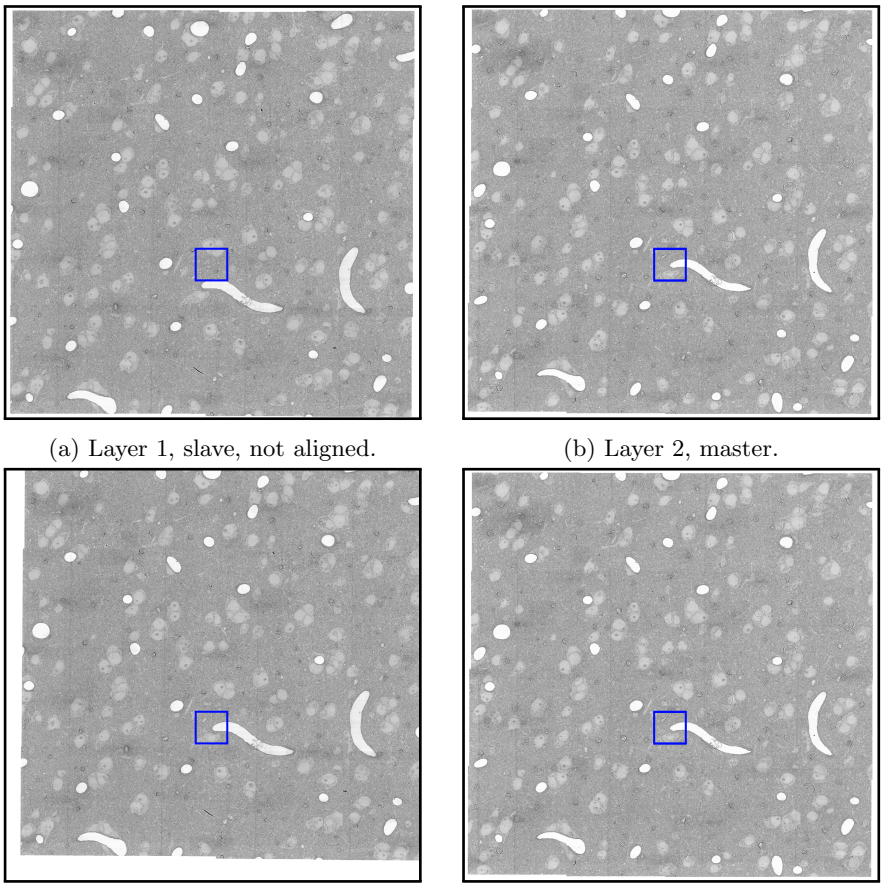
Before starting this thesis, we had a lot of questions that we hoped to answer in the course of this work. We primarily worked with three datasets, Bock11, Kasthuri11 and Templier220, shown in detail in Section 2.1. Bock and Kasthuri are pre-aligned section stacks from the openconnectome project [1] that are both freely available. Templier220 is our own dataset, which is not yet aligned. Our questions can be grouped the following way:

1.6.1 NCC or SIFT

- Should we use an intensity based similarity measure such as NCC 2.4.2 or a landmark based such as scale invariant feature transforms (SIFT) 2.4.1?
- What are the strengths and limitations of both methods?

1.6.2 Minimal Resolution

- What is the lowest possible resolution to which we can downsample the Bock11 and Kasthuri11 datasets and still be able to correctly retrieve their order?



(a) Layer 1, slave, not aligned.

(b) Layer 2, master.

(c) Layer 1, slave, aligned to the master.

(d) Layer 2, master.

Figure 1.3: This figure shows the effects of image alignment. The images (a) and (b) show two sections before the alignment, (c) and (d) show the same sections after the alignment. To match corresponding regions on two images, one has to change only one of those images. We call the image that is changed 'slave' and the one that stays the same 'master'. Therefore, images (b) and (d) are exactly the same. The blue squares have a real size of 2500^2 px and are all at the exact same position on each image, always at pixel coordinates $(14'900, 18'940)$. Note further that without alignment, two subpatches at the same position on two images can contain completely different parts of the images. This effect gets worse the smaller the subpatches get.

- Is the resolution of LM data high enough for SOR, or do we need EM images?

1.6.3 The Templier220 Dataset

- Is the image quality of Templier220 sufficient to guarantee successful SOR?
- How should a pipeline look that is able to perform SOR on the Templier220 dataset?
- What methods should be used?

1.6.4 Computational Complexity

- How much time do we need to solve 1000 sections if they are already aligned?
- Can we do SOR of 1000 sections on a standard desktop PC, or do we need high end machines? Do we even need a cluster?
- How long does it take to solve SOR for non-aligned images?
- How long does it take to find the optimal path among all sections?

1.6.5 Tools

- What tools should we use to work with image data? Is there one tool that fulfils all our needs?
- What tools are available to find the optimal path among all sections?
- Do we need to implement a graph optimization algorithm ourselves?
- Are all those tools freely available?

Chapter 2

Methods

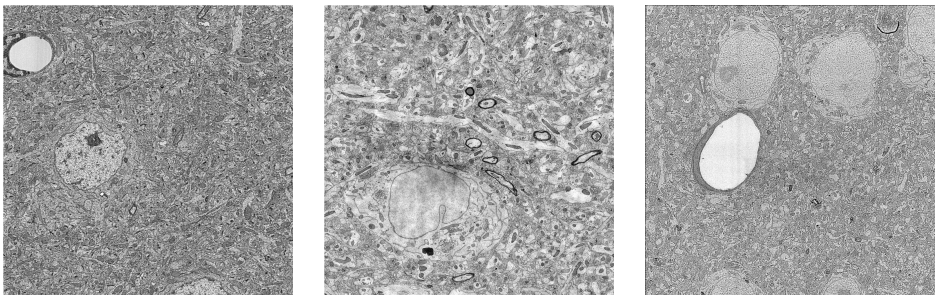
This chapter should present to the reader all methods we either used in our final pipeline, or that were tested but ultimately dropped. Furthermore, it documents the environment used to run all code and the origin of the image data used.

2.1 Data

An important aspect of this thesis is the data we are working with. Since our own data was unordered, we were dependent on already aligned sections on which we could test our methods and hypotheses.

2.1.1 The Open Connectome Project

One major source of data was the Open Connectome Project [1]. The website contains section data from various animal species and in various degrees of resolution. Additionally, the data can be downloaded using a Python API, which makes working with it convenient.



(a) Bock11. (b) Kasthuri11. (c) Templier220.

Figure 2.1: Shown are examples of all 3 datasets we used. All subpatches shown here have the same size of 4000^2 px.

The benefit of using this data is that the images are already perfectly aligned and the order of all sections is known. From all sets available, we used the ones from Kasthuri et al. [15] and from Bock et al. [4]. For both, Figure 2.1 gives an example of how the data look. The section images from Bock et al. have a maximum size of $40'000$ x $50'000$ px. However, a lot of sections do not show the maximal size and some sections are missing all together. We were able to find 50 consecutive sections that were without major errors and exhibiting the full size to test the effects of downsampling on image similarities. The images from Kasthuri compensated what Bock was missing: The data are clean and complete over all 1850 sections. However, the sections only have a size of around 5000^2 px.

2.1.2 Templier220

Apart from the data from Open Connectome, we gathered section images with the new method developed in the lab. A subpatch can be seen in Figure 2.1. The whole images have dimensions of around $32'000^2$px. They were therefore too big to be processed in one run. Instead, each section image was kept in a trakEM 2.2 project as a composition of separate tiles. The single tiles were stitched together using affine transformations.

2.2 The FIJI Environment

In this thesis, we had an enormous amount of image tasks to solve. This includes image loading, alignment, geometric transformations of images, edge enhancements, the computation of feature points or NCC. This is only possible with a sophisticated environment that is specialised on such tasks and that lets its user solve them quickly. For this thesis, we used the imageJ [26] distribution FIJI [25]. ImageJ is an open source image processing program for multidimensional image data. Its primary focus lies on scientific image data. The distribution FIJI offers an advanced API for different high level languages, as well as many user written plugins. The plugin we used the most was trakEM [5] along with the excellent libraries for image similarity and transformations [24, 27, 23, 22]. Together, these software packages allowed for convenient image handling, stacking, and alignment, and made the development of this thesis possible in the first place.

2.3 Transformations

Image transformations are a set of rules to transform one image into another. There are different types of transformations, depending on the properties that are changed and the ones that are kept intact. Euclidean transformations are a combination of translations and rotations, angles and distances between points are preserved. The affine transformations additionally include scaling and shear mapping. We expect the reader to already know the basics and since we will not go above those in this thesis, we omit further explanations. An in-depths introduction of all relevant concepts can be found in Gonzalez [10].

We use image transformations for different purposes during our pipeline. First, the stitching of the mosaic tiles is realized using affine transformations for each tile. Second, we use euclidean transformations for the image alignment. Third, as part of the mean displacement error measurement, which is explained in Section 2.4.1.

2.4 Image Similarity

As already discussed, we were repeatedly confronted with the task of defining image similarities. Furthermore, different stages in our pipeline required different methods to define such similarities. The following sections present the different methods used in our pipeline.

2.4.1 SIFT

Scale Invariant Feature Transform (SIFT) is a method to detect and match local features in images. It was first described by Lowe, D. in 1999 [19]. The algorithm has since proved itself to be one of the best feature detection algorithms, used for many different tasks such as object recognition, image stitching, video tracking or face recognition.

We used SIFT features to assign a dissimilarity value for any pair of section images. Given two section images **A** and **B**, we calculate the mean displacement error of SIFT features (MDE) as follows:

(I) Calculate SIFT feature points for **A** and **B**. We call those sets of feature points F_A and F_B.
(II) Based on the feature points, calculate a rigid transformation $Tr_{A,B}$ to transform F_A into F_B.
(III) If a model is found: filter the sets A and B to identify corresponding points and remove outliers using RANSAC [8]. This leads to the modified feature sets F'_A and F'_B that contain only corresponding points. If no model is found, we assume the images **A** and **B** to be dissimilar, and set MDE to a fixed, high value to reflect this assumption.
(IV) Apply $Tr_{A,B}$ to F'_A and calculate for each point the euclidean distance to its corresponding point in F'_B. Call the set of all distances $d_{A,B}$
(V) Take the arithmetic mean over all distances $d_{A,B}$. Call this value MDE.

In our code, we used the FIJI implementation of SIFT, which was written by Saalfeld et al. and is freely available on GitHub [24]. We used SIFT mostly on downsampled images. The reasons for this are explained in Chapter 3.1.

We needed to redefine the parameters for different downsampling factors, different patch sizes as well as different datasets. As a general rule of thumb, one needs several hundred feature points on both images before trying to find a transformation, and there should remain around 50 feature points in the final sets F'_A and F'_B after removing outlieres with RANSAC. If there are less features preserved, the estimated transformation gets less accurate. In the worst case, one might even miss a possible transformation.

Notice that MDE is a dissimilarity measurement: two sections are said to be similar if the MDE is small, and dissimilar if the value is large. Similarity and dissimilarity can be converted into each other by using a monotonous decreasing function. For our experiments, we used the function $f = \frac{1}{x}$.

The idea behind the usage of MDE is that the point sets of two consecutive images should be similar, and we should therefore be able to transform the two sets into each other. On the other hand, if the sections are too dissimilar, we expect the SIFT points to be distributed dissimilarly as well. As a result, no model should be found, or RANSAC should be unable to find any corresponding points.

2.4.2 Normalized Cross Correlation

Normalized Cross Correlation (NCC) is a well known algorithm in computer science that is used for image similarity estimations for many years. We will not explain it here, but rather point the interested reader to Gonzalez [9], which explains NCC in a detailed manner.

NCC is a direct and convenient way to measure image similarity. We used it on high resolution images up to a size of 2000^2 px. As with SIFT, we used the FIJI implementation of NCC. It was developed by Schindelin et al. and is available on GitHub [27].

2.4.3 Hierarchical Image Similarity

Both methods presented so far, MDE of SIFT features as well as NCC are unable to compute an accurate enough similarity measurement for big images such as the section images from our Templier220 dataset. The images are too big to just compute NCC on them, and need to be downsampled to such an extent to compute SIFT features that too much information is lost to accurately compute similarities. This section presents the final workflow we developed in this thesis to overcome those problems. It is designed to be able to create the similarity matrix for the Templier220 dataset and other large-scale, non-aligned datasets. If the section images are already aligned, or if they are small enough to be processed as a whole, only a subsection of those steps needs to be done.

Pipeline Summary

We can summarize our pipeline in the following manner:

(I) Downsample images
(II) Compute MDE on each pair of downsampled images
(III) Use the found transformations to align possible neighbours at high resolution
(IV) Take high resolution subpatches of all possible neighbours
(V) If needed, align high resolution subpatches further
(VI) Compute similarities of high resolution subpatches with NCC
(VII) Solve the TSP on the obtained similarity matrix

We will now explain each step in detail, except for the last one, which gets covered in the separate Section 2.5.

Step One: Downsampling

As already discussed, the starting point of this thesis are large images, built up from smaller subimages. The basic idea is to downsample the images and first create a rough estimate of possible neighbouring candidates, and then to examine those candidates closer to find the true consecutive sections among them. To distinguish the original images and the downsampled ones, we will from now on call them high resolution images and low resolution images.

Step Two: Computing MDE

Once the images are downsampled, we compute the MDE 2.4.1 for all pairs of low resolution images. This leaves us, for each section, with a list of candidates for neighbouring sections, as well as euclidean transformations for all those candidates.

At this point, we have to decide whether we want to compute a similarity or a dissimilarity matrix. We opt for the former and therefore set all values of image pairs for which we could not find a transformation to a fixed value close to zero. Notice that we should avoid setting the similarity to exactly zero since we might want to transform it to a dissimilarity value at one point using the function $f = \frac{1}{x}$, as explained in Section 2.4.1.

Step Three: Aligning

To closer examine possible neighbours, we have to go back to the high resolution images, taking subpatches and compare them. This is harder than it first seems, since we are still dealing with non-aligned images, as already explained in Section 1.5.2. We can, however, use the transformations we calculated for MDE and apply it to the high resolution images. Since these transformations were calculated based on SIFT features points, which themselves were based only on downsampled versions of the section images, we do not expect the alignment

to be perfect. However, this first alignment is good enough to take subpatches that roughly show the same image contents.

Step Four: Take Subpatches

After aligning all possible neighbours, we are able to take high resolution subpatches. One important thing to note is the way the alignment and the followed collection of subpatches should be done. Consider a given section image **A**. Suppose further that, using MDE, we found three possible neighbours to **A** which we call **B**, **C** and **D**. Additionally, we found the euclidean transformations $Tr_{A,B}$, $Tr_{A,C}$ and $Tr_{A,D}$. We have to make sure to always transform **B**, **C** and **D** to **A**, but not A to the other images. This way, we avoid taking unnecessarily many subpatches since we only need one patch from **A**. At the end, we are left with just four subpatches instead of six. We called this the Master-Slave system. When applied to the whole image stack, it means that one section image should never be transformed, but always be used as master. On the other hand, one section image will always be a slave, so different high resolution subpatches will be taken for this section every time it is a neighbouring candidate. All other section images are somewhere between those two extremes.

Step Five: Further Alignment

As already mentioned in step 2.4.3, we cannot expect the alignment to be perfect, since we used highly downsampled images as source. It therefore might be needed to further align the high resolution subpatches we took in the previous step. Whether this is indeed needed is hard to predict and depends on the exact NCC implementation one uses as well as the amount the original image had to be downsampled to be processed. The subpatches should probably be small enough to use traditional alignment methods. If they are still too big, a repetition of the previous steps of downsampling the subpatches, MDE calculation, applying the transformations and finally taking subsubpatches should be enough.

Step Six: Compute NCC

Once the high resolution subpatches are aligned enough, we can now finally compare them to get an accurate estimation of section similarities. This is done using NCC for reasons explained in Chapter 3.1. This leaves us with high accuracy similarity values for each pair of possible neighbours.

At this point we have computed the whole similarity matrix. We either set the similarity values between two images to a value close to zero if no transformation was found in step 2.4.3, or we use the NCC similarity. We can now proceed with SOR using graph optimization algorithms.

Alternatives

As described, we need image comparisons and similarity calculations at two points during our pipeline: to get a rough overview over possible neighbours, and then again to get a final similarity matrix from high resolution subpatches.

It seems tempting to avoid the second part and just try to find an accurate enough similarity matrix in one step, just using the downsampled images. Maybe one can choose the downsampling factor, the SIFT parameters, and the parameters of how to choose corresponding feature points on the two images in such a fashion that a perfect order retrieval is possible. However, we were not able to find such a combination of parameters, and all results we got were so far away from the right solution that we abandoned this idea.

2.5 Path Optimization

Once the similarity matrix is computed, we can solve for the optimal path that maximizes similarity between consecutive sections. As already introduced in Section 1.3, the similarities between section images can be interpreted as distances between sections. This allows us to state the problem of SOR as a graph optimization problem, in which finding the correct order of all sections is equivalent to finding the shortest path in a weighted graph.

One point to note is the difference between open and closed TSP. In the classical TSP, the salesman has to return to the city in which his tour started. This is called the closed TSP. Returning to the initial city is not needed for our sections: the last section of our stack has nothing to do with the first section, and therefore should not be similar to it or be connected with it in any way. This is called open TSP, the salesman does not return to its starting point. It is easy to transform any closed TSP into an open one by introducing a dummy point which has a distance of zero to any other node. The dummy point can be visited at any point of the tour, and does not increase the tour length. Solving the closed TSP on the now $N+1$ points, one being a dummy point, also gives the solution to the open TSP on N points. One can just remove the dummy point from the final tour once the optimal tour over all $N+1$ points is calculated. It is therefore sufficient to just consider closed TSP in the following sections.

2.5.1 Brute Force

The naive idea is to just try all possible paths. Unfortunately, the salesman can, in each step, choose from all the cities that he did not already visit, leading to $(N-1)!$ possible paths for N cities. As explained earlier 1.1, our similarity values are symmetrical. Therefore, it does not matter whether a path is taken forwards or backwards. This leads to the total number of $\frac{(N-1)!}{2}$ possible paths. But even with this restriction, the number of possible solutions explodes for even small N: For only 15 sections, we would have to compute $43'589'145'600$ different paths. This method is therefore not an option for several thousands of sections.

2.5.2 Greedy Approach

The idea behind this algorithm is simple: randomly choose a starting point from all sections. Then always go to the nearest section that was not visited yet until all sections were visited. The approach is computationally fast and easy to implement, but as shown by Gutin et al. [12], it can also lead to the worst possible path. Which outcome one gets is completely dependant on the structure of the data one runs the algorithm on, as well as which starting point was selected at the beginning. For our results with the method, see Chapter 3.2.3.

2.5.3 openOPT

OpenOPT is an open source solver for various optimization problems, among which is also a TSP solver. It was developed by Dmitrey Kroshko in Python and is available on the openOPT website [16]. Since the rest of our code was written in Python as well, openOPT was an interesting possible choice to keep the number of different required software packages small.

2.5.4 Concorde

Concorde is a highly tuned software package to solve TSP. Developed around the year 2000, it is written in ANSI C and available as source code as well as pre-compiled on the Concorde website [3]. Despite its age, it is one of the best solvers available [14]. Different TSP instances can be fed to the Concorde solver via a special data format, called TSPLIB [21]. This file can easily be created once the similarity matrix is computed, since it is just a container for the distance values.

2.5.5 Defining An Error Measurement

When testing the methods of image similarity, followed by the solution of the problem by a TSP solver, we were confronted with the problem of quantitatively measuring the quality of the found order. We can identify the following outcomes for our attempt to retrieve the order of the sections:

- The retrieval is perfect.
- There are single swaps either in-place or over bigger distances. The majority is ordered correctly.
- The sections are unordered. The retrieval failed.

Below are some examples of such errors. The numbers in the brackets represent the true positions of the sections inside the stack.

(I)	$[0, 1, 2, 3, 4, 5, 6]$	The retrieval is perfect.
(II)	$[0, 1, 3, 2, 4, 5, 6]$	There is a single in-place swap of sections 2 and 3.
(III)	$[0, 1, 5, 4, 3, 2, 6]$	There is a single swap of sections 2 and 5. It is important to note that the reversal of the order is not an actual error. The sections 3 and 4 are ordered correctly since both their neighbours are correct.
(IV)	$[0, 3, 5, 4, 2, 1, 6]$	The sections are unordered. SOR failed.

An error measurement should address some key points. First, if the order retrieval is perfect, the error should be zero. Second, the error should be higher for example (III) than for example (II) for multiple reasons. First of all, sections 1,2, and 3 might actually look similar, and it really might be a close decision where to put which. This should not be the case for sections further apart. Therefore, we should punish such errors harder than simple in-place swaps. Second, for the later use of our section stack data in research, errors of type (III) are more severe than the ones from example (II). The SOR in example (IV) is completely worthless and should not occur under any circumstances. If it does, the error should be higher than for any remotely correct solution.

As a first step towards an error measurement, let us define the distance d between two sections i and k as the distance they are away from each other in the true, ordered stack:

$$d_{i,k} := |p_i - p_k| - 1 \tag{2.1}$$

p_i and p_k being the position indices of sections i and k in the true image stack. Now we can define the retrieval error E of an retrieved image stack as the sum of all distances between all N consecutive sections:

$$E := \sum_{i=0}^{N-1} d_{i,i+1} \tag{2.2}$$

where i is the new position of a section in the retrieved image stack. For the four examples given above, this leads to errors of 0, 2, 6 and 8, respectively, which matches nicely all the requirements we formulated.

Chapter 3

Results

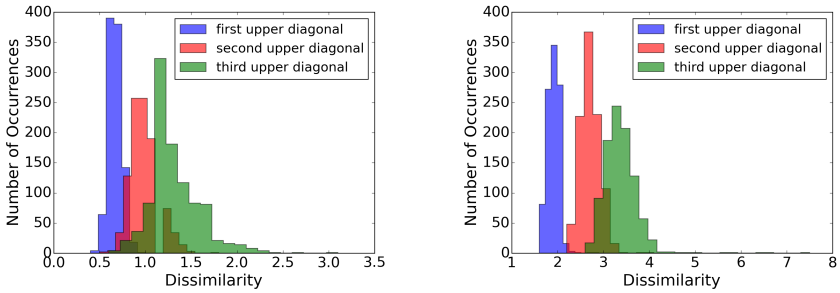
From the many questions we had at the beginning of this thesis 1.6, we were able to answer most in a satisfactory manner, and we want to present those answers in the following chapter.

3.1 SIFT or NCC

In principle, it is possible to create the similarity matrix with either NCC or MDE of SIFT features. We found for both methods key advantages, but also disadvantages. They are summarized in the following points.

3.1.1 Structure of biomedical image data

The data we are dealing with contains a lot of background noise in the form of subcellular neural structures. The big challenge however is that we need exactly those fine changes in the background to correctly retrieve the order of all sections, since big, global structures such as blood vessels change too slowly from one section to the next. SIFT was not developed for this purpose, but rather to be robust against noise and to recognise global structures. The fact that the images first get blurred before features are computed is another hint that SIFT is not intended to characterize fine details on images. This leads to SIFT having problems giving a fine enough similarity measurement for close sections, as shown in Figure 3.1. It shows two similarity distributions from two similarity matrices, both derived from the Kasthuri dataset. For one, MDE was used, for the other NCC. There is a clear gap between the values of the first subdiagonal and the ones from the second when using NCC, indicating that the two cases can be distinguished clearly. This gap is not present when using MDE, indicating that the method lacks capability to distinguish between close sections. In the end, this suggests to not use MDE for the final creation of the similarity matrix, but to rather rely on NCC.



(a) Distribution of MDE using SIFT.

(b) Distribution of NCC.

Figure 3.1: These are the similarity distributions of two full similarity matrices of 1000 sections with a size of 2000^2 px. (a) was created using SIFT features to compute MDE, for (b), NCC was used. One sees the difficulty of MDE to detect the exact differences between close sections. This is done a lot better by NCC, shown by the clear distinction between the values of the first upper diagonal and the ones from the second.

3.1.2 Computational overhead

NCC is a simple and well defined method that can be used with a few lines of code. MDE, on the other hand, requires to compute SIFT features first, which themselves have a high number of parameters that need to be fine tuned depending on the data one is working on. Further, the right type of transformation has to be determined, and RANSAC needs to find enough inliers to produce a meaningful mean value afterwards. All those parameters are unknown a priori and depend on the exact data one works with, and wrongly assigned values often lead to diffuse errors whose origin is hard to localize. This does not mean that MDE is a bad measurement value, but its parameter tuning should not be neglected. This suggests that one uses NCC whenever possible.

3.1.3 Aligning Images

The biggest drawback of NCC is that it requires images that are already aligned. If this is not the case, we need to use MDE. Even more, MDE serves double duty in that it not only identifies possible neighbours based on global appearance, but also automatically delivers a transformation to align those candidates.

3.1.4 Conclusion

To summarize Sections 3.1.1, 3.1.2 and 3.1.3, we can say that we should use NCC whenever possible. However, as soon as we are confronted with non-aligned images, we need to use MDE, since it is our only tool to handle alignment when we are simultaneously confronted with missing information about an image stack's true ordering.

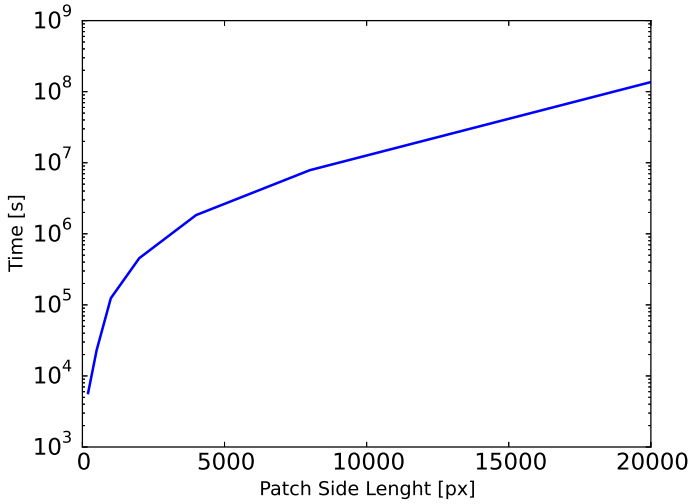


Figure 3.2: Shown is the time needed to compute a full similarity matrix of 1000 sections, depending on the size of the images. For patch sizes up to 4000^2 px, the Kasthuri dataset was used. Above that size, Bock11 was used. The runtime presented in this plot for patch sizes bigger than 2000^2 px was obtained by extrapolating the time needed to compute NCC on only a few images.

3.2 Proof of Concept

A first step that needed to be done is to show that a fully automated SOR is doable at all. For this, we used pre-aligned image stacks from the Kasthuri and Bock11 datasets 2.1.1. Since we knew the order of those stacks, it was easy to test the basic parameters, as well as finding a good graph optimization method. As explained in Section 3.1, we exclusively used NCC for those experiments.

3.2.1 Runtime

The computationally most expensive step is the creation of the similarity matrix. For a full similarity matrix, one has to compare each section to each other, leading to total of $\frac{N*(N-1)}{2}$ comparisons, where N is the total number of sections. For 1000 sections we have to compute $500'000$ similarity values. It was therefore crucial to know how long it takes to compute the matrix. As can be seen in the Figure 3.2, the computational time needed to compute NCC soon increases over acceptable levels with increasing patch sizes. Patches of the size of 200^2 and 500^2 px can be processed in a a few hours without trying to optimize the algorithm used in Fiji. 1000^2 px need a bit more than a day to finish. A practical limit was reached with NCC on 2000^2 px subpatches, as the computation took 5 days to finish on a Windows 7 machine, using a Intel Core i7 CPU with 8 Cores at 3.07 GHz each and 18GB RAM. The time needed for 2000^2 px subpatches was reduced to less than one day by using a high end machine with 48 cores and

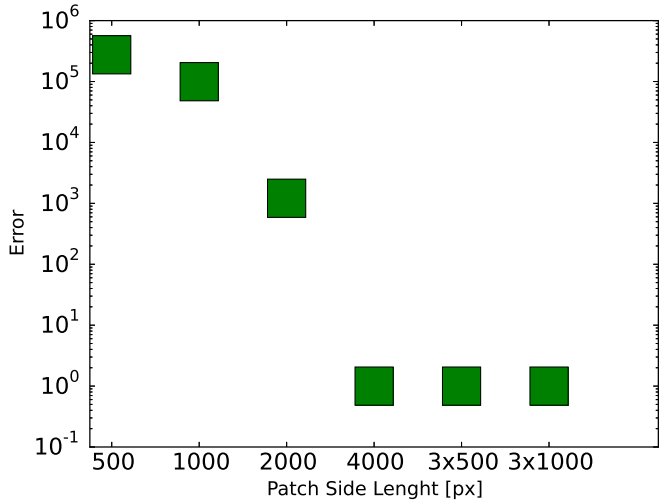


Figure 3.3: This figure shows the dependency of patch size to retrieval accuracy. Concerning the results for averaged patches, we placed the subpatches on a diagonal over a bigger patch from the Kasthuri or Bock11 dataset. We did, however, not notice any significant differences when we placed the subpatches differently, e.g. in the corners of the bigger patch. Finally, we would like to point out that we increased the error value for a perfect SOR to be 1 instead of 0 to be able to use a logarithmic scale in this plot.

256GB of RAM. But even with this computational power, patch sizes above this limit proved to be impracticable to solve. As a result, we do not recommend using patch sizes bigger than 2000^2 px, unless significantly bigger computational capacities are freely available. This leads to a de facto upper boundary for patch sizes of 2000^2 px.

3.2.2 Smallest Possible Patch Size

As shown and explained in Section 3.2.1, the maximum patch size that can reasonably be processed is around 2000^2 px. First of all, we still need to show that SOR works in the first place. Additionally, we are missing a lower boundary: how small can a patch be so that we can still retrieve the full section stack correctly ? The intuition is straight forward: the smaller the patch gets, the more likely it is that two non-consecutive patches randomly look similar, violating thereby our fundamental assumption that only consecutive sections are similar. The results of our tests can be seen in Figure 3.3. Please not that the last step, the graph optimization, will be explained in Section 3.2.3.

Figure 3.3 shows that a perfect SOR over 1000 sections is indeed possible and constitutes the proof of concept validating our approach. Additionally, it shows that we need to do either one of two things to successfully retrieve the section order:

- Use a patch size of 4000^2 px or bigger
- Average the results of multiple patches

Since the use of patches bigger than 2000^2 px is strongly discouraged because of too big computational costs, only the second point remains. One of the smaller advantages of NCC is that it can easily be done on multiple subpatches. The similarity results of all subpatches can then be averaged to get a much better result than if only one big patch was used. This way, one can go down to $3*500^2$ px, which represents the lower boundary for patch sizes.

3.2.3 Path Optimization

Finally, we had to determine what strategy to use to solve TSP. As already discussed in Section 2.5, we have the possibility to either use brute force, a greedy algorithm that always visits the next nearest section, or to use a sophisticated TSP solver like openOPT or Concorde.

Brute force got immediately dropped since the number of possibilities grows over-exponentially, and it would be a waste of time and resources. This leaves us with the other three possibilities, which we all tested.

Greedy Algorithm

We started with our own implementation of a greedy algorithm. The idea is simple: randomly choose a starting point from all sections. Then always go to the nearest section that was not visited yet until all sections were visited. Figure 3.4 shows the solution when using this method on the 1000 section NCC similarity matrix from the Kasthuri dataset with patch sizes of 2000^2 px.

As can be seen from the graph, the retrieval was not good enough. The reason for this gets apparent when we examine the plot closer. The greedy algorithm did a good job at the beginning, where it more or less got the order of all sections correctly. However, once in a while, one or a few sections were missed. And here is where the problem lies: those sections need to be visited at the end of the tour, where they cause big errors. Because of that, the last 50 section positions returned from the algorithm are basically worthless, as they are just a collection of all sections that got missed during the tour. This is exactly the reason why TSP is such a hard problem: decisions made during the tour do not have negative effects until the end. Even more: It could be necessary to make locally non-optimal decisions whose positive effects are only visible at the end of the tour.

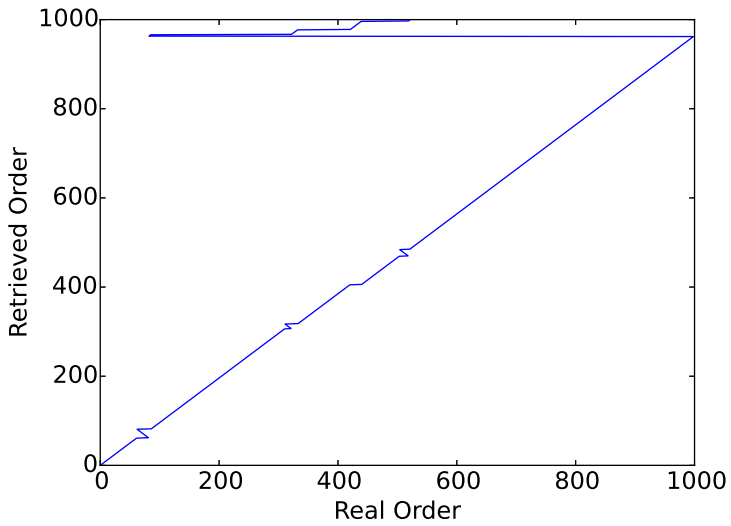


Figure 3.4: Test of the greedy approach to solve TSP on a set of 1000 sections from the Kasthuri dataset. Plotted is the order found by a greedy algorithm vs. the true order of the sections. If the order retrieval was perfect, the graph would show a straight line. We can see one of the biggest disadvantages in using this approach: missed sections have to be visited at the end of the tour, leading to big errors at the end.

This result was obtained using clean data, big patches, and the best possible alignment. Since we are not able to correctly retrieve the order of such a similarity matrix, chances are low we ever manage to do it on a harder dataset. For this reason, we discarded this approach and concentrated on sophisticated TSP solving implementations.

openOPT or Concorde

In our search for a dedicated TSP solver we found and tested two open source implementations: openOPT and Concorde. Both were already described in Sections 2.5.3 and 2.5.4. Unfortunately, both implementations proved to be difficult to use. We will not go into the details at this point, since they are not relevant, but refer the interested reader to the Appendix .1 and .2, where we explain the practical difficulties we had with Concorde and especially openOPT.

There was one major criterion, however, that seperated openOPT and Concorde from each other: speed. Concorde has outperformed openOPT by an order of magnitudes. Solving a set of 1000 sections took openOPT several hours. Concorde, on the other hand, often solved the same problem instance in fewer than 30 seconds, often even as few as 5 seconds. All other requirements Concorde had, e.g only being able to solve the problem when presented as TSPLIB-file, got nearly irrelevant in the face of such a performance difference. This was the most important point that led to our decision to use Concorde as our TSP solver of choice.

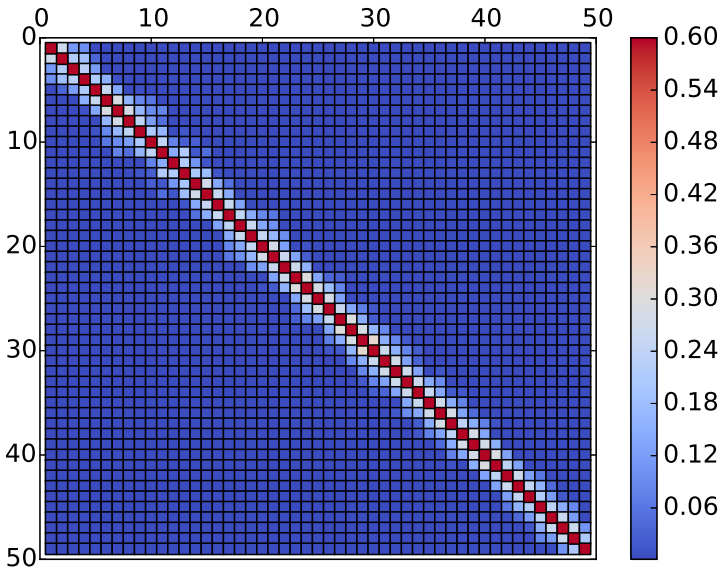


Figure 3.5: This is the similarity matrix of a 50 sections dataset from Bock11. Each section had a size of $20'000^2$px, which allowed us to test our pipeline from start to end on a known dataset. The recreation using Concorde was perfect, however, we observe an unwanted high number of pre-set similarity values, which is especially bad near the main diagonal.

Finally, it is important to note that our data has a special structure, that makes solving a TSP instance of it a lot easier than a randomly distributed set of 1000 cities. Our sections only have a few close neighbours, all other sections are far away. Our data therefore has, when imagined on a 2d plain, the structure of a line. To find the shortest path in such a situation is a lot easier than in a general case, which might be the reason Concorde was able to solve our problem instances so fast. We did not, however, test Concorde on other, more general TSP instances.

3.3 Testing the Pipeline

After showing that SOR in principle works, the next step is to retrieve the order of the Templier220 dataset using the pipeline we described in Section 2.4.3.

3.3.1 Test on Bock11

As a first experiment to test the pipeline, we took 50 sections from the Bock11 dataset, each section having a size of $20'000^2$ px. That way, we were able to test the whole pipeline including the downsampling on a known dataset. Since the data was already aligned, we expected the found transformations to not change

the images in any way, e.g. the transformations should be similar to the identity matrix. If they were not, the subpatches would actually become non-aligned.

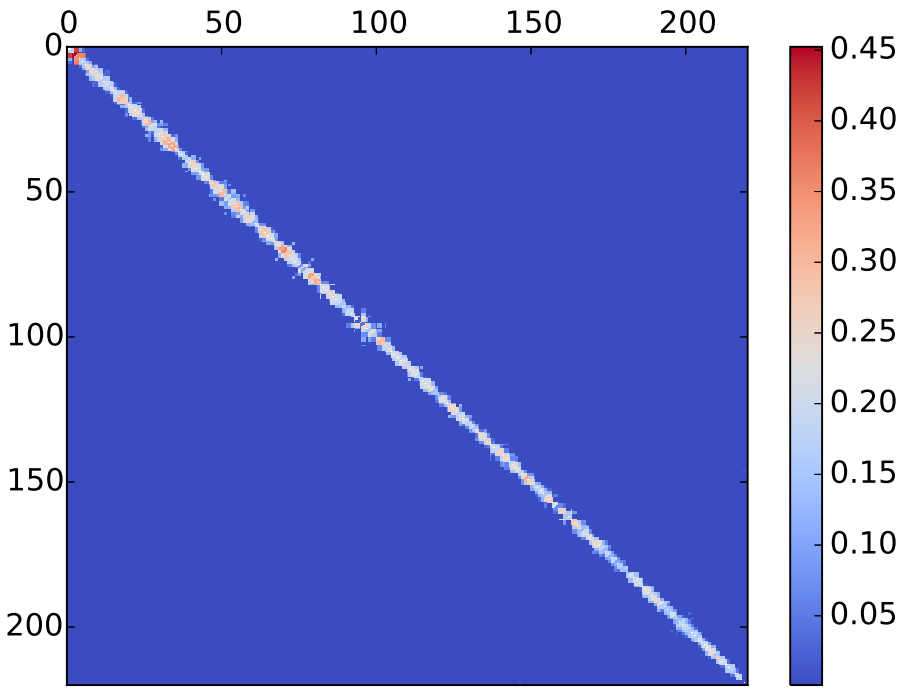
The result was again promising. The recreation was successful and was done without any error. Also, as predicted, the transformations were solely used to identify possible neighbouring sections, but the images were not changed in the aligning step. Figure 3.5 shows the similarity matrix obtained in this process. One thing to note about this similarity matrix is that most entries are fixed, low values. Remember that, if we do not find a transformation for a given pair of section images, the similarity value of this pair is set to a low value, indicating that we are sure that they are not neighbours. The high number of such values in our similarity matrix comes from the fact that we are trying to find transformations twice, the first time for the downsampled images, and a second time when we refine the alignment of the high resolution patches. The number of sections for which we do not find any transformation adds up, leaving us with only a small list of sections that we actually need to consider when trying to retrieve the order.

At first, this seems like a good property, since it creates the impression of an accurate and precise method. But in fact, quite the opposite is the case: it creates the dangerous possibility of not having a single possible neighbouring candidate for a given section. This would lead to severe errors, since we do no longer have the possibility of "compensating" such errors by single in-place swaps of sections. In other words, such a similarity matrix either returns the perfect result, or results that are absolutely wrong. Additionally, for reasons already explained in Section 3.1, we do not want to use SIFT to measure the exact similarity of neighbouring sections, but rather use NCC.

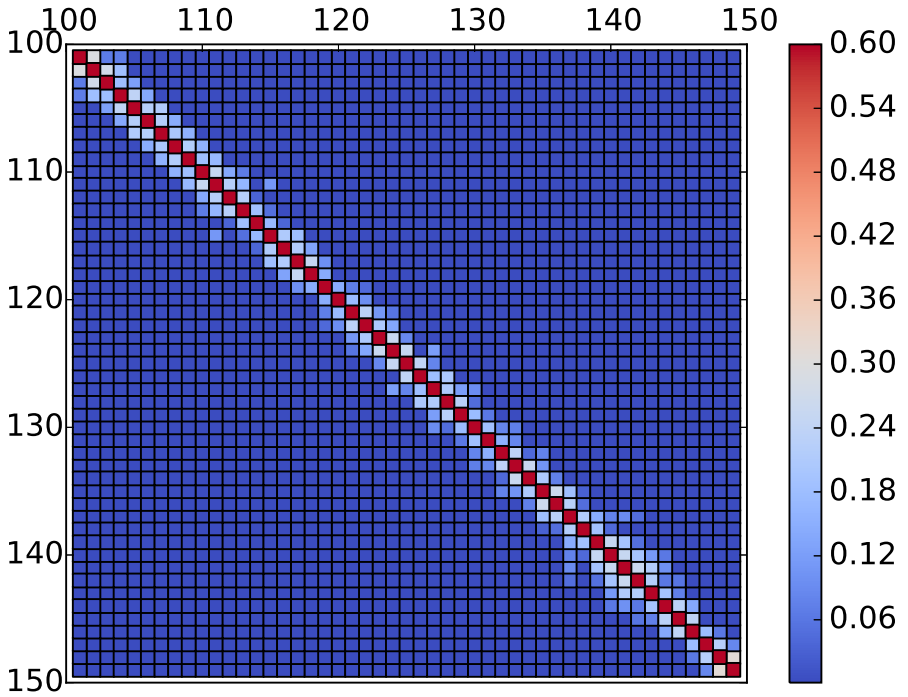
3.3.2 SOR of the Templier220 dataset

Using the same methods as for the Bock11 dataset, we can finally apply our pipeline to our own dataset. The resulting similarity matrices can be seen in Figure 3.6. The results were again positive and promising. When we arranged the sections the way our pipeline suggested, the majority of the 220 sections seemed to fit perfectly. There were, however, around 10 swaps, which correspond to the error types 2.5.5 (II) and (III). Additionally, we can again observe the unwanted phenomenon of too many fixed low values, even close to the main diagonal. This is not something we want to see, as we want to use NCC as our method to decide the exact similarity of close sections.

Nonetheless this similarity matrix is a good hint that the recreation was mostly successful, apart from our subjective impression. The structures of both the Templier220 and the Bock11 similarity matrices are nearly identical, suggesting that the underlying SOR was at least nearly as successful as it was on the Bock11 dataset.



(a)



(b)

Figure 3.6: Figure (a) shows the complete similarity matrix for the Templier220 dataset, Figure (b) shows a subset of all similarity values to closer examine the properties near the main diagonal. As it was already observed in the similarity matrix of the Bock11 dataset 3.5, we have a high number of pre-set, low similarity values close to the main diagonal, which is not optimal. Apart from that, the data is exactly how we want it to be.

Chapter 4

Discussion

During the last chapters we presented our pipeline, while highlighting the decisions and observations that led to its final form. The last part of this thesis summarizes our results and names problems that are still unsolved.

4.1 Conclusion

The goal of this thesis was to introduce and examine the tools needed to retrieve the order of section images. This was successful, as we could show how to reliably retrieve the order of up to 1000 sections without any error. Further, we hoped to be able to create a pipeline to also retrieve the order of big, non-aligned section stacks that we created ourselves in the Hahnloser Laboratory. The observations in 3.3 with our pipeline strongly suggest that our final pipeline leads to a good retrieval result. Furthermore, our personal inspection of the suggested result led us to the same conclusion. We therefore conclude that the second goal of this thesis, the section order retrieval of the Templier220 dataset, was successful as well.

There were, however, also signs of problems that need to be addressed in the future. First of all, we already noticed that our pipeline led to too few possible neighbouring candidates, increasing thereby the risk of not finding any neighbour at all. We think that this can be solved by relaxing the SIFT parameters, as well as the parameters used by RANSAC to find inlier among the SIFT feature points. This should result in more found transformations. The downside, of course, is an increased computational cost. We think, however, that this will increase the SOR performance and is therefore worth this price.

Finally, we need to point out that we assume that our images are free from artefacts. Those artefacts however, are present on the images and can arise from both the section collecting process as well as the imaging process afterwards. It is therefore important to make the method robust against those errors in the future.

4.2 Future Work

The first thing that needs to be done from now on is a further validation of our impression that the SOR of the Templier220 dataset was indeed a success.

Once this is done, we can answer the few questions 1.6 that still remain unanswered. In particular, these were the questions about the minimal resolution, or, the other way around, how much one can downsample the images, so that MDE still produces correct results.

Finally, we need to address the limitations that were already explained in Section 4.1. First of all, this means relaxing the SIFT parameters to find slightly more neighbouring candidates. Second, this means making the method more robust against the different types of image errors.

.1 Concorde Platform Dependancies

Although Concorde is one of the best TSP solvers available, it's code was not updated in many years. As a result, the last operating systems Concorde is guaranteed to run on is Windows XP and Red Hat Linux 8.0, which was released in 2002.

We encountered problems running the pre-compiled Concorde executables on newer Windows platforms, as the software finished instantly and returned no results at all. The attempt to compile Concorde from source failed as well. As a work around, we ran Concorde on a Linux MINT Virtual Machine, using VirtualBox as emulator software. We transferred each TSPLIB-file onto the VM, solved it there, and transferred the solution back onto the host machine. This is of course as cumbersome as it gets, but after writing a script that did all this automatically, we at least had a running solution that was efficient enough to work with. There exists a second workaround in the form of the Neos Concorde Server [11, 6, 7] which is available online [2].

.2 Fate of openOPT

At the beginning of this thesis in the summer of 2015, the website of openOPT [16] was available without limitations. We tested the software, and finally decided against its usage. Apart from the lacking speed, which was already described in Section 3.2.3 and which was our main reason to drop it, it became more and more apparent that openOPT itself relied heavily on external code and libraries. For this reason, a direct usage of it as embedded part in our own code was hard to do, which initially was our motivation to look into a Python implementation. Additionally, in the summer of 2015, openOPT was only sparsely documented and some utility was still under development.

During the finalisation of this thesis in the winter of 2015, the openOPT website stopped being accessible. We have no knowledge about the reasons for this, but wanted to point out that the code itself is still available on the Python Package Index [17]. However, large parts of the documentation are not accessible any longer.

Bibliography

[1] Open connectome project. `http://www.openconnectomeproject.org/`. Accessed: 2015-11-28.

[2] Neos concorde server. `http://www.neos-server.org/neos/solvers/co:concorde/TSP.html`, 2015. Accessed: 2015-12-28.

[3] David Applegate, Robert E. Bixby, Vasek Chvátal, and William J. Cook. Concorde. `http://www.math.uwaterloo.ca/tsp/concorde.html`. Accessed: 2015-12-20.

[4] Davi D Bock, Wei-Chung Allen Lee, Aaron M Kerlin, Mark L Andermann, Greg Hood, Arthur W Wetzel, Sergey Yurgenson, Edward R Soucy, Hyon Suk Kim, and R Clay Reid. Network anatomy and in vivo physiology of visual cortical neurons. *Nature*, 471(7337):177–182, 2011.

[5] Albert Cardona, Stephan Saalfeld, Johannes Schindelin, Ignacio Arganda-Carreras, Stephan Preibisch, Mark Longair, Pavel Tomancak, Volker Hartenstein, and Rodney J Douglas. Trakem2 software for neural circuit reconstruction. *PloS one*, 7(6):e38011, 2012.

[6] Joseph Czyzyk, Michael P Mesnier, and Jorge J Moré. The neos server. *Computing in Science & Engineering*, (3):68–75, 1998.

[7] Elizabeth D Dolan. Neos server 4.0 administrative guide. *arXiv preprint cs/0107034*, 2001.

[8] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[9] Rafael C. Gonzalez and Richard E. Woods. 12.2.1 matching. In *Digital Image Processing*, chapter 12.2 Recognition Based on Decision-Theoretic Methods, pages 866 – 903. Prentice Hall, 3 edition, 2007.

[10] Rafael C. Gonzalez and Richard E. Woods. 2.6.5 spatial operations. In *Digital Image Processing*, chapter 2.6 - An Introduction to the Mathematical Tools Used in Digital Image Processing, pages 72 – 98. Prentice Hall, 3 edition, 2007.

[11] William Gropp and Jorge Moré. Optimization environments and the neos server. *Approximation theory and optimization*, pages 167–182, 1997.

[12] Gregory Gutin, Anders Yeo, and Alexey Zverovich. Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the tsp. *Discrete Applied Mathematics*, 117(1):81–86, 2002.

[13] Philipp Hanslovsky, John A. Bogovic, and Stephan Saalfeld. Post-acquisition image based compensation for thickness variation in microscopy section series. *CoRR*, abs/1411.6970, 2014.

[14] Kurt Hornik and Bettina Gruen. Tsp-infrastructure for the traveling salesperson problem. *Journal of Statistical Software*, 23(2):1–21, 2007.

[15] Narayanan Kasthuri, Kenneth Jeffrey Hayworth, Daniel Raimund Berger, Richard Lee Schalek, José Angel Conchello, Seymour Knowles-Barley, Dongil Lee, Amelio Vázquez-Reina, Verena Kaynig, Thouis Raymond Jones, et al. Saturated reconstruction of a volume of neocortex. *Cell*, 162(3):648–661, 2015.

[16] Dmitrey Kroshko. openopt. `http://openopt.org`. Accessed: 2015-08-20.

[17] Dmitrey Kroshko. Python package index. `https://pypi.python.org/pypi/openopt`. Accessed: 2015-12-20.

[18] Gilbert Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.

[19] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[20] C.H. Papadimitriou and D.S. Johnson. Computational complexity. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, chapter 3: Computational complexity, pages 37 – 87. John Wiley & Sons, Chichester, UK, 1985.

[21] Gerhard Reinelt. Tsplib - a travelling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

[22] Stephan Saalfeld, Albert Cardona, Volker Hartenstein, and Pavel Tomančák. As-rigid-as-possible mosaicking and serial section registration of large sstem datasets. *Bioinformatics*, 26(12):i57–i63, 2010.

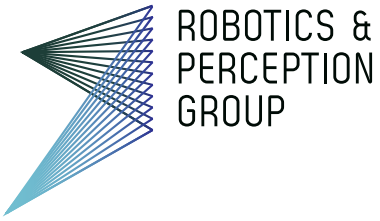
[23] Stephan Saalfeld, Richard Fetter, Albert Cardona, and Pavel Tomancak. Elastic volume reconstruction from series of ultra-thin microscopy sections. *Nature Methods*, 9(7):717–720, 2012.

[24] Stephan Saalfeld, Johannes Schindelin, and Stephan Preibisch. mpicbg. `https://github.com/axtimwalde/mpicbg/`, 2015. Accessed: 2015-12-28.

[25] Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis Rueden, Stephan Saalfeld, Benjamin Schmid, et al. Fiji: an open-source platform for biological-image analysis. *Nature methods*, 9(7):676–682, 2012.

[26] Johannes Schindelin, Curtis T Rueden, Mark C Hiner, and Kevin W Eliceiri. The imagej ecosystem: An open platform for biomedical image analysis. *Molecular reproduction and development*, 82(7-8):518–529, 2015.

[27] Johannes Schindelin, Stephan Saalfeld, Stephan Preibisch, and Curtis Rueden. Imglib1. `https://github.com/fiji/legacy-imglib1`, 2015. Accessed: 2015-12-28.



Title of work:

Automated Order Retrieval Of Ultra Thin Brain Sections

Thesis type and date:

Bachelor Thesis, December 2015

Supervision:

Thomas Templier
Prof. Dr. Richard Hahnloser
Prof. Dr. Davide Scaramuzza

Student:

Name:	Fabian Weiersmueller
E-mail:	fabian.weiersmueller@uzh.ch
Legi-Nr.:	11-717-071

Statement regarding plagiarism:

By signing this statement, I affirm that I have read the information notice on plagiarism, independently produced this paper, and adhered to the general practice of source citation in this subject-area.

Information notice on plagiarism:

`http://www.lehre.uzh.ch/plagiate/20110314_LK_Plagiarism.pdf`

Zurich, 29. 12. 2015: _____