Abstract

St. Olavs hospital has supplied a dataset of 2703 tissue samples from the tumor periphery from approximately 900 patients organized on tissue microarrays (TMA). In this project we wish to examine all these tissue samples with image processing to determine if second harmonic generation microscope images of tissue can improve classification of cancer type (I, II, III) or in other words, cancer aggresiveness. This thesis documents methods which automates the microscope imaging of TMA and show how images can be correlated to clinical data. Datamining methods can then be used on this dataset to look for patterns which can be used in classifcation.

Automated microscope scanning is easy in consept, but the implementation depends on many aspects of the experimental setup. Some of the aspects discussed in this thesis are:

- Develop image analysis algorithms that are robust to experimental variations.
- Handle systematic errors like intensity variation and rotation between scanning raster pattern and stage coordinate system.
- Automatic stitching of regular spaced images with little signal entropy in seams.
- Adjusting z-plane tilt for large area samples with micrometer precision.
- Interfacing with commercial Leica software.

The focus of this thesis is on TMA and the experimental setup with a Leica SP8 microscope, but some the aspects listed above are not unique to this context only.

The conclusions are:

- Large area scans should adjust specimen plane to be at even distance to the objective to be time effective and avoid out of focus images.
- Using heuristics/constraints improves the reliability to automatic stitching algorithms, failing gracefully on images with little entropy in overlap.

- Leica LAS version X 1.1.0.12420 have limited support for automatic microscopy, but it's possible to work around limitations to leverage fully automated TMA-scanning.

Contents

Preface

Five years at NTNU have been a rollercoaster ride. Uphills at times, but also a great deal of fun. I'm grateful for the number of marvellous people I have met, the flexibility the student life brings, all the fun with the student society Spanskrøret and not to forget all the things I've learned.

A special thanks go to all the professors who share their knownledge every day, even at times when their students seems unmotivated.

The last year I have been warmly included in Magnus Borstad Lilledahl's research group, with Andreas Finnøy, Elisabeth Inge Romijn and Rajesh Kumar. It's been educational to work with them and exciting to get an insight in how they perform their work. Anna Bofin and Monica J. Engstrøm at St. Olavs has also been very welcoming, showing me histological patterns in tissues and provided the dataset. Thank you all!

I would also like to thank my family, who always have been supportive for the choices I've made.

Lastly, the greatest thanks go to my life companion Yngvild, it wouldn't have been the same without you.

The future is already here, it's just not very evenly distributed. - William Gibson

Chapter 1

Introduction

With a population in Norway just above 5 million[1], three thousand women are diagnosed with breast cancer each year[2]. This makes breast cancer the most common kind of cancer, affecting one of every eleventh woman. Luckily breast cancer is often not deadly or treatable, shown by the fatalities which was 649 in 2012[2]. Norwegian University of Science and Technology (NTNU) and St. Olavs hospital have been cooperating on reasearch to find new ways to improve diagnosis. The cooperation yielded a study of 37 subjects which showed positive results on difference of collagen structure from different parts of tumor tissue[3]. The goal of this project is to delvelop the necessary tools to expand the study from 37 subjects to the whole dataset of approximately 900 subjects.

The means to achieve the expanded dataset is to automate microscope imaging, with main focus on tissue microarrays. Tissue micro arrays are glass slides with samples arranged in a matrix pattern seen in figure 1.1. As tissue microarrays is a standard way of organizing tissue samples, not unique to breast cancer tissue, this project is relevant for other studies too.

The tissue micro array shown in figure 1.1 is $\approx 24 \times 15$ mm in size. Using a moderate objective of 25x with 400×400 µm field of view, a single scan of the total dataset is

$$\frac{24\text{mm}}{400\mu\text{m}} \cdot \frac{15\text{mm}}{400\mu\text{m}} = 2250 \text{ images.}$$

Depending on the precission of the microscope stage, images are not necessarry easily put together. Also, keeping the microscope in focus for the whole surface can be challenging. Another

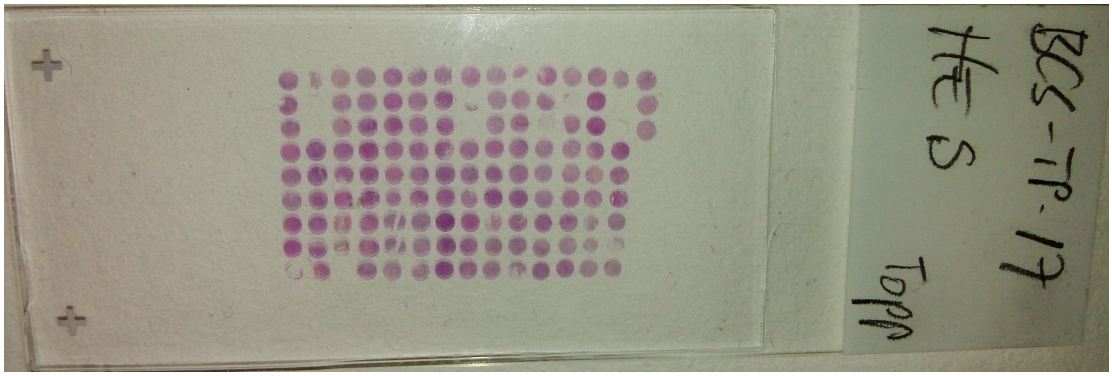


Figure 1.1: Tissue micro array of breast tissue at perifery of tumor. Three test samples (upper right of the array) are beside the 14×9 samples to avoid mix up of patients when rotating the slide.

approach would be not to scan the whole area in one scan, but to scan each of the $14 \cdot 9 = 126$ tissue specimens one by one. The challenge with scanning each region one by one is that the samples are often not equally spaced, and a lot of manual, error prone, labor is required to define the areas to scan. The method in this thesis tries to simplify the scanning process and prepare the images for further analysis.

The thesis is written with focus on two parts, namely automating the collection of images and correlating samples to clinical data. Together, the methods described should enable researchers to run experiments on large datasets of tissue microarrays.

A reader of this text should be familiar with general physics. Matters that are specific to scanning microscopy and image processing will be described in the *theory* section, along with software concepts used. The *methods* section seeks to make the reader able to replicate the experiment on any kind of microscope, but some software and solutions is specific to the Leica SP8. The *discussion* holds details on alternative approaches and should clarify reasons for the choices made. As the project mainly consisted of developing automated microscope scanning, the methods is also the result of the thesis, hence a result chapter is not included.

All source code in the project was implemented in the programming language Python[4]. The reader does not need to be proficient in Python programming, but acquaintance with the syntax is assumed. Code blocks will be used to clarify how problems have been solved or algorithms implemented. Details not essential to the problem at hand have been omitted to keep focus on the essential parts. As the total amount of source code is above thousand lines it's not included in the appendix but rather available at Github with full history[5]. A brief description on installation

of the software is included in the *appendix*.

Chapter 2

Theory

Tissue microarrays

A tissue microarray is a collection of specimen aranged in a matrix pattern. The specimens are typically sliced with microtome from a paraffin block containing cylinders of tissue in rows and columns. Cylinders for the paraffin block are often picked out by a pathologist who evaluate the histology of a larger tissue sample and choose appropriate locations.

The thickness of slices are in the magnitude of 4-5 µm, which gives efficient use of tissue samples in the sense that several hundred TMAs can be made from a block containing cylinders which can be several mm^6. *Specimen spot* will refer to a single sample in the array.

Scanning microscope

Figure 2.1 illustrate the internal workings of a Leica SP8 scanning microscope which have an epi-illumination setup. Epi-illumination is when the detectors (26) and light source (1, 3, 5, 7) are on the same side of the objective (18). But as seen, the epi-setup also allows for transmitted detectors (19), which were the ones in use. By scanning one means that the light source is focused to a specific point of the specimen, and scanned line by line in a raster pattern. While the laser is scanned over the surface, a photomultiplier tube (PMT) measures the incoming light in regular time intervals (samples) and each measured sample is saved to an image pixel.

A *photonmultiplier tube* is a sensor which converts photon intensity into an electrical signal. The tube works by accelerating electrons that have been liberated from an electrode by incomming photons. The flux of electrons is multiplied several times by aranged electrodes inside the tube, resulting in an amplification which makes it possible to measure small amounts of light[7].

The scanning is done by a galvanometric mirror (14). The term *non-descanned detector* indicate that the light does not travel via the scanning mirror before reaching the detector. In SP8, (17) and (19) are non-descanned detectors, where (17) measure reflected light and (19) measure transmitted light. The condensor and aperture is not illustrated in figure 2.1. Both is present between the glass slide and the external non-descanned detector (19). Condensor collects light for the transmission non-descanned detector. The aperture is an adjustable opening which can be used to limit the amount of incomming light. Higher aperture values means more opening.

Field of view is the spatial area which fits inside one image. The field of view depends on the magnification of the objective and the scanner zoom. Scanner zoom is when the scanner is set to oscillate with a smaller amplitude while still sampling at the same rate. As field of view is at the magnitude of 1×10^{-4} m, specimen must be moved around to image a larger area. The device that moves the specimen is called a stage. Here stage position, or specimen position, is denoted with a upper case X to distinguish it from lower case x which denote image pixel position.

The resolution of a conventional light microscope is given by the objective and/or condensor *numerical aperture* (NA)[7]:

$$d = \frac{1.22\lambda}{NA_{condensor} + NA_{objective}}. \tag{2.1}$$

Here d is the minimum separable spatial distance defined by the Rayleigh criterion, λ is the wavelength of the light and NA is the numerical aperture.

A *dichroic mirror*, also called a dichromatic beamsplitter, is a filter which split light of different wavelengths. The filter has a sharp transition between reflecting and transmitting light, resulting in short wavelengths being reflected and long wavelengths passing through[7]. This is useful when having several detectors which should detect different wavelengths. The mirror is usually angled 45° to disperse the short wavelengths 90°.

Second harmonic generation (SHG) is a nonlinear scattering process of two photons with the same wavelengths. The process is an interaction where the photons is transformed to a single emitted photon of half the wavelength. The process is dependent on orientation of electric dipoles in the specimen and aligned assemblies of asymetric molecules usually provides the proper conditions.

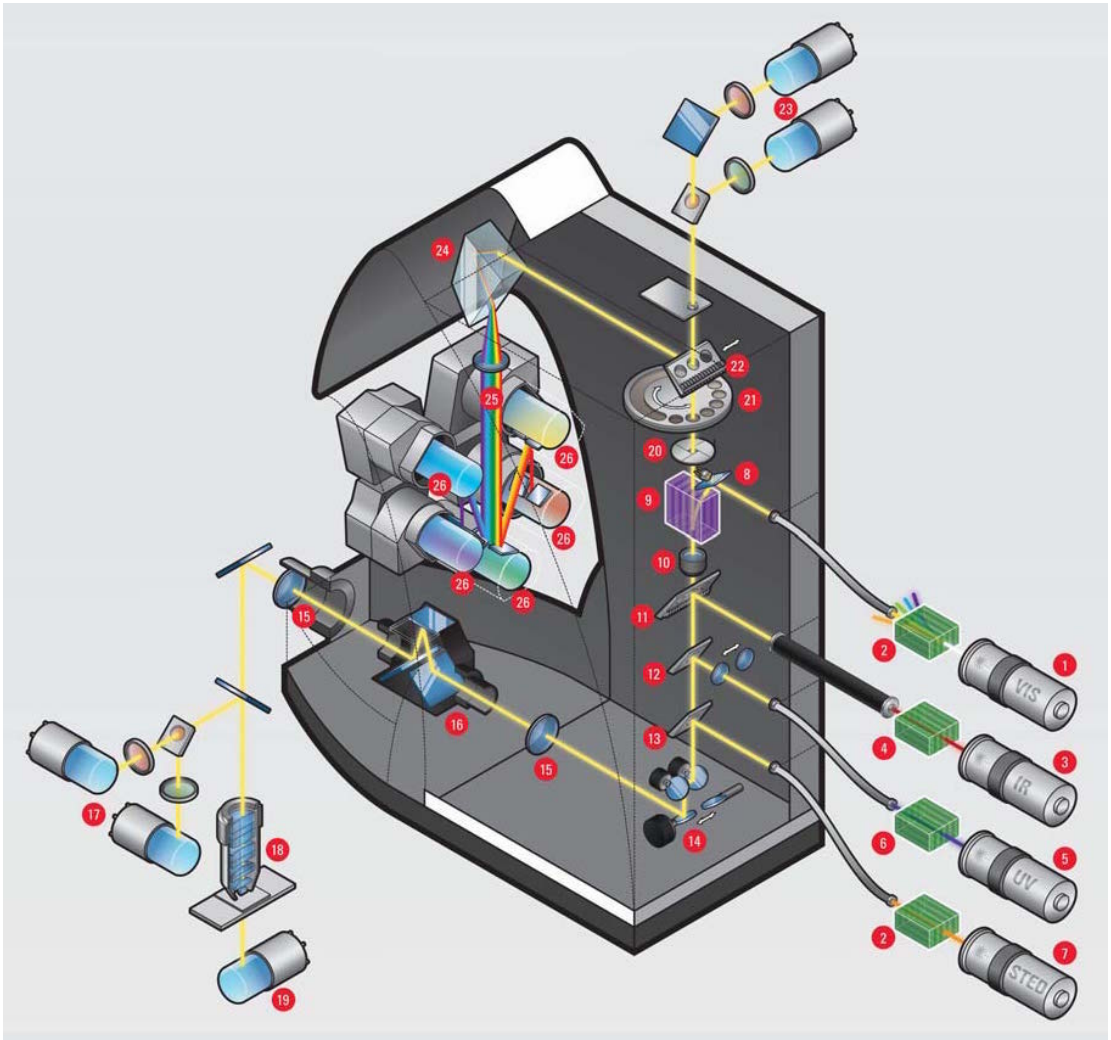


Figure 2.1: Internals of a Leica SP8 microscope. Picture from Leica SP8 brochure[8].

Collagen does hold the proper conditions for SHG-imaging[7].

As the probability for SHG is extremely low, a high intensity laser is necessary to generate it.

Image processing

Contents of this section is worked out from the book Digital Image Processing by Gonzalez and Woods[9].

An *image* is a two dimensional array of values, where each position in the array is called a *pixel*. Resolution is the number of pixels an image holds. E.g., a resolution of 1024×1024 is an image with 1024 pixels in both x- and y-direction, totalling 1×10^6 pixels. Each pixel represent a physical position of the specimen, where the value is the amount of light measured from the detector when scanning the specimen surface with a light source. The physical size of the pixel depends on objective, zoom and resolution. All images in this thesis are 8 bit grayscale images, meaning that each pixel can hold $2^8 = 256$ values. In an ideal experiment a pixel value of zero denote zero detected light and 255 is the maximum, but this is an simplification as noise is measured too.

$f(x, y)$ denotes the intesity of pixel at position (x, y), where $(0, 0)$ is the top left of the image, positive x-direction going right and positive y-direction going down. $m \times n$ will denote the number of pixels in in respectively x- and y-direction. A subscript of the image name is used if several images are discussed, e.g., m_f is the number of x-pixels in image $f(x, y)$.

The histogram of an image is the count of intensities in the image. In example, an image with 8 bit depth spans values from 0 to 255 and the histogram consists of 256 bins. The 0-bin contain the sum of pixels equal to zero. Summing up all the histogram bins gives total number of pixels in the image.

Otsu thresholding

Otsu tresholding optimizes the between-class variance in terms of intensity values. The computation is done on the image histogram, giving the optimal threshold for separating intensity classes. The output is a segmented binary image where all pixels above the threshold is `True` and the rest of the pixels `False`.

Spatial image filters

A *spatial image filter* consists of a center pixel, it's neighborhood defined by a structuring element and an operation. Structuring element is typically a rectangle, but can be of any shape. The operation can for example be calculating the mean of the neighborhood, assigning the mean value to the center pixel. Formally the spatial filter is defined as

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t). \tag{2.2}$$

Here $g(x,y)$ is the result, $w(x,y)$ is the structuring element, $f(x,y)$ is the image the filter is performed on and assuming odd size of the structuring element, $a = (m_w - 1)/2$ and $b = (n_w - 1)/2$.

In the case of a mean filter with neighborhood or 3×3, $w(x,y)$ would consist of 3 rows and 3 columns with the value $1/9$.

Sliding window filters

A *sliding window filter* is similar to a spatial filter in the sense that there is a center pixel and a neighborhood defined by a structuring element. The neighborhood is called a *sliding window* as neighborhood is updated by removing values going out of the neighborhood and adding values coming into the neighborhood when moving to the next pixel.

Typically the window is kept as a histogram in memory, instead of doing computation directly with the image values. Doing computation on the histogram can be more efficient for certain operations, as the image memory is accessed less often.

Image registration

Image registration is the process of putting images into the same coordinate system. In this context the sources are images from different microscope stage coordinates. One way of finding how images are relatively displaced is by using cross-correlation. The cross-correlation of two images is the process of zero-padding the one image and using the other image as structuring element. Cross-correlating $f(x,y)$ by $g(x,y)$ is defined as

$$h(x,y) = f(x,y) \star g(x,y) = \sum_{s} \sum_{t} g(s,t) f(x+s, y+t). \tag{2.3}$$

Here $g(x, y)$ is the structuring element of size $s \times t$ and $f(x, y)$ is the zero-padded image. The structuring element in cross-correlation is often called a *template* and the process of cross-correlation is called *template matching*. The maximum peak(s) in $h(x, y)$ is where the template has the best match, which may be in several positions if several matches are made. The cross-correlation is dependent on intensity variations and requires the images to have high entropy to get clear matches. E.g., a strictly even background have low entropy and gives equal match for the whole image.

If $f(x, y)$ and $g(x, y)$ are large images, calculation of equation 2.3 is computational costly. To reduce the calculation one might use the cross-correlation theorem which uses Fourier transform to reduce number of calculations. A 2D discrete Fourier transform (DFT) of an image $f(x, y)$ is computed by

$$F(u, v) = \mathfrak{F}\{f(x, y)\} = \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} f(x, y) e^{-i2\pi(ux/m+vy/n)}. \tag{2.4}$$

Here $F(u, v)$ is the frequency domain image and $\mathfrak{F}\{f(x, y)\}$ is the notation for the Fourier transform of $f(x, y)$.

Similar the inverse Fourier is defined as

$$f(x, y) = \mathfrak{F}^{-1}\{f(x, y)\} = \frac{1}{mn} \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} F(u, v) e^{i2\pi(ux/m+vy/n)}. \tag{2.5}$$

The sums of equation 2.4 and equation 2.5 are independent and can be separated in rows and columns, yielding the fast Fourier transform which reduces the calculation complexity from $O(mn)$ to $O(m \log m + n \log n)$.

As briefly mentioned, DFT has the property that a element wise multiplication in the frequency domain with one of the images complex conjugated is equivalent as a cross-correlation in the real domain. The cross-correlation theorem states

$$f(x, y) \star g(x, y) = \mathfrak{F}^{-1}\{F^*(u, v) G(u, v)\}. \tag{2.6}$$

Here it's assumed that images are zero padded and $F^*(u, v)$ denotes the complex conjugate of $F(u, v)$. Cross-correlation in the frequency domain is also called phase correlation.

Software

Leica LAS

Leica Application Suite (LAS) is the software that contols the SP8 microscope. LAS comes with an function called *Matrix Screener*, which allows the user to define structured areas to scan. The software uses the conce])s *fields* and *wells*. A field is essentially an image, and a well is a collection of regular spaced images. The wells may be regular spaced, or an offset between wells can be defined in the graphical user interface. When the scan job is started LAS stores images in a tree of folders in TIFF (see *Image formats).

CAM

In addition to controlling the microscope with the graphical user interface, a function called *Computer Assisted Microscopy* (CAM) can be enabled. CAM is a socket interface, meaning one send bytes over a network interface. This is very similar to how one can write bytes to a file, but in addition the socket interface can respond and send bytes back. The network interface runs on TCP port 8895 and one may communicate locally or over TCP/IP network. A set of 44 commands are available, but only three of them are intresting for the purpose of controlling scans; load, autofocusscan and startscan. More details on the interface can be read in the manual[10] or by studying the source code of the Python package leicacam[11]. Code block 1 show how one can communicate with the microscope in Python.

Code block 1 Communicating with the Leica SP8 microscope using Python.

```
from socket import socket

CAM = socket()                          # initialize object
CAM.connect(('localhost', 8895))        # connect to LAS
welcome_msg = self.socket.recv(1024)  # get up to 1024 bytes
msg = b'/cli:python /app:matrix /cmd:getinfo /dev:stage'  # cmd as bytes string
CAM.send(msg)                           # send messange
response = CAM.recv(1024)               # read response
```

XML

Extensible Markup Language is a declarative language which most high level programming languages speak, which makes it suitable for computer program communication. A XML-file

contain a single root and tree structure with parent and children nodes. Any position in the tree can be specified with an *XPath*. Code Block 2 show a typical structure of a XML-file.

Code block 2 Illustration of a typical XML-tree structure.

```
<?xml version="1.0"?>
<root>
    <parent>
        <child attr="val1">text1</child>
        <child attr="val2">text2</child>
    </parent>
    <parent>
        <child attr="val3">text3</child>
        <child attr="val4">text4</child>
    </parent>
</root>
```

The XML-file might be nested with several childen and parents, but code block 2 holds for illustration purposes. XPath for the first child in the first parent is `./parent/child[@attribute="val1"]`. Here `.` is the root, `/` defines path (or nesting if you like) and `[@attribute="val"]` defines that the attribute named `attr` should be of value `val1`. This XPath finds the child with `text1`, as this is the only child with `attr="val1"`. In converse, `./parent/child` finds all children. Code block 3 show how one would read properties from the XML-file in code block 2.

Code block 3 Accessing XML properties with the Python build-in module xml.etree.

```
import xml.etree.ElementTree as ET

tree = ET.parse('/path/to/file.xml')           # read xml
first_child = tree.find('./parent/child')      # find one element
print(first_child.attrib['attr'] == "val1")    # check attribute value
all_children = tree.findall('./parent/child') # find all elements
print(len(all_children))                       # number of elements found
```

Scanning Template

A *scanning template* is a XML-file read by LAS which defines which fields and wells to scan. The structure of the file is the following:

- `./ScanningTemplate/Properties` holds experiment settings like start position, displacement between fields and wells, which Z-drive to use, and so on.

- `./ScanFieldArray` holds all fields (images) and their settings as attributes of `./ScanFieldArray/ScanFieldData`.
- `./ScanWellArray` holds all wells (collection of images) and their settings as attributes of `./ScanWellArray/ScanWellData`.

OCR

Optical character recognition (OCR) is recognition of characters in an image. OCR internals are not discussed, but it basically works by looking at patterns in the image to convert it to text.

Image formats

Image formats referred to in this text are:

- *Tagged Image File Format* (TIFF) is ISO standarized[12] and can contain both raw and compressed images. TIFF images can be opened in most image programs.
- *Portable Network Graphics* (PNG) is both ISO and W3 standarized[13],[14]. Image data is stored with lossless compression. PNG images can be opened in most image programs.
- *Leica Image Format* (LIF) is not a standarized format. LIF can be opened by several programs for scientific image processing (e.g., LAS, Matlab and Fiji).

Chapter 3

Methods

TMA samples can contain up to 1000 samples for each glass slide[6]. Though the complexity can be handled by a human, the process of manually scanning TMA consist of a lot error prone work. Good tools to organize the work of scanning TMAs is therefor vital in helping the researcher.

The methods described here seek to provide those tools, reducing mental overhead for the microscope operator being the main aim. Using the methods described, the user avoids a lot of repetitive, trivial, labor and can turn his attention on the research. This chapter contain description of microscope settings, steps of automated scanning and procedure for correlation to clinical data.

Microscope

The images were collected with a Leica SP8 microscope using LAS software version X 1.1.0.12420 from Leica Microsystems CMS GmbH. Two lasers were used, a Coherent laser and a LASOS argon laser. Full specification of lasers is in table 3.1.

Table 3.1: Lasers

Brand	Model	Specifications
Coherent	Chameleon Vision-S	Modelocked Ti:Sapphire, wavelengths 690-1050 nm, 2500 mW, 80 MHz pulsed repetition rate, $\approx$ 75 fs pulse width

Brand	Model	Specifications
LASOS	LGK 7872 ML05	Argon continious wave, wavelengths 458, 476, 488, 496 and 514 nm, 65mW

All images are from transmitted light measured with non-descanned PMT detectors. Two non-descanned PMT detectors were used with dichrioc mirror of 495 nm and band pass filters of 525/50 nm and 445/20 nm. Rotation of scanning pattern was set to 1.7° to align scanning coordinate system with stage coordinate system (read more in *Rotation*). Frequency of scanning mirror was set to 600 lines/second (maximum speed with 0.75 zoom).

Images were saved as TIFF with 8 bit intensity depth and then converted to PNG to reduce storage space. The images were also rotated 270°, as LAS stores the TIFF-images with axes swapped with regards to the stage coordinate system. The procedure is listed in code block 4.

Code block 4 Compress and rotate images.

```
from leicaexperiment import Experiment
from PIL import Image

experiment = Experiment('path/to/experiment')
experiment.compress(delete_tif=True) # lossless PNG compression

for filename in experiment.images:
    img = Image(filename)
    img = img.rotate(270)            # image axes same as stage axes
    img.save(filename)
```

Overview images

Overview images were collected with a technique similar to bright-field microscopy except that the light source is a scanning laser. 10x air objective along with argon laser in table 3.1, 514 nm emission line was used. Output power was set to 2.48% and intensity to 0.10. Forward light was imaged using 0.55 NA air condensor with non-descanned PMT detector and 525/50 nm bandpass filter.

The aperture of transmitted light and the detector gain was adjusted so that the histogram of intensities was in the center of the total range without getting peaks at minimum and maximum values. Zoom 0.75 and 512×512 pixels image resolution was used, which gives images of $\approx 1500 \times 1500$ µm and resolution of $\approx 3 \times 3$ µm.

SHG images

SHG images were collected with a 25x/0.95 NA water objective. The pulsed infrared laser was set to 890 nm, intensity 20%, gain 40%, offset 80% and electro-optic modulator on. 0.9 NA air condensor was used and forward light was measured with non-descanned PMT detector using a 445/20 nm bandpass filter. Gain of PMT detector was adjusted so that signal spanned the whole intensity range. Aperture was set to 24 (maximum). Resolution of 1024×1024 pixels was used.

Automated TMA scanning

The automated scanning aims to lift the burden of manual labor and prevent errors in the imaging process. The procedure finds specimen spots in an overview image and scans the specimen areas with wanted acquisition parameters. The process consists roughly of the steps:

1. Collecting overview images with low magnification.
2. Segment specimen spots in the overview image.
3. Scan each specimen spot with chosen acquisition parameters (e.g., high magnification).

The steps listed above is fairly straight forward, but several instrumental and technical details are important to get a working solution. To better get an overview of the procedure, the aspects are listed here and described in further detail in it's own section.

In **step 1**, collecting overview images:

- Correcting for uneven illumination,
- adjusting scanner rotation
- and reliable stitching.

In **step 2**, segmentation:

- Discriminate specimen spots from background,
- excluding false positives
- and calculating specimen row and column position.

And in **step 3**, scanning each specimen spot:

- Calculating stage position from pixel position
- and communicating with the microscope.

Together, the steps provides an automated scanning which is invariant to variations in intensity, specimen sizes and tissue microarray size.

Step 1: Collecting overview images

To find specimen, overview images were collected with the settings described in the microscope section above. To minimize scanning time, minimum zoom (0.75) was used, which yielded the intensity variation seen in figure 3.1 (a) and (b). The uneven illumination is unwanted mainly beacuse of two reasons:

- Discriminating specimen intensities from background intensities with thresholding can give false positives when intensities are overlapping.
- Contrast is weakened, giving less clarity for human viewing purposes.

In addition, rotation of scanner raster pattern should be adjusted to avoid jagged stitch. The stitching mechanism will also be described, as existing stitching software was found to be unreliable.

Uneven illumination

The uneven illumination in the experimental setup is illustrated in figure 3.1 (a). By assuming the intensity variation in all the pixels follow the slope of the background, equalization can be done by dividing the image by the normalized intensity profile of the background. The procedure is listed in code block 5.

Code block 5 Equalizing an image

```
equalized = img.astype(np.float)        # assure datatype have real division ability
equalized -= images_minimum             # normalize
equalized /= images_maximum - images_minimum
equalized /= intensity_profile          # equalize
equalized[equalized > 1] = 1            # clip values
```

As seen in code block 5, the image is first normalized. `images_minimum` and `images_maximum` was found by selecting the median of respectively minimum and maximum intensity of all images.

Normalizing to the same range for all images is prefered to using local minimum and maximum which can give considerable differences to normalization between images.

`intensity_profile` is a curve fit for one of the background rows in a selected image. The row was found by calculating variance of all rows in the image and choosing the one with least variance. The user should verify that the row indeed is a background row by plotting it or viewing the image.

Figure 3.1(b) show the selected image and the row with least variance is indicated with a white line. The intensity profile was fitted to a second degree polynomial to avoid noise and all images were equalized by the code in code block 5. The intensity profile with it's curve fit can be seen in figure 3.2(a). The effect on pixel values can be seen in figure 3.2(b) and (c), where each dot represents a pixel value with increasing image x-position on the x-axis.

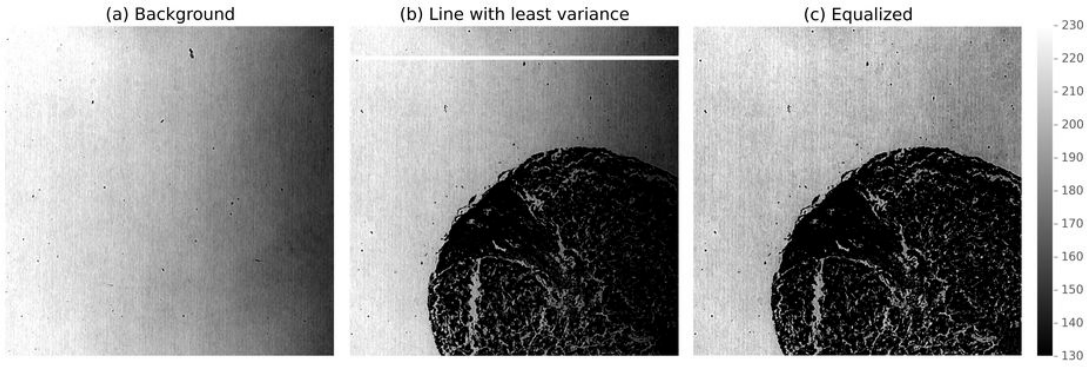


Figure 3.1: **(a)** Image of glass slide only for illustrating the uneven illumination. Dots are impurities on the glass slide. **(b)** Original image. Image is selected for finding the intensity profile. The white line is the row with least variance used for equalization. The line is wider than one pixel for viewing purposes. **(c)** Equalized version of (b). Note that (a), (b) and (c) are displaying values from 130 to 230 to highlight the intensity variation, colorbar is shown to the right.

The intensity variation was in one dimension only, which allowed for the simpler dividing by a row intensity profile. For more complex intensity variations, a similiar approach can be done by curve fitting the background to a surface, then divide images by the surface intensity profile.

Rotation

To get images registered to the stage coordinate system the scanning pattern and the stage should share the same coordinate system. It's not uncommon that it does not, giving the result

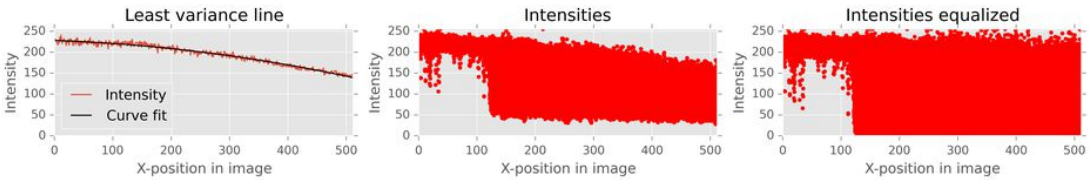
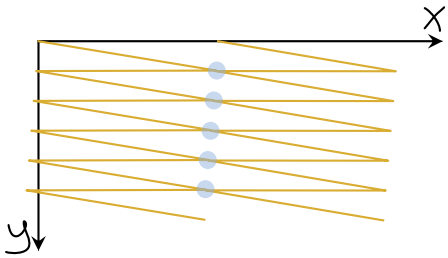
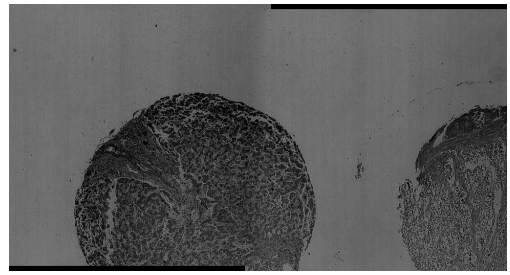


Figure 3.2: **(a)** Intensities for the line with least variance of figure 3.1(b). The curve is fitted to a second degree polynomial to supress noise. **(b)** Intensities for image in figure 3.1(b). Each dot represents a pixel. **(c)** Intensities for the equalized image in figure 3.1(c). Each dot represents a pixel. Note that the intensities is both spread across the whole intensity range (0-255) and the skewness is fairly straightened out.



(a) Illustration of rotated scanning mirror coordinate system. Stitch highlighted as dots on end of lines.



(b) Stitch of two images when stage and scanning pattern does not hold the same coordinate system.

Figure 3.3: Illustrations and stitch of two images with scanning pattern rotated compared to stage movement. In (a) the first row of the first image lines up with second row in second image. The second image should therefor be one pixel above the first image. In (b) relative scanning pattern rotation is counter clockwise, giving the second image below the first image. A calculation of stage position by y-equivalent to equation **??** gives a systematic error in the y-position if stitches are jagged.

of a jagged stitch seen in figure 3.3.

Relative rotation between scanner raster pattern and stage coordinate system was measured by calculating displacement of two neighbor images using phase correlation. The rotation is then given by

$$\theta = \arctan\left(\frac{\Delta y}{\Delta x}\right). \tag{3.1}$$

Here Δy and Δx is the displacement in pixels between images. To align the coordinate systems,

scanning rotation was set to $-\theta$ in LAS.

Stitching

To allow whole specimen spots to be found by segmentation, overview images must be stitched together. Stitching by existing software gave unreliable results seen in figure 3.4(a) due to lack of control in translation constraints. To make sure the stitching does not fail, the method here takes the assumptions:

- Images are regular spaced.
- Images are of same size.
- Scale in edge of images are constant, e.g., translation is the only transformation between images.
- Side by side images have translation in one dimension only (see section on rotation above).

The procedure may not work well for all experimental setups, but showed good performance in regards to precision for the Leica SP8 stage.

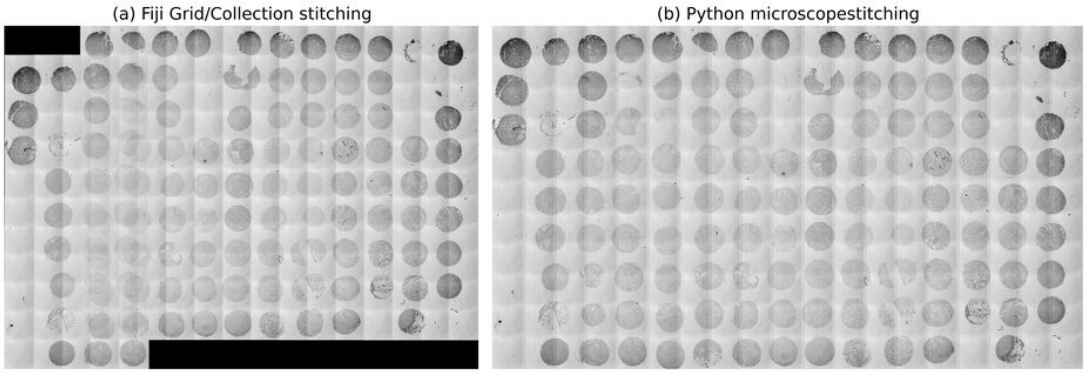


Figure 3.4: **(a)** Unreliable stitching with Fiji. The image translation calculated by phase correlation is chosen without adhering to displacement constraints. **(b)** Using same overlap for all images gives reliable stitch.

The procedure of stitching consists of phase correlating all neighbor images, calculating the median translation and using this median translation for all images. The median is used as correlation between two images with little entropy in the seam is prone to fail. More details on this matter are described in the discussion. Code block 6 show the basics of the procedure on a row of images for sake of simplicity.

Code block 6 Stitch row of images by using median translation from phase correlation.

```
from skimage.feature import register_translation
import numpy as np

# find all neighbor translations
translations = []
prev = row_of_imgs[0]                    # row_of_imgs: list of 2d arrays
for img in row_of_imgs[1:]:              # exclude first image
    translation, error, phasediff = register_translation(prev, img)
    translations.append(translation)     # add translation to the list
    prev = img                           # reference to previous image
translations = np.array(translations)    # allow for slice notation
offset_y = np.median(translations[:,0])  # median x translation
offset_x = np.median(translations[:,1])  # median y translation
assert offset_x == 0, "x-offset should be zero, " \\
                    + "adjust the scanning mirror rotation"

# combine into one image
y, x = img.shape                         # assume images of same size
n = len(row_of_images)                   # number of images
total_height = n*y - offset_y*(n-1)      # stitched image height
stitched_img = np.zeros((total_height, x))  # empty image
for i, img in enumerate(row_of_images):
    y_start = i*y - i*offset_y           # limits in stitched image
    stitched_img[y_start:y_start+y, :] = img
```

Step 2: Find specimen spots by segmentation

After step 1 we have a large stitched overview image of specimen spots. We would now like to classify which parts of the image that are background and which parts hold the specimen spots. Looking at figure 3.4(b) the contrast in the center of the TMA is weaker than on the edges. To improve this, the crucial observation is that background signal tend to vary less than specimen signal. This fact makes it easier to discriminate specimen spots to background by filtering the image before segmenting it with Otsu.

In addition, relying only on Otsu thresholding gives a lot of small segments which are not specimen spots. To exclude these false positives, area size of segments were used as a classification.

Lastly, we'll want to calculate row and column of the specimen spots so that the image can be correlated to clinical data.

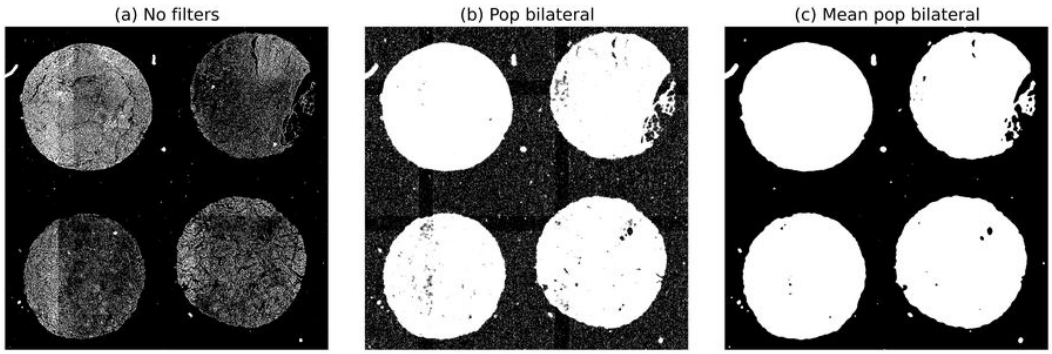


Figure 3.5: Otsu thresholding of figure 3.4(b) zoomed into four specimen spots for clarity. **(a)** Otsu thresholding applied without any filters. Picks out dark areas, but disjointed, especially for brighter areas in specimen spots. **(b)** Thresholding after a local bilateral population filter. Quite noisy in the background. **(c)** Thresholding after local bilateral population and local mean filter. Background noise is gone and sample spots are segmented continuously.

Filter and segment the overview image

As briefly mentioned, the goal of filtering the overview image is to improve discrimination between areas with background and specimen so specimen spots can be distinguished. A filter that has the appropriate characteristics is the population bilateral filter, which counts number of pixels in the neighborhood of the center pixel that is within a specified intensity range relative to the center pixel intensity.

The stitched overview image was $5122 \times 8810 = 45$ Megapixels, giving total filter time of 20 seconds with `skimage.filters.rank.pop_bilateral` on a single core of a Intel i3 2.3 GHz CPU. As the process of segmentation was implemented as an interactive graphical user interface, filter time of 20 seconds was considered unresponsive. To approve responsiveness, the filter was implemented as a sliding window filter in Python and compiled with numba[15]. The numba compiled filter took 4.5 seconds on a single core of a Intel i3 2.3 GHz CPU. As the microscope computer was equipped with 16 CPU cores, the filtering was parallized with dask[16], giving filtering in real time.

Assuming one has an algorithm that updates the local histogram based on a structuring element, the inner computation of a population bilateral filter is given in code block 3. A full implementation of the filter can be seen in the filters submodule of leicaautomator[17]. Values of `s0 = s1 = 10` gave high discrimination of specimen and background on overview images collected with settings specified in the microscope section.

```
def pop_bilateral_inner_computation(histogram, val, s0, s1):
```

```
    "Returns number of pixels that are within [val-s0, val+s0]."
    count = 0
    histogram_max = histogram.size

    for bin in range(val-s0, val+s1+1):
        if bin < 0 or bin >= histogram_max:  # do not try to count outside range
            continue
        count += hist[bin]                   # add counts from bin
    return count
```

To reduce noise after the bilateral population filter, a mean filter was applied. The size of structure element was 9×9 pixels for both filters. Figure 3.5(a), (b) and (c) show how the segmentation is affected by the filters. Code for reproducing the steps is in code block 7.

Code block 7 Filter and segment an image with local bilateral population and Otsu thresholding.

```
import numpy as np
from skimage.filters import threshold_otsu
from skimage.util import apply_parallel  # available from v0.12
from scipy.ndimage import uniform_filter
from leicaautomator.filters import pop_bilateral

selem = np.ones((9,9))                   # 9x9 structuring element
filtered = apply_parallel(pop_bilateral, image, depth=4,
                          extra_keywords={'selem': selem})  # apply filter on
                                                            # all cpu cores
filtered = apply_parallel(uniform_filter, image, depth=4,
                          extra_keywords={'size': 9})       # mean filter
threshold = threshold_otsu(filtered)     # get optimal threshold
segmented = filtered >= threshold        # low values indicate specimen
```

Excluding false positives in segmentation

After segmentation we have a binary image as shown in figure 3.5(c). The image contains several small dots that are not specimen spots. The dots can be removed by sorting all segment regions by area size, then excluding the smallest ones. Figure 3.6(a) show segments sorted by falling area size. Code block 8 illustrate how the small segments were excluded, keeping only the largest ones.

Calculating row and column position

As specimen spots are pretty well arranged in rows and columns, calculating the specimen row and column position lifts the burden of labeling the scanned specimen by the user.

Code block 8 Exclude small segments which are false positives.

```
from skimage.measure import label, regionprops

labels = label(segmented, background=0) # background=0: exclude background
regions = regionprops(labels)           # measure region properties
regions.sort(key=lambda r: -r.area)     # sort by area size, largest first

max_regions = 126
if len(regions) > max_regions:
    regions = regions[:max_regions]     # only keep max_regions
```

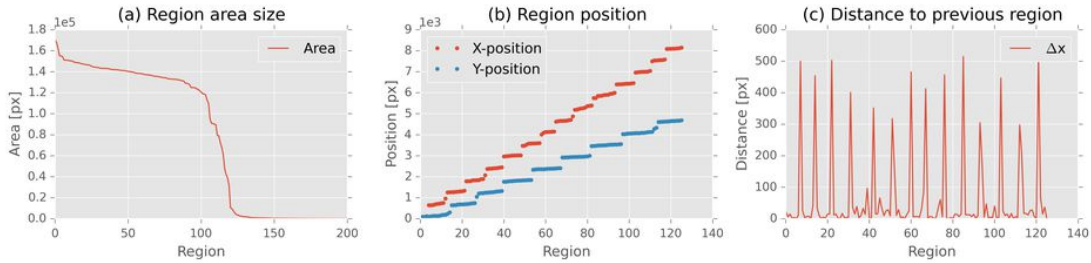


Figure 3.6: **(a)** Sorted region areas. Area size drops dramatically around region 125 comparable to the number of specimen spots on each slide which was $14 \cdot 9 = 126$. Plot does not have corresponding x-axis with (b) and (c), as regions are sorted by size. **(b)** Regions sorted by position. The two plots do no share the same x-axis. There is a gap between the positions when row and columns are increasing. **(c)** X distance to previous region when regions are sorted by x-position. Same x-axis as in (b) for the x-position plot. 14 peaks indicate that the image contain 15 columns.

By looking at two fairly vertical columns of specimens, one can observe that the x-coordinate of specimens group around a mean x-coordinate and that there is a jump in x-coordinate when going to the next column of specimens (seen in figure 3.6(b)). A derivative can be calculated by sorting the segmented regions by coordinate and subtract the current region's position to the previous region's position (seen in figure 3.6(c)). The derivative can then be used to increment row or column when looping through the segmented regions and adding the row and column property to the region in question. The procedure is shown in code block 9.

Interactive segmentation

As experimental factors like detector gain, laser intensity, light absorption of specimen, etc. can give considerable variations in images, step 2 was implemented as a interactive graphical user

Code block 9 Calculate row and column position to specimen spots.

```
for r in regions:
    r.y, r.x, r.y_end, r.x_end = r.bbox  # for notational convenience

for direction in 'yx':                   # same algorithm for row and columns
    regions.sort(key=lambda r: getattr(r, direction))

    previous = regions[0]
    for region in regions:               # calc distance to previous region
        dx = getattr(region, direction) - getattr(previous, direction)
        setattr(region, 'd' + direction, dx)
        previous = region
```

interface. The interface allows the user to adjust filter settings and verify which regions to scan by deleting, moving or adding regions. The interface is show in figure 3.7.

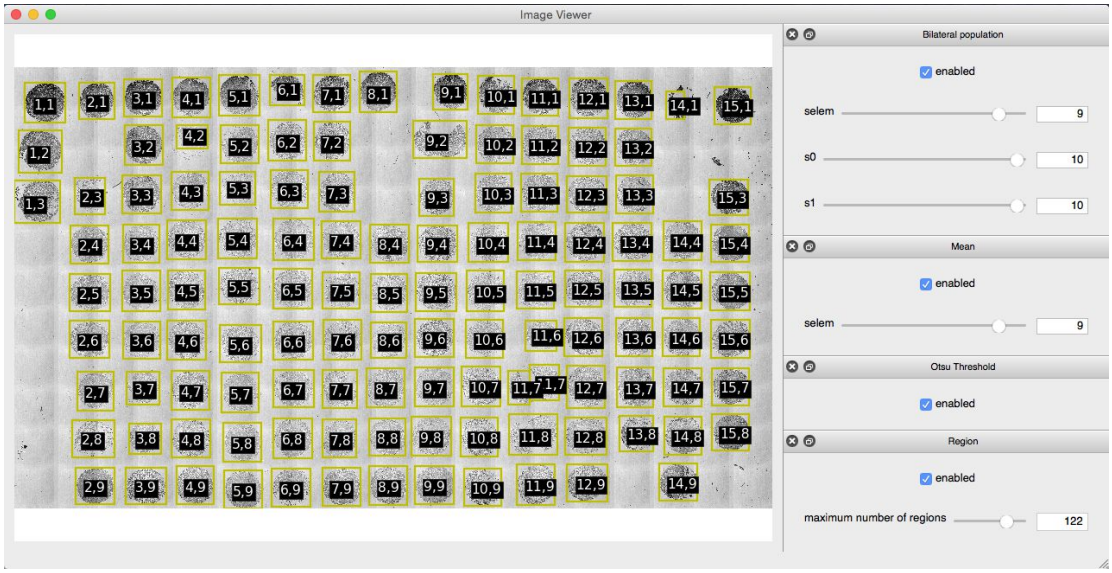


Figure 3.7: The process of segmentation in a graphical user interface. Regions 4,2, 11,7 and 14,1 might be adjusted by the user, all other regions are detected fairly well.

Step 3: Scanning each specimen spot

From step 2 we have a list of regions and their pixel position in the stitched overview image. Last step is to calculate the stage position to the regions and scan the regions by communicating with the microscope.

Calculate stage position from pixel position

To convert pixel position to stage position one need a reference point and the pixel resolution. For simplicity, the procedure for calculating stage coordinate is shown for x-coordinate only, as the calculations for y-coordinate is fully equivalent. Pixel resolution was calculated by

$$x_{resolution} = \frac{\Delta x}{\Delta X}. \tag{3.2}$$

Here Δx is displacement in pixels from the stitch in step 1, and ΔX is stage displacement in meters read from XPath `./ScanningTemplate/Properties/ScanFieldStageDistanceX` in the overview scanning template in the experiment folder (AdditionalData/{ScanningTemplate}overview.xml).

Keeping stage position constant when zooming, either by changing objective or decreasing amplitude of scanning mirror oscillation, yields the same physical position in center of field of view. This means that image stage position reported by the microscope is the center pixel. One can use the center of the first image as the reference point, but using pixel (0,0) is simpler as one can find out where the center pixel is one time, then later forget about it.

In other words, the reference point for x-position is at $f(0, y)$, the left most pixel. This reference point was calculated by

$$X_{ref} = X_{center} - \frac{m}{2} \cdot x_{resolution}. \tag{3.3}$$

In equation 3.3 X_{center} is the stage position for the top left image, m is the number of pixels in the image and $x_{resolution}$ is from equation 3.2. X_{center} was read from XPath `./ScanFieldArray/ScanFieldData[@WellX="1"][@WellY="1"][@FieldX="1"][@FieldY="1"]/FieldXCoordinate` in the overview scanning template.

The stage x-coordinate for any pixel is then given by

$$X = X_{ref} + x \cdot x_{resolution}. \tag{3.4}$$

Here X is the stage x-coordinate, X_{ref} is the reference point and $x_{resolution}$ is from equation 3.2.

Moving the stage to the position calculated from equation 3.4 will center the location in the field of view. By reversing equation 3.3 one moves the position to the edge of the image. How much to shift the position depends on the field of view in the scan job, given by the objective and the zoom. The start coordinate of the scan job was calculated by

$$X_{start} = X + \frac{\Delta X_{field-distance}}{2}. \tag{3.5}$$

Here X_{start} is the x-coordinate for the first image, X is calculated from the bounding box coordinate to the region in question, and $X_{field-distance}$ is stage displacement between fields. Similar to equation 3.2, $X_{field-distance}$ was read from `./ScanningTemplate/Properties/ScanFieldStageDistanceX` in the acquisition scanning template found in folder `C:\Users\TCS-User\AppData\Roaming\Leica Microsystems\LAS\MatrixScreener\ScanningTemplates`.

Using the stage displacement and not the true field of view gives an error in the calculation of X_{start} by

$$\epsilon = \frac{1}{2}(\Delta X_{field-distance} - \Delta X_{img}), \tag{3.6}$$

as stage displacement $X_{field-distance}$ is not strictly equal to the field of view X_{img} when images are scanned with overlap. This was considered neglectible as $\Delta X_{field-distance} \approx \Delta X_{img}$ and number of scanned fields was calculated by

$$F_x = \lceil \frac{\Delta X}{\Delta X_{field-distance}} \rceil, \tag{3.7}$$

which is a slight overestimate. In equation 3.7 F_x is number of fields in x-direction, ΔX is size of detected specimen spot and $X_{field-distance}$ is displacement between fields.

Scanning each region

After the step above one have start position X_{start} and number of fields to scan F_x. What remains is communicating with the microscope and record output filenames of the scans.

To avoid unnecessary long stage movements between rows or columns, regions were looped through in a zick-zack pattern, given by their row and column position. For each region the scanning template was edited, the template was loaded and the scan was started through CAM. Single scanning templates were used due to a LAS software limitation; scanning templates with irregular displaced wells is not supported. Code block 10 illustrates the scanning procedure, using the high level communication interface leicacam.

Code block 10 Automated scanning of regions with CAM.

```
from leicascanningtemplate import ScanningTemplate as ST
from leicaautomator import zick_zack_sort
from leicacam import CAM

cam = CAM()  # instantiate connection to microscope

# regions sorted as [r(1,1), r(1,2), r(2,2), r(2,1), r(3,1), r(3,2), ...]
# here r(2,1) is region(col=2, row=1)
regions = zick_zack_sort(regions, ('well_x', 'well_y'))

tmpl_path = r"C:\Users\TCS-User\AppData\Roaming\Leica Microsystems\LAS" \
          + r"\MatrixScreener\ScanningTemplates" + "\\"
tmpl_name = tmpl_path + '{ScanningTemplate}leicaautomator'
for n, region in enumerate(regions):
    tmpl = ST(tmpl_name + str(n%2) + '.xml')  # alternate between tmpl_name0/1.xml
                                              # LAS cannot load same filename twice
    tmpl.move_well(1, 1, region.real_x, region.real_y)  # start position for first field
    tmpl.enable_fields((region.fields_y, region_fields_x))  # limit size of scan
    tmpl.write()                            # save scanning template
    cam.load_template(tmpl.filename)        # load scanning template into LAS
    cam.autofocus_scan()                    # do autofocus
    cam.wait_for('inf', 'scanfinished')  # wait for autofocus to finish
    cam.start_scan()                        # run scan job
    region.experiment_name = cam.wait_for('relpath')['relpath']  # record output filename
    cam.wait_for('inf', 'scanfinished')  # wait for scan to finish
```

Alignment of z-plane

The specimen spots in figure 1.1 are 5 µm thick, making it challenging to keep the distance from specimen plane to objective equal when moving 25 mm. Also, if the specimen plane is substantially tilted, a single image might become out of focus at edges. To overcome changing z-coordinate when moving large distances, the stage insert seen in figure 3.8(b) was developed. The stage insert allows the user to adjust the specimen plane before scanning.

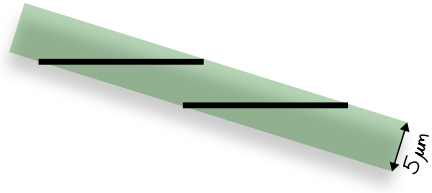
To demonstrate the level of accuracy required for the stage insert consider the field of view of a 63x objective with minimum zoom (0.75) which is 246×246 µm. To get the stage insert steady for this level of precission, mouldable glue was added to corners of stage insert and glass slide holder. This makes both the stage insert and glass slide fixed, even when adjusting the specimen plane.

The specimen plane was adjusted by the procedure:

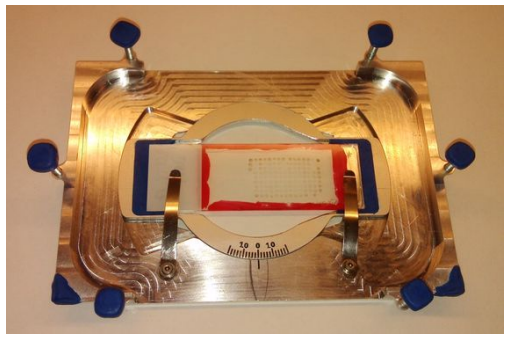
1. Find which of the corners in the tissue microarray has the highest z-coordinate.

2. Set stage z-coordinate some microns above the highest corner.
3. Adjust all corners into focus (e.g., lifting them).
4. Repeat until specimen plane is leveled at same z-coordinate.

This makes it possible to image a whole specimen of 1.2 mm with one autofocus scan only and also avoids the scenario illustrated in figure 3.8(a).



(a) Tilted z-plane of sample seen from the side. Black lines indicate two images and the objective focus for those. Illustration is not drawn in scale.

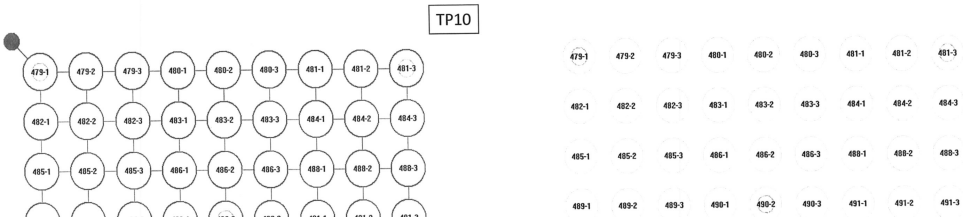


(b) Sample holder with adjustment of specimen-plane.

Figure 3.8: (a) When having a tilted specimen plane, stage z-coordinate must be adjusted to keep specimen in focus when moving x- or y-coordinate. Also, the seam between images is not be from the same physical area, which might cause some trouble for thicker samples when they are stitched. **(b)** Stage insert which allows the user to adjust the specimen plane. Mouldable glue was used to make the insert fit precisely in the microscope and keep the glass slide fixed in it's holder.

Correlating images with patient data

Each TMA glass slide contains samples from 42 patients, meaning that there is three specimen spots for each patient. The slides are numbered and specimen spots on all slides are given identifiers. Figure 3.9 illustrates some of the identifiers for slide ten (TP-10, tumor peripheral number ten), called a slide map. As seen, the identifiers consists of two numbers. The first number is the patient identifier and the second number is the sample number. The patient identifier is not incrementing systematically, so the slide maps were scanned to read out the identifier for each position.



(a) Original slide map. **(b)** Filtered slide map.

Figure 3.9: (a) Top of slide map TP-10. Identifiers are not incrementing systematically and are inside circles, making them hard to read directly with OCR. **(b)** Only text inside circles are kept after the slide map has been filtered.

Before the slide maps were read with OCR, they were filtered to include only text inside circles. The filter removes the rest by:

- Segment the image with Otsu threshold.
- Widen segments by dilation (make sure segmentation connects lines).
- Selects circles in the segmented image by a circle score.
- Remove everything outside selected circles.

The circle score was calculated as shown in code block 11.

Code block 11 Calculate score of region being a circle.

```
def circle_score(r):                    # r is a skimage.measure.regionprops object
    y0,x0,y1,x1 = r.bbox                # for notational convenience
    height = y1-y0                      # calc height
    width = x1-x0                       # calc width
    radius = (r.convex_area/3.14)**0.5  # calc expected radius from convex area
    score = 10-abs(height-width)        # high score if height == width
    score += 10-abs(radius - height/2)  # high score if height/2 == expected radius
    if r.area < 5000 or r.area > 8000:  # penalty for wrong sizes
        score -= 20
    return score
```

All slide maps were filtered with code block 12. After the filtering, Prizmo[18] was used to read the slide maps. The text output was checked for errors programatically. The following was checked:

- Identifier should be of correct format.

- Identifier should increment.
- Patients should be registered with correct slide in database column `TP_nr`.
- Each patient should have three samples.

Code block 12 Filter slide map and keep only text inside circles.

```
import numpy as np
from skimage.morphology import binary_dilation
from skimage.measure import label, regionprops

thresh = filters.threshold_otsu(img)      # segment image with Otsu thresholding
binary = img <= thresh
selem = np.ones((3,3))
binary = binary_dilation(binary, selem)  # enhance lines
labeled = label(binary)                   # find connected segments

mask = np.zeros_like(img, dtype=np.bool) # create mask of circles in image
for r in regionprops(labeled):            # for every segment
    if circle_score(r) > 0:               # circle found
        y,x,y1,x1 = r.bbox                # for notational convenience
        m = np.index_exp[y:y1, x:x1]      # where circle is found
        mask[m] = r.convex_image          # use the convex image as mask

img[-mask] = 255                          # set all pixels except contents of
                                          # circles to 255 (white)
```

OCR errors were fixed manually and other errors were recorded (see section Slide map errors in the appendix).

Every patient identifier from the slide map was saved to a Stata database along with its slide number, row and column. A database with outcomes of was supplied, and code block 13 show how the clinical data can be correlated with specimen spots.

Code block 13 Get patient outcome of sample on TP-1 row 3 column 5.

```
import pandas as pd

locations = pd.read_stata('data/ids/locations.dta')    # read databases
clinical_data = pd.read_stata('data/clinic_data.dta')

condition = (locations.TP_nr == 1) & \   # position query
            (locations.TP_rad == 3) & \
            (locations.TP_kolonne == 5)


patient_id = locations[condition]['ID_deltaker']  # get patient id
assert len(patient_id) == 1                       # 1 patient registered at row/col

condition = clinical_data.ID_deltaker == patient_id.iloc[0]  # clinical data query
outcome = clinical_data[condition]['GRAD']   # get outcome
```

Chapter 4

Discussion

The glass slides in this discussion holds 14 columns of specimen spots, which is 60 non-overlapping images with a 25x objective. This means that an operator of the microscope must keep track of the current stage position in the array with limited field of view.

Automated scanning is a low hanging fruit because we have the conditions:

- Specimen spots in TMA are relatively easy to discriminate to background.
- Tissue is somewhat aranged.
- Tools in microscope software exists for controlling a scan.

Scanning

To illustrate the pros of using the method described in this thesis, lets compare it to the manual approach. By using LAS matrix screener, the procedure is fairly structured. The manual labor in the scanning would roughly consist of:

1. Count number of rows and columns.
2. Align TMA in microscope.
3. Measure average inter sample displacement.
4. Find the maximum sized specimen spot and measure it's size.

5. Define an experiment holding the correct number of rows, columns, displacement between samples and sample size.
6. Update inter sample offsets one by one.
7. Potetially disable fields on specimen spots with smaller size than the largest.
8. Potentially identify and rule out missing samples.
9. Make sure autofocus positions holds signal (e.g., specimen spot should be in the autofocus image).
10. Scan.

The procedure was tested out and step 6 was the most labor intensive, browsing through 126 specimen spots and aligning them. An alignment of one specimen spot took about 40 seconds, giving 1.5 hours of intensive click-and-adjust. Also, an error in some of the steps can potentially disrupt steps further down the line, making the procedure even more time consuming. In example, inaccuracy in average displacement between samples leads to displacement adjustment of many wells, accidentally bumping the sample holder could impose restart of the procedure, and so on.

A simple means to avoid some of the steps in the intricate procedure above is using a single scan containing the whole matrix area. The procedure then simplifies to:

1. Align TMA in microscope.
2. Find outer boundaries.
3. Create predictive focus map or define autofocus for more or less regular spaced intervals containing a specimen spot.
4. Scan.
5. Separate specimen spots in images and assign row and column to them.

Compared to the first procedure listed, this procedure have the advantage of being less labor intensive when on the microscope, but manually browsing through $24\,\text{mm} \cdot 15\,\text{mm} \,/\, (400\,\mu\text{m})^2 = 2250$ images may be a daunting task without a specialized tool.

The main concern with the last procedure was focus and a couple of scans confirmed the concern by having out of focus portions. The out of focus can be of several reasons, e.g., inter specimen z-displacement or temperature changes moving the specimens in z-direction. As the autofocus in LAS runs before the scan, the only way to tackle temperature changes is by chopping up the scan in several chunks. As the goal was to reduce manual labor, doing this as a part of the procedure was not considered viable.

In other words, the most likely way to get desired result is by using the first procedure listed. So most part of the procedure was automated and the method described is therefore a combination of the two procedures above. Several of the steps remain the same but automated, so the procedure for the user of the microscope reduces to:

1. Align TMA in microscope.
2. Find outer boundaries for overview scan.
3. Verify that the algorithms have picked out the specimen spots.

- If not, the user may adjust filter settings or directly edit the detected regions.

4. Scan.

This was considered to meet the goals; reduce mental overhead when collecting images from TMA glass slides.

Uneven illumination

Rotation

LAS comes with a interactive graphical user interface for calibrating the scanning rotation. When using the rotation calibration a live image is shown with a line drawn in the middle of the image. One can adjust the rotation in real time while moving the stage. A reference point should then follow the line if the scanning mirror and stage holds the same coordinate system. The user himself have to find the rotation in a inductive manner by counting pixels or measuring how far the reference point moves away from the line when moving the stage. Accuracy depends on how easily the reference point is distinguished from the rest of the image and how thoroughly the user is with his measurements. In comparison, the procedure described in the rotation section of the method gives the same precission in less time.

Stitching

With 10x objective and 0.75 zoom, maximum field of view is reported as 1550×1550 µm. Average specimen spot diameter was ≈ 1200 µm. These two facts would allow for imaging specimen spots

into separate images if they were neatly arranged. This was not found out to be true for our dataset, and it would also burden the user of the microscope to measure and define a scan with correct inter specimen displacement. A more robust way is therefor to combine all images into one.

Combining images can be done in interactive manner, where a program loads images as one "moves" around. But creating this abstraction would demand for a way other programs can "talk" to the abstract image object containing all images. Therefor a simpler approach was used, stitching all images into one large image. This allows for any program that can open PNG to work with the images.

First approach on stitching was to use existing stitching software, in specific the *Grid/Collection stitching*-plugin of Fiji[19]. The plugins finds displacements between images by using phase correlation, and it works fairly well except for the lack of control when phase correlation fails. The failing of the phase correlation is mainly due to little entropy in the seam between images. It can be seen in figure 3.4, where the failed row have to much overlap. The failed row is a clean cut in the sense that the overlap between the images contain background only and no specimen. A background surface is quite even and gives a flat correlation in contrast to the wanted peak which express a match is found. In other words, the overlap between the images contain too little information for correlation and the match fails.

In addition to failures of phase correlation, we would also like to constrain stitch between two images to be in one dimension only. This is due to the systematic error which may occur if coordinate system of stage and scanning pattern is not the same. E.g., consider two side by side images as in figure 3.3. We know that the stage translation is only in x-direction, but the phase correlation tells us otherwise. As we want to register images into the stage coordinate system, rotation of scanning mirror is adjusted, but some minor rotation may still be experienced. This might be due stage inaccuracy, unlinearities in scanning pattern or wrong match from the phase correlation. Whatever the cause, offsetting images in dimension only gives at worst an error in X in the end of every stitch, but in case of offsetting in bot dimensions gives at worst a growing error. A way to overcome the error is by calculating X from the nearest image metadata, but this was not looked into.

Taking away outliers in the registered translation of figure 3.4 gave standard deviation of 2.5 pixels, which in the context of overview images gives enough precission for defining the SHG scan job.

The stitching algorithm can be used with the python package microscopestitching[20], code block 14

show an example of how to use it.

Code block 14 Stitching images with the Python package microscopestitching.

```
from microscopestitching import stitch
from glob import glob

files = glob('path/to/images/*')
images = []
for i, file in enumerate(files):
    # rectangle of 4 rows and len(files)//4 columns
    row = i % 4
    column = i // 4
    images.append((file, row, column))

stitched_image = stitch(images)
```

Segmentation

Filtering

Calculating row/col and correlating to clinical data

Interactive user interface

Communicating with microscope

Leica LAS design:

- user should be mainly in LAS
- automating on the side as a supplement
- load before CAM can be used
- does not load all settings from XML
- no ‘take single image’ command

Stage insert

Chapter 5

Conclusion

A procedure for collecting microscope images of tissue micro arrays and correlating specimen array elements to clinical data has been demonstrated. The software packages are developed with the Leica SP8 microscope in mind, but could be adjusted for other microscope that has the ability to scan and export images by a communication interface.

Chapter 6

Appendix

Python software

The software in this thesis is written in Python due to Python's cross-platform support, simple syntax and vast scientific ecosystem. With Python one gets free access to a lot of scientific software libraries of high quality and top-level support through channels like Github. As source code for most libraries are available, stepping into the nitty-gritty details can give insight in algorithms and be very educational.

Any Python package mentioned in the code blocks is install-able through pip. In example leicacam can be installed by opening a terminal and type `pip install leicacam`. The computer must have pip[21] and the required compilers if the package depends on compiling code. This is true for most of the software, it depends on fast algorithms implemented in compiled languages like C and Fortran.

Compiling the huge scientific libraries like numpy and scipy can take a while, so it's recommended to use a Python distribution like Anaconda[???]. Anaconda pre-ships with the most common scientific libraries and it also contains the package manager conda which have pre-compiled packages available for most operating systems.

Slide map errors

```
TP2, row  3, col  6 - patient id missing in db: 66
```

```
 TP6, row  1, col  9 - patient id missing in db: 222
 TP3, row  1, col  3 - id 68, wrong TP_nr in db: 3.0 != 2.0
 TP6, row  1, col  3 - id 209, wrong TP_nr in db: 6.0 != 4.0
 TP6, row  1, col  6 - id 221, wrong TP_nr in db: 6.0 != 5.0
TP22, row  2, col  6 - id 130, wrong TP_nr in db: 22.0 != 3.0
TP22, row  2, col  9 - id 244, wrong TP_nr in db: 22.0 != 5.0
TP22, row  3, col  3 - id 281, wrong TP_nr in db: 22.0 != 6.0
TP22, row  3, col  6 - id 296, wrong TP_nr in db: 22.0 != 6.0
TP22, row  3, col  9 - id 309, wrong TP_nr in db: 22.0 != 6.0
TP22, row  4, col  3 - id 318, wrong TP_nr in db: 22.0 != 6.0
TP22, row  4, col  6 - id 376, wrong TP_nr in db: 22.0 != 7.0
TP22, row  4, col  9 - id 396, wrong TP_nr in db: 22.0 != 8.0
TP22, row  5, col  3 - id 413, wrong TP_nr in db: 22.0 != 8.0
TP22, row  5, col  6 - id 449, wrong TP_nr in db: 22.0 != 9.0
TP22, row  5, col  9 - id 453, wrong TP_nr in db: 22.0 != 9.0
TP22, row  6, col  3 - id 487, wrong TP_nr in db: 22.0 != 10.0
TP22, row  6, col  6 - id 493, wrong TP_nr in db: 22.0 != 10.0
TP22, row  6, col  9 - id 525, wrong TP_nr in db: 22.0 != 10.0
TP22, row  7, col  3 - id 728, wrong TP_nr in db: 22.0 != 15.0
 TP3, row  9, col  6 - TP_nr not registered in db for ID_deltaker 140
 TP5, row  9, col  9 - TP_nr not registered in db for ID_deltaker 251
 TP9, row 10, col  9 - there should be 3 samples: ['467a-1']
 TP9, row 11, col  3 - there should be 3 samples: ['467b-1', '467b-2'])
 TP9, row 12, col  6 - there should be 3 samples: ['471a-1', '471a-2']
 TP9, row 12, col  9 - there should be 3 samples: ['471b-1']
TP10, row  8, col  6 - there should be 3 samples: ['507-1', '507-2']
TP10, row 12, col  6 - there should be 3 samples: ['525-2', '525-3']
TP11, row 11, col  6 - there should be 3 samples: ['566-1', '566-2']
 TP3, row  1, col  3 - patient id did not increment:
    ['68-1', '68-2', '68-3']       <  ['102b-1', '102b-2', '102b-3']
 TP4, row  1, col  3 - patient id did not increment:
    ['162a-1', '162a-2', '162a-3'] <  ['163-1', '163-2', '163-3']
 TP6, row  1, col  3 - patient id did not increment:
    ['209-1', '209-2', '209-3']    <  ['268-1', '268-2', '268-3']
TP11, row  6, col  3 - patient id did not increment:
```

```
    ['549-1', '549-2', '549-3']    <  ['552-1', '552-2', '552-3']
TP22, row  2, col  6 - patient id did not increment:
    ['130-1', '130-2', '130-3']    <  ['3067-1', '3067-2', '3067-3']
```

References

1. Statistisk sentralbyrå. *Folkemengde, 1. Januar 2015*. Statistisk sentralbyrå; 2015. http://www.ssb.no/befolkning/statistikker/folkemengde/aar/2015-02-19. Accessed May 21, 2015.

2. Statistisk sentralbyrå. *Dødsårsaker, 2012*. Statistisk sentralbyrå; 2013. http://www.ssb.no/helse/statistikker/dodsarsak/aar/2013-11-01. Accessed May 21, 2015.

3. Brabrand A, Kariuki II, Engstrøm MJ, et al. Alterations in collagen fibre patterns in breast cancer. a premise for tumour invasiveness? *APMIS*. 2015;123(1):1-8. doi:10.1111/apm.12298.

4. Python Software Foundation. The official home of the python programming language. Python.org. 2015. https://www.python.org/. Accessed May 22, 2015.

5. Seljebu A. Arve seljebu's repositories. 2015. https://github.com/arve0?tab=repositories. Accessed May 22, 2015.

6. Kononen J, Bubendorf L, Kallionimeni A, et al. Tissue microarrays for high-throughput molecular profiling of tumor specimens. *Nat Med*. 1998;4(7):844-847. doi:10.1038/nm0798-844.

7. Murphy DB, Davidson MW. *Fundamentals of Light Microscopy and Electronic Imaging*. Hoboken, N.J.: Wiley-Blackwell; 2013.

8. Leica Microsystems CMS GmbH. Leica TCS SP8. 2014. http://www.leica-microsystems.com/fileadmin/downloads/Leica%20TCS%20SP8%20STED%203X/Brochures/Leica%20TCS%20SP8-Brochure_EN.pdf. Accessed May 22, 2015.

9. Gonzalez RC, Woods RE. *Digital Image Processing*. Vol. 3 edition. Upper Saddle River, N.J: Prentice Hall; 2007.

10. Sieckmann F. *CAM Documentation, MatrixScreener 3 & 4*. Leica Microsystems CMS GmbH; 2013.

11. Seljebu A. Leicacam - control leica microscopes with python. 2015. https://github.com/

arve0/leicacam. Accessed May 22, 2015.

12. ISO. Tag image file format for image technology (TIFF/IT) ISO 12639:2004. May 2004.

13. ISO. Portable network graphics (PNG): Functional specification ISO/IEC 15948:2004. March 2004.

14. Duce D. Portable network graphics (PNG) specification (second edition). November 2003. http://www.w3.org/TR/PNG/. Accessed May 28, 2015.

15. Continuum Analytics. Numba. January 2015. http://numba.pydata.org/.

16. Continuum Analytics. Dask - parallel processing through blocked algorithms. 2015. http://dask.pydata.org/en/latest/. Accessed May 31, 2015.

17. Seljebu A. Leicaautomator - automatic scanning with leica SPX microscopes. 2015. https://github.com/arve0/leicaautomator. Accessed May 31, 2015.

18. Creaceed S.P.R.L. Prizmo. 2015. http://www.creaceed.com/prizmo. Accessed May 31, 2015.

19. Fiji is just ImageJ. May 2015. http://fiji.sc/wiki/index.php/Fiji. Accessed May 29, 2015.

20. Seljebu A. Microscopestitching - automatic merge/stitching of regular spaced images. 2015. https://github.com/arve0/microscopestitching. Accessed May 29, 2015.

21. Python Packaging Authority. User guide - pip 7.0.1 documentation. May 2015. https://pip.pypa.io/en/stable/user_guide.html. Accessed May 27, 2015.