



Franz Zehentner, BSc

Transfer of Sourcecode to JAVA for Implementation of Angular Dispersion Analysis Plug-in into ImageJ

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Biomedical Engineering

submitted to

Graz University of Technology

Supervisor

Dipl.-Ing. Dr.techn. Gerhard Sommer
Justyna Niestrawska, M.Sc.RWTH

Institute of Biomechanics

Graz, January 2018

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

Date

Signature

Contents

1	Introduction	1
2	Background and State of the Art	3
2.1	Histology of the Human Aorta	3
2.2	Motivation	4
2.3	State of the Art	4
3	Introduction to Java and ImageJ	9
3.1	Structure of ImageJ	10
3.2	Introduction to ImageJ	12
4	Mathematical Background	13
4.1	Fourier Transformation	13
4.2	Image Preprocessing	17
5	Theoretical Workflow	23
5.1	Preprocessing	23
5.2	Analysis	23
5.3	Results	24
6	Execution	25
6.1	Workflow	25
6.2	Summary	32
7	Manual	35
7.1	ImageJ	35
7.2	Plug-Ins	35
7.3	Angle Extraction	35
8	Verification	39
8.1	Generate Testimages	39
8.2	Verification Results	41
9	Conclusion	47
	Bibliography	51

Abstract

In this thesis a plug-in for the image processing tool ImageJ [1] is developed. The basis is a method published in [2]. This Angle Extraction Plug-In processes microscopic images of specimen with a distinct fiber angle distribution such as arterial walls. Before the actual analysis the image is preprocessed to remove possible contamination, which appears as bright areas on the image. A sequence of a median filter, a normalization step and again a median filter decrease the noise caused by removing the high-intensity areas. The next step is the Fourier transformation and analysis thereof. This is carried out by using properties of the Fourier transformation and the relationship between linear structures in the image and the intensities of the Fourier power spectrum. The user defines the parameters of the analysis and depending on those the results are calculated and visualized.

ImageJ is a widely used Java-based image processing and analyzing program. The idea is a basis framework that can be extended by developing plug-ins. The development was done using Eclipse Mars [3] as the environment. Before presenting the user with the user interface the requirement of a compatible image opened is checked and afterwards the choice of parameters and preprocessing steps are displayed. If there are multiple images to be analyzed the user can observe the progress bar for an estimation of the remaining time. At the end a result table with the normalized intensities values per angle-span and an image representing those values in a graphical way is presented to the user. The finalized plug-in is to be published and added to the collection at the official website ¹.

¹<https://imagej.nih.gov/ij/plugins/index.html>

Acknowledgment

I want to thank Prof. Holzapfel as the head of the Institute of Biomechanics for the opportunity to work in this interesting field and finish my study at the Technical University of Graz. I also want to thank Justyna Niestrawska for being very patient and helping me throughout the whole process of writing this thesis.

Raphael, Philip, Jürgen and I kept each other motivated during classes, helped if one got stuck and most importantly are good friends outside the classroom. This group made the past few years fun and I am glad that I could go to university with them.

Thank you Matthew Trombley for taking time to read this thesis and correct my mistakes. It took the worries of using wrong grammar and bad spelling away and gave me more confidence to hand it in.

Very special thanks go out to my family. They are not just caring for me financially but have been supporting my study in every way they could. Without their approval I would not have been able to be persistent and focused enough to overcome all challenges the past time brought up.

Last but most definitely not least I want to express my gratitude to my wife. Over the past couple years she encouraged me to pursue my study when I did not have the motivation to do it. Even after giving birth to our son Jakob she had enough strength to care about my degree and pushed me to finish it. Without her it would have been impossible to be who I am today: graduate of the Technical University of Graz, a happily married husband and a proud father.

1 Introduction

Cardiovascular diseases (CVDs) are the number 1 cause of death globally. In June 2016 the World Health Organization (WHO) estimated 17.5 million or 31% of the total deaths caused by CVDs [4]. The research in this area has the goal of decreasing those numbers.

One approach is creating models for various types of tissue [5, 6, 7]. Those models can include patient specific data by different types of parameters. To understand the underlying mechanics many different experiments like inflation and extension tests [8], uniaxial tension tests [9, 10], biaxial tension tests [11, 12] and triaxial shear testing [13] are conducted to gather enough data for creating constitutive models that describe the mechanical properties and behavior [14, 15]. Those models can be used to predict fatal incidents like aortic aneurisms [16].

Such experiments generate huge amounts of data that need processing and evaluating. To ensure a reproducible and fast procedure automated methods with clear parameters must be used. One part of the data are microscopic images of tested specimen. To be able to connect mechanical responses to the configuration of the mechanical behavior defining elements an analysis of their distribution has to be made. Responsible for the mechanical properties are most of the time fibrous structures like elastin or collagen [17]. The automated methods need to quantify fiber angle distributions for many images unattended and visualize the results as well as export the underlying numbers.

In case of the human aorta one of the main mechanical components collagen is arranged in different ways depending on the layer of the vessel. [5] already showed that if two fiber families tilted by an angle occur, the angle is a crucial parameter in the stretching behavior. Fig. 1.1 shows the influence of the angle γ regarding the relationship between circumferential stretch λ_Θ and the internal pressure p of a fiber-reinforced circular tube.

With the help of the automated methods for extracting fiber-angle distributions of real patient data, for example, a bivariate von Mises distribution [14]

$$p = ce^{b_1 \cos 2(\varphi-\alpha)+b_2 \cos 2\vartheta} + e^{b_1 \cos 2(\varphi+\alpha)+b_2 \cos 2\vartheta}, \quad (1.1)$$

using c as a normalization factor, b_1 as the in plane distribution descriptor, b_2 as the out of plane distribution descriptor, α as the angle between the circumferential direction and the fiber family and φ, ϑ as polar angle coordinates in and out of plane can be replaced. In Fig. 1.2 the effects of distribution descriptors can be observed. They have a significant

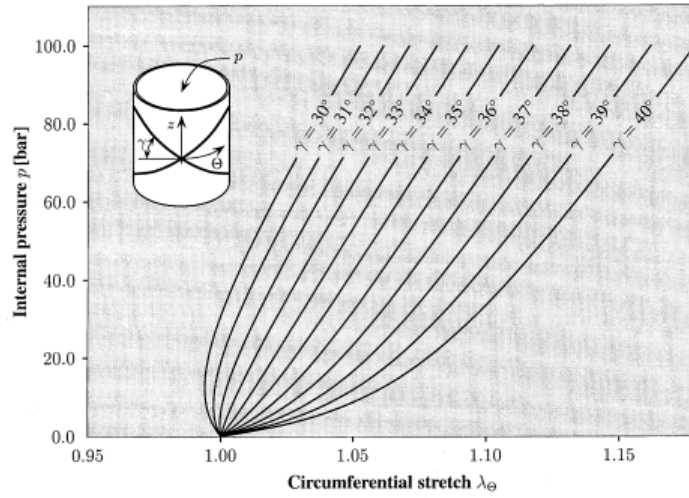


Figure 1.1: The fiber angle γ is a crucial parameter in solving the two-dimensional solution for the stretch-pressure diagram (circumferential stretch λ_Θ and pressure p) of a fiber-reinforced circular tube. In order to create realistic models a correct fiber-angle distribution of microscopic images of tissue samples is essential (taken from [5]).

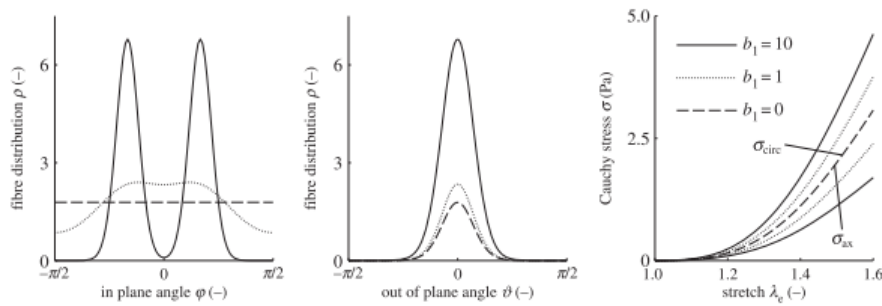


Figure 1.2: Formula 1.1 is an example of a model for the distribution of fiber-angles. The out of plane distribution descriptor b_2 was set to 5 and different values for the in plane distribution descriptor b_1 were chosen to show different angle profiles (in plane to the left and out of plane in the middle). $b_1 = 0$ would be isotropic angles in plane. To the right the stress-stretch diagram is shown where different angle distributions have a significant impact (taken from [14]).

impact on the stress-stretch response which again highlights the importance of the angle distributions for describing the mechanical behavior in a correct way.

2 Background and State of the Art

The Institute of Biomechanics at the Technical University of Graz does intensive research on soft tissue. One of the foci is the human aorta [18, 13, 19] and its mechanical properties in a healthy and pathological state [20, 14].

2.1 Histology of the Human Aorta

The focus of this thesis is extraction of angle distributions of microscopic images. Therefore the emphasis is set on the different types of fibers and their distribution in the aortic wall (see Fig.2.1).

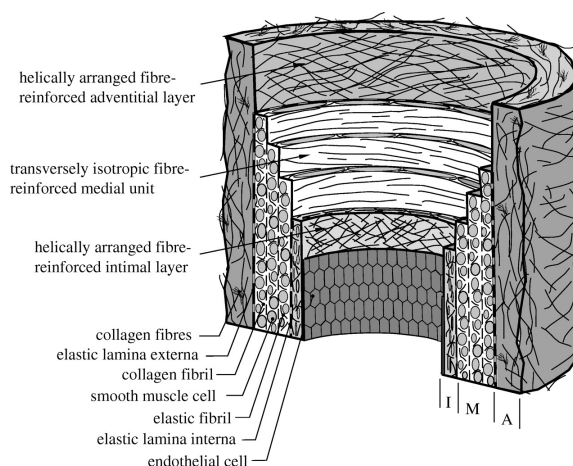


Figure 2.1: Schematic of an aorta and its collagen fiber distribution. In the adventitia (A) the collagen fibers are aligned in the axial direction, in the media (M) in circumferential direction and in the intima (I) an endothelial cell layer and with aging a more isotropic distribution can be found. [2]

2.1.1 Intima

The intima is made of endothelial cells arranged in one layer. In young and healthy humans, the intima does not contain any fibers and therefore samples of this kind do not need an analysis concerning fiber angle distribution. Pathological changes like atherosclerosis can

lead to changes and a deposition of for example collagen fibers. In this case microscopic images of the intima can be analyzed regarding fiber angles. Also aging processes change the histology of the intima in a manner where fiber angle analysis can be useful [18].

2.1.2 Media

The media is made of a network of collagen, elastin and smooth muscle cells. As a result, the media has the main influence on the mechanical behavior and was investigated thoroughly [21]. [17] shows a clear separation of the media by elastic laminae into different layers. Those layers have a distinct fiber distribution and microscopic images of dissected media specimen can be evaluated using an angle extraction tool. In the human aorta this fiber distribution can be seen as two fiber families with two dominant angles in plane to the circular direction. Therefore, these angles have a large influence on the mechanical behavior and need special attention when analyzing specimen of this section of the aorta [18].

2.1.3 Adventitia

The outer layer is called adventitia and is connecting the blood vessel to the surrounding tissue through the loose connective tissue around it. The fibrous part of the adventitia is made out of wavy collagen fibers arranged in helical structures and serve as protection against rupture because once they are stretched out the stiffness increases. Depending on the anatomical place and age of the patient the adventitia can vary in thickness [18].

2.2 Motivation

An automated method was developed by Andreas Schriefl to quantitatively analyze microscopic images regarding their fiber distributions in two dimensions [2]. This data can be used for parameter determination for computational modelling using fiber reinforced models [19]. A big part of this method was determining the angle distribution of multiple microscopic images.

The goal of this thesis is to develop a plug-in for ImageJ [1] (see Sec. 3.2) to analyze fiber distributions of multiple microscopic images quantitatively and present the user with the resulting data and visualize those results as well. At the end the plug-in will be published and made available at the ImageJ-Plug-In collection.

2.3 State of the Art

Before implementing a new plug-in a thorough investigation of the available tools for ImageJ was conducted and the most relevant ones are shown here.

2.3.1 ImageJ Angle Measurement

ImageJ has a simple angle measurement functionality already included. Three points are selected and the angle is shown in the status bar (see Fig. 2.2). It cannot be used for stack-images and the result cannot be stored or saved.

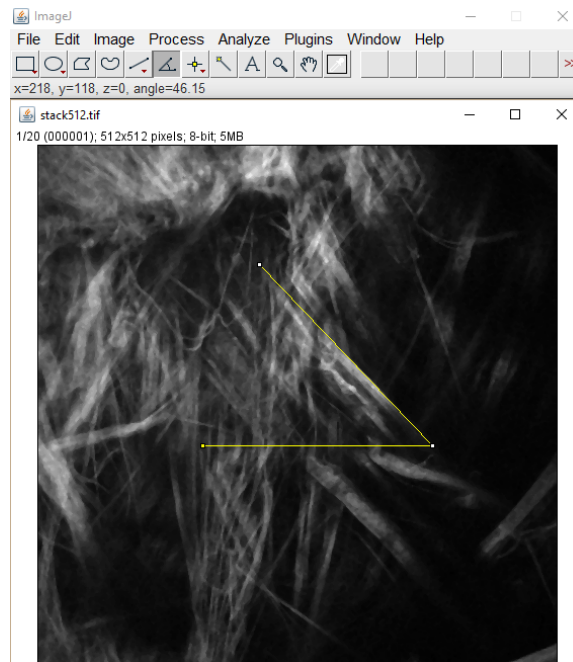


Figure 2.2: ImageJs angle tool allows for direct measurement of a single angle by selecting three points and the result is shown in the status bar of the main window.

2.3.2 OrientationJ

OrientationJ [22] is a plug-in for ImageJ (see Fig. 2.3) and has four functionalities:

- visual representation of the orientation
- quantitative orientation measurement
- distribution of orientations
- Harris corner detection

The software can use one of 6 structure tensors to calculate the orientation for every pixel. The user specifies also minimum energy in % relative to the maximum energy in the image and the coherency window in which the histogram of orientations is created. Because this is evaluated for each pixel more global restrictions like the minimum length of a fiber cannot

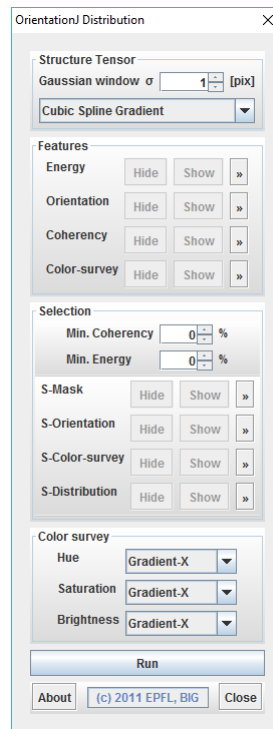


Figure 2.3: OrientationJ Distribution uses a specified structure tensor to calculate direction information. Different features can be visualized and restrictions regarding coherency and energy applied. The S-Distribution plot shows the orientation in degrees for all slices.

be taken into account. The distribution plot and the underlying data are for all slices. Single plots for single slices are not possible but needed for the application this thesis is focusing on.

2.3.3 Directionality - Fiji

Fiji [23] is an image processing package including ImageJ and many plug-ins and tools pre-installed. Directionality [24] is a plug-in for Fiji and has two analyzing methods implemented. Fig. 2.4 shows a test-image to the right and the result of the analysis to the left.

Fourier components analysis: This analysis splits the image into square parts and calculates the Fourier power spectra of those parts followed by a measurement of power using spatial filters as stated in [25].

Local gradient orientation: This local method calculates the gradient using a 5x5 Sobel [26] filter. A histogram is created by adding up corresponding square of gradient norms. This results in a histogram with the same dimensions as the Fourier component analysis. There is also a Gaussian fitting of the largest peak possible (see Fig. 2.4 in the histogram)

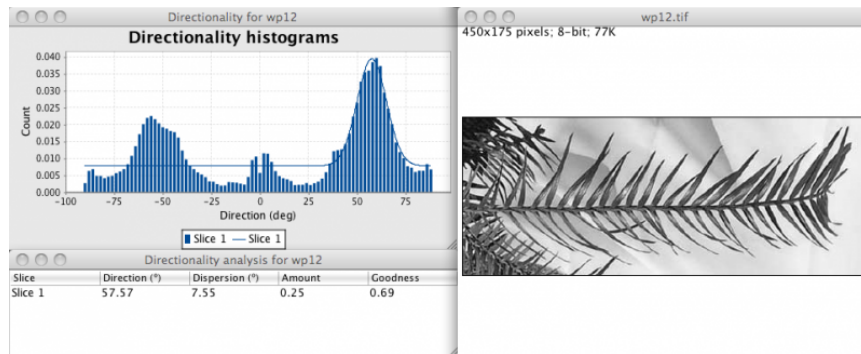


Figure 2.4: The test-image to the right was analyzed using Directionality. The resulting histogram can be seen to the left.

but the parameters are not displayed by default and can only be retrieved using an additional script similar to OrientationJ; length restrictions cannot be selected. Therefore, all structures present will contribute to the result. To analyze fibers in biological tissue this feature is crucial.

3 Introduction to Java and ImageJ

Java is a platform independent programming language and uses the concept of object-oriented programming (OOP). A class is a construct of attributes (also: variables) and sets of program instructions, called member variables and member functions. These properties are marked either `public` or `private`. `Public` variables and methods can be accessed by any other object or program instruction, whereas `private` properties can only be used by instances of this specific class. If many of them are created, they can be arranged in so called packages by their functionality or meaning. This results in a better overview if the program is quite large. The class itself does not contain any data and is used as a stencil for objects. This encapsulation suggests a strict boundary to other objects regarding activities and meaning. For example, a `Person`-object can have name, address, social security number and birthday as member variables. Once created, the person-object is now an instance of this class, defined by a unique variable-name and the data being stored. If we want to metaphorically ask, ‘How old are you?’, a member function has to be implemented which calculates the actual age. Here a major advantage of OOP is used: The age-calculation is coded only once in the object-stencil and is executed by using the individual data stored in the instance. Asking the question itself is called a ‘method call’. This is the tool for objects to communicate with each other and get access to other data and information.

Another principle of OOP is inheritance. It allows more general classes to be used for more detailed and specialized classes. In our previous example the `Person`-class could be used as a parent and a new `Athlete`-class as the child. Now an `Athlete`-object has also member variables like name, address etc. but can also have new, more detailed variables such as sports, personal bests, weight and height or new methods like running or throwing a javelin. Those are not available for a `Person`-object. Not just the variables are inherited but also the functions. If the programmer wants to change or adapt the behavior they can be altered but have to have the same name as in the parent object.

Especially if the software is being developed by different people or groups an agreement has to be found in order to get the code to work. So called interfaces are used to ensure code-compatibility. They are a special type of class, which can only contain constants and method signatures (just the header of a function without implementation). Since an instance does not have any actual code in it, a class can implement an interface to use the predefined variables and methods. [27]

3.1 Structure of ImageJ

ImageJ uses the concepts of OOP to utilize its highly flexible behavior and creates an easy possibility to extend the program. It is organized in 12 packages (see Tab. 3.1).

<code>ij</code>	<code>ij.gui</code>	<code>ij.io</code>	<code>ij.macro</code>
<code>ij.measure</code>	<code>ij.plugin</code>	<code>ij.plugin.filter</code>	<code>ij.plugin.frame</code>
<code>ij.plugin.tool</code>	<code>ij.process</code>	<code>ij.text</code>	<code>ij.util</code>

Table 3.1: The 12 packages containing all classes and interfaces for ImageJ. They are separated by functionality according to the self-explanatory package names.

3.1.1 Important Classes and Interfaces

The 5 most important classes and interfaces used in the Angle Extraction Plug-In with their methods are explained here.

ImageProcessor

The `ImageProcessor` is a superclass for 4 different processors that inherit a base set of variables and functions depending on their data-type.

- `ByteProcessor`
The pixel data format is `byte` and is stored in 8 bit numbers.
- `ColorProcessor`
The pixel data format is `int` and represents a 32 bit RGB image.
- `FloatProcessor`
The pixel data format is `float` and is stored in 32 bit floating-point numbers.
- `ShortProcessor`
The pixel data format is `short` and is stored in unsigned 16 bit numbers.

Using OOP, the function calls are the same for every processor type and carried out according to the underlying data format. This class is used to modify and analyze an image, because a direct access to pixel values is possible. The most important functions are:

- `convertToFloat`
This function returns a `FloatProcessor` of the current image and is used to convert color images into intensity (gray-value) images.
- `getFloatArray`
This method returns a two-dimensional array with the actual pixel data. One entry represents one pixel.

- `setFloatArray`
After modifying the image, this method is used to update the `ImageProcessor`'s pixel data.

ImagePlus

An `ImagePlus` object contains an `ImageProcessor` and several other variables with meta data (like bit depth and dimensions) regarding the image. This class is used to represent a single image and provides many useful methods such as:

- `draw`
This method opens a new window and shows the image-data stored in the corresponding `ImageProcessor`.
- `getProcessor`
This method returns a reference to the object's `ImageProcessor` for computations and modifications to the image data.
- `setProcessor`
After altering the data in the `ImageProcessor` this method replaces the old `ImageProcessor` and updates the image if it is shown in a window.

ImageStack

An image-stack is an array of images with the same dimensions and data-type. Access to individual images is possible via an index. This index can also be used to delete or insert single images to the stack.

ExtendedPlugInFilter

`ExtendedPlugInFilter` is an interface, which is used to develop plug-ins with an extended functionality such as asking the user for parameters, a preview function or a progress bar. It is a child of the `PlugInFilter` and has to have the following four functions implemented:

- `setNPasses`
Informs the plug-in, how many times the run-function is called.
- `showDialog`
In this function a user interface is implemented and asks for different parameters.
- `run`
This method is called once for each image of an image-stack or just once if a single image is active only. The actual image-processing is implemented here.

- `setup`
Once the filter is loaded the `setup`-function is called only once and defines the capabilities of the plug-in.

DialogListener

The interface `DialogListener` requires an implementation of `dialogItemChanged`, a function that is called every time a user changes parameters in the plug-in interface. This is used to apprehend the most recent values entered.

With the usage of OOP principles and instances of those classes and interfaces the aspired functionality of the plug-in can be achieved.

3.2 Introduction to ImageJ

ImageJ is a Java based, open source image processing and analyzing tool with basic functions already implemented. Once started the main window (see Fig. 3.1) appears. Besides being able to handle most of the common image formats (e.g. jpeg, tiff, gif, png). ImageJ is also able to work with image stacks. A stack is a collection of related images, like in the case of this Master Thesis microscopic images through the depth of a sample specimen.

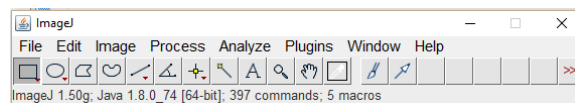


Figure 3.1: Main window of ImageJ.

3.2.1 Plug-Ins and Development

The structure of ImageJ allows it to be easily extended by plug-ins, scripts and macros. The main difference between a script or a macro and a plug-in is the need for compilations. Scripts (using JavaScript) and macros (Java-like language with a certain set of commands) are interpreted iteratively. A plug-in is a class derived from the `PlugIn/PlugInFilter` class and therefore has full access to ImageJ and Java APIs and is the most powerful and flexible choice of those three alternatives; they are written in Java [28].

The Angle Extraction Plug-In is implemented for ImageJ v1.50g [28] using Java Version 8 Update 73 [29]. The development environment was Eclipse Mars.1 Release (4.5.1) [3]. To set up a working environment an instruction from [30] was followed. It is setting up a project with the ImageJ source code and creating a new project (the plug-in) depending on it, so it can use all ImageJ classes and commands. During debugging the plug-in automatically shows up in the Plug-In menu.

4 Mathematical Background

The Angle Extraction Plug-In uses the Fourier transformation and some basic image pre-processing to calculate the results. The modalities used for this plug-in are described in this chapter.

4.1 Fourier Transformation

The Fourier transformation is named after Jean-Baptiste Joseph Fourier (1768 - 1830) and based on the Fourier series expansion

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cdot \cos(nx) + b_n \cdot \sin(nx)), \quad (4.1)$$

which states that every function is a sum of sines and cosines scaled by their coefficients

$$a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) dx, \quad (4.2)$$

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cdot \cos(nx) dx \quad (4.3)$$

and

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cdot \sin(nx) dx \quad (4.4)$$

with $n \in \mathbb{N}$ [31].

If the function is in the interval $[-L, L]$,

$$x \equiv \frac{\pi x'}{L} \quad (4.5)$$

and

$$dx = \frac{\pi dx'}{L} \quad (4.6)$$

are used to rewrite the Fourier series as follows:

$$f(x') = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \cdot \cos\left(\frac{n\pi x'}{L}\right) + b_n \cdot \sin\left(\frac{n\pi x'}{L}\right) \right). \quad (4.7)$$

Also the coefficients now read as follows:

$$a_0 = \frac{1}{L} \int_0^{2L} f(x') dx, \quad (4.8)$$

$$a_n = \frac{1}{L} \int_0^{2L} f(x') \cdot \cos\left(\frac{n\pi x'}{L}\right) dx' \quad (4.9)$$

and

$$b_n = \frac{1}{L} \int_0^{2L} f(x') \cdot \sin\left(\frac{n\pi x'}{L}\right) dx'. \quad (4.10)$$

The Fourier transformation uses Euler's formula

$$\cos(x) + i \cdot \sin(x) = \exp(ix) \quad (4.11)$$

with $x \in \mathbb{R}$, to represent the waves of the series expansion as exponential functions. As this representation uses complex exponents the Fourier coefficients also need to be of complex value.

Finally the Fourier transformation

$$\mathcal{F}(f(x)) = F(k) = \int_{-\infty}^{\infty} f(x) \exp(-i2\pi kx) dx, \quad (4.12)$$

k being the variable in the Fourier space, maps the time domain to the frequency domain (disassemble a signal in its spectrum), where the real and the complex part correspond to amplitude and phase of the analyzed function or signal, respectively ([32]).

4.1.1 Discrete Fourier Transformation - DFT

As there are no continuous functions in computer-aided calculations a discretization has to be made. Not only the continuous Fourier transformation has to be discretized, also the data has to be adapted if it is present in a continuous form. This happened implicitly due to the pixel-structure of an image stored on a computer. The step from a continuous integral to a discrete sum is carried out by using the Dirac delta function

$$\delta(a) = \begin{cases} 1 & \text{if } a = 0 \\ 0 & \text{if } a = \textit{else} \end{cases} \quad (4.13)$$

and its property

$$\int f(x) \delta(a - x) dx = f(a). \quad (4.14)$$

The final form of the DFT using N discrete samples

$$\mathcal{F}(f[k]) = F[n] = \sum_{k=0}^{N-1} f[k] e^{-i \cdot \frac{2\pi}{N} nk} \quad (4.15)$$

uses n as the spectral variable and $k = 0, 1, 2, \dots, N$ as the sample index. Square brackets [...] are used for discrete functions and round brackets (...) for continuous functions [33].

4.1.2 Fast Fourier Transformation - FFT

Especially for large data sets, a fast execution is crucial to the usability of a program. Therefore a reduction of the complexity of the problem is wanted. To achieve this goal the DFT is rewritten to

$$F[n] = \frac{1}{\sqrt{N}} \sum_{k=0}^N f[k] W_N^{nk}, \quad (4.16)$$

using

$$W_N^{nk} = e^{-i \cdot \frac{2\pi}{N} nk} \quad (4.17)$$

as a substitution for the exponential function.

Evaluating all possible W_N^{nk} for a fixed number of samples N shows that if N is a power of 2 the decimation-in-time-algorithm works best. It splits up the input data in 2 equally long parts

$$W_N^{2mn} = e^{-i \cdot \frac{2\pi}{N} (2mn)} = e^{-i \cdot \frac{2\pi}{\frac{N}{2}} mn} = W_{\frac{N}{2}}^{mn} \quad (4.18)$$

with even (substitute $k = 2m$) and odd (substitute $k = 2m + 1$) sample indices by rewriting

$$F[n] = \sum_{m=0}^{\frac{N}{2}-1} f[2m] W_{\frac{N}{2}}^{mn} + W_N^{m \cdot n} \sum_{m=0}^{\frac{N}{2}-1} f[2m + 1] W_{\frac{N}{2}}^{mn}. \quad (4.19)$$

If the factors W_N^{nk} are analyzed, one can see they repeat themselves and only N different values are needed. If a continuous signal is discretized using $N = 8$ samples, a total of 8 factors can be precomputed.

$$W_8^3 = e^{-i \cdot \frac{2\pi \cdot 3}{8}} = e^{-i \cdot \frac{3\pi}{4}} \quad (4.20)$$

and

$$W_8^{11} = e^{-i \cdot \frac{2\pi \cdot 11}{8}} = e^{-i \cdot 2\pi} e^{-i \cdot \frac{3\pi}{4}} = e^{-i \cdot \frac{3\pi}{4}} \quad (4.21)$$

show the periodicity of the factors with a period of N .

If taking the sign also in account, only $\frac{N}{2}$ values have to be calculated. This is due to the 2π -periodicity of the sines and cosines expressed as an exponential function (see Eq. 4.18 and Eq. 4.11). In Tab. 4.1 all 8 factors are precomputed. The angles in degrees correspond to angles inserted in the Euler identity (see Eq. 4.11).

The observation $W_N^k = -W_N^{k+\frac{N}{2}}$ (see Tab. 4.1) reduces the needed calculations by a factor of 2. This means if a continuous signal is sampled N times, only $\frac{N}{2}$ factors are required to be precomputed.

Including all those findings, a N -point DFT can be decomposed into two $\frac{N}{2}$ -point transformations

$$F[n] = G[n] + W_N^m H[n], \quad (4.22)$$

Factor	Exponential Function
W_8^0	$e^{-i \cdot \frac{2\pi \cdot 0}{8}} = 1$
W_8^1	$e^{-i \cdot \frac{2\pi \cdot 1}{8}} = e^{-i \cdot 45^\circ} = \frac{1-i}{\sqrt{2}}$
W_8^2	$e^{-i \cdot \frac{2\pi \cdot 2}{8}} = e^{-i \cdot 90^\circ} = -i$
W_8^3	$e^{-i \cdot \frac{2\pi \cdot 3}{8}} = e^{-i \cdot 135^\circ} = -\frac{1+i}{\sqrt{2}}$
W_8^4	$e^{-i \cdot \frac{2\pi \cdot 4}{8}} = e^{-i \cdot 180^\circ} = -1$
W_8^5	$e^{-i \cdot \frac{2\pi \cdot 5}{8}} = e^{-i \cdot 225^\circ} = -\frac{1-i}{\sqrt{2}}$
W_8^6	$e^{-i \cdot \frac{2\pi \cdot 6}{8}} = e^{-i \cdot 270^\circ} = i$
W_8^7	$e^{-i \cdot \frac{2\pi \cdot 7}{8}} = e^{-i \cdot 315^\circ} = \frac{1+i}{\sqrt{2}}$

Table 4.1: Factors used in the FFT. N (here: 8) samples can be thought of as N points equally distributed on the unit circle. The angles are inserted in the Euler identity (see Eq.4.11) and evaluated. Using this and the abstract unit circle one can see: A shift of 180° or π is just a multiplication by -1 and this eliminates the need for $\frac{N}{2}$ (here: 4) factors to be computed.

with $G[h]$ being the DFT of the even input and $H[n]$ the DFT of the odd input data (see Eq. 4.19).

If this algorithm is applied recursively and the number of samples N is a power of 2, the FFT is calculated by solving 2-point DFTs at the highest stage of recursion ([34]). If comparing the complexity of the DFT to the FFT by comparing the number of complex multiplications one can see, that for the DFT N^2 multiplications are needed. Due to the splitting operation the FFT has only $\frac{N}{2}$ multiplications. The number of splits possible is $\log_2(N)$, therefore a total of only $\frac{N}{2} \log_2(N)$ complex multiplications are needed. The other operations are computationally inexpensive complex additions.

If the Fourier Transformation is applied on an image, N represent the number of pixels, therefore only dimensions of a power of 2 can make use of the FFT. Otherwise the much slower DFT has to be used.

4.1.3 DFT and FFT in Image Processing

Before applying the DFT or FFT, an adjustment has to be made to include the second dimension ([35], shown in Eq. 4.23).

$$\mathcal{F}(f[k, l]) = F[m, n] = \frac{1}{\sqrt{MN}} \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f[m, n] \cdot e^{-i \cdot 2\pi(\frac{mk}{M} + \frac{nl}{N})} \quad (4.23)$$

$f[m, n]$ represents a single pixel in an image of dimensions $M \times N$ and $F[k, l]$ is the corresponding entry in the discrete 2D Fourier spectrum. Separating the exponential term,

allows also separating the double-sum and therefore the 2D transformation splits into two 1D transformations. Especially for image processing, this gives a computational advantage: If only one dimension is a power of 2, this Fourier transformation can be calculated using the FFT, while the other dimension needs the slower DFT.

Either result of the FFT or DFT leads to a complex-valued 2D matrix. The data can be visualized in different ways. The most common one is to take the absolute value. Because different entries in the Fourier data correspond to different frequencies, the mean-value of the image lies at $F[0, 0]$. Most of the time this will be the largest value by far. To visualize this spectral data, the logarithm of the power spectrum (multiplying the Fourier data by its complex conjugate: $P[m, n] = F[m, n]F^*[m, n]$) value is taken and can be interpreted as intensity in a gray-scale image.

The power spectrum seen as a gray-scale image contains information about the frequencies that occur in the original image. Using the term directions instead of frequency while talking about image processing makes it better understandable. High values of the power spectrum at specific points are shown as bright areas. Therefore those bright areas represent structures of a specific size and angle.

Fig. 4.1 shows a following findings:

- Different lengths of structures at a certain direction show up at different positions in the power spectrum. Longer structures are further away from the center but are on the same line if the angle stays the same
- The visualized power spectrum is shifted by 90°
- The center represents the mean value of the image and has the largest value

This interpretation is the basis of the angle distribution analysis the ImageJ plug-in.

4.2 Image Preprocessing

Before the actual angle analysis certain steps have to be executed to improve the results.

4.2.1 Maxima Removing

The data to be analyzed in this thesis are microscopic gray-scale images. It is possible dust particles or similar contamination get on the specimen as shown in Fig. 4.2 to the left. These contaminations show up as bright spots and are taken in account during the Fourier transformation. In order to exclude those artifacts a threshold is set by the user and all pixel values above are set to the lowest occurring value (see Fig. 4.2 to the right). If it would be set to 0, the normalization-step (see Subsec. 4.2.3) would not detect the lowest

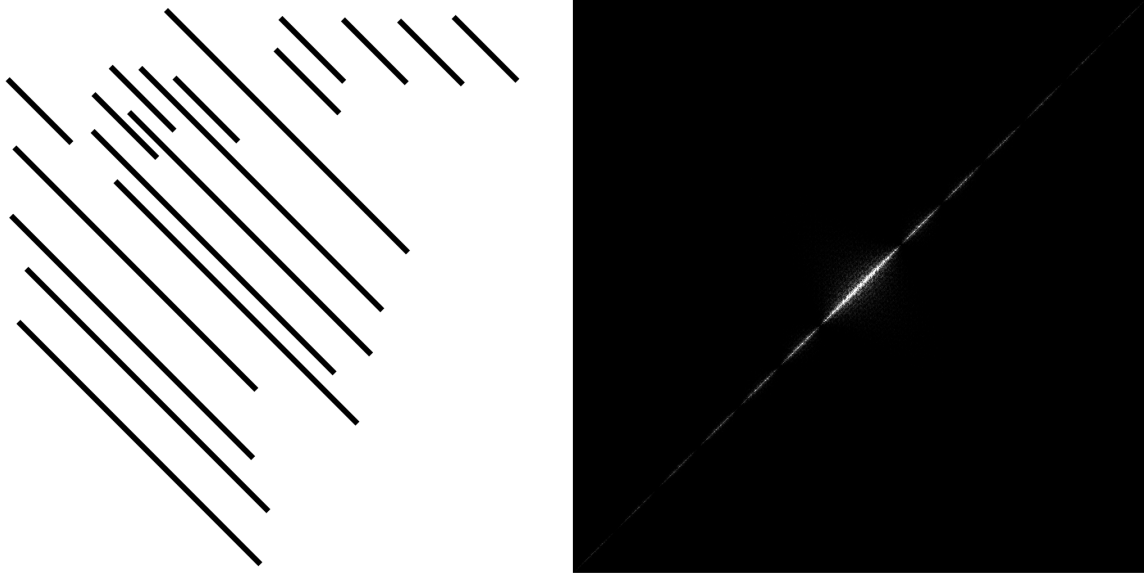


Figure 4.1: A test-image with its corresponding logarithmic scaled Fourier power spectrum.

intensity of the original picture and calculate a wrong normalization factor as described in Eq. 4.24). Therefore, those artifacts do not influence the result of the Fourier transformation.

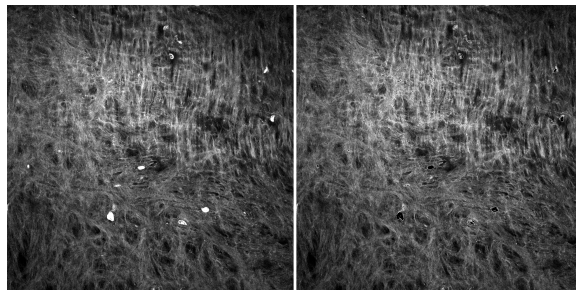


Figure 4.2: The bright white spots on the left side are artifacts due to contamination of the specimen. They are filtered out by the maxima removing step of the image pre-processing. Here a threshold of 250 was set and pixels with a higher intensity are replaced with the lowest occurring intensity in this image.

4.2.2 Median Filtering

Additionally to removing the maxima a 3×3 median filter is applied (see Fig. 4.3). It takes all pixels within its quadratic window with odd side length, sorts them ascending according to their value and replaces the current pixel with the median value. At the border where the

window cannot be filled the image is mirrored along the edge and those values are used. This filter handles outliers very well, especially if for example in a brighter area a few single pixels are set to the minimum value due the maxima removing step. The median filter corrects those outliers and the area does not lose as much information, hence only a few spots are above the threshold. In the workflow of the Angle Extraction Plug-in the median filtering is applied after removing the maxima. Fig. 4.4 shows a specimen with its maxima removed (left) and the output after the filtering operation (right).

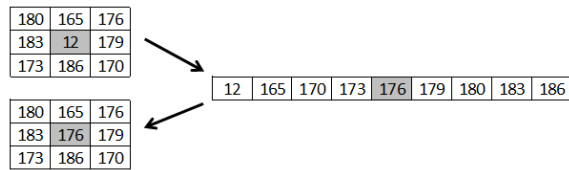


Figure 4.3: 3x3 median filter applied on the center pixel.

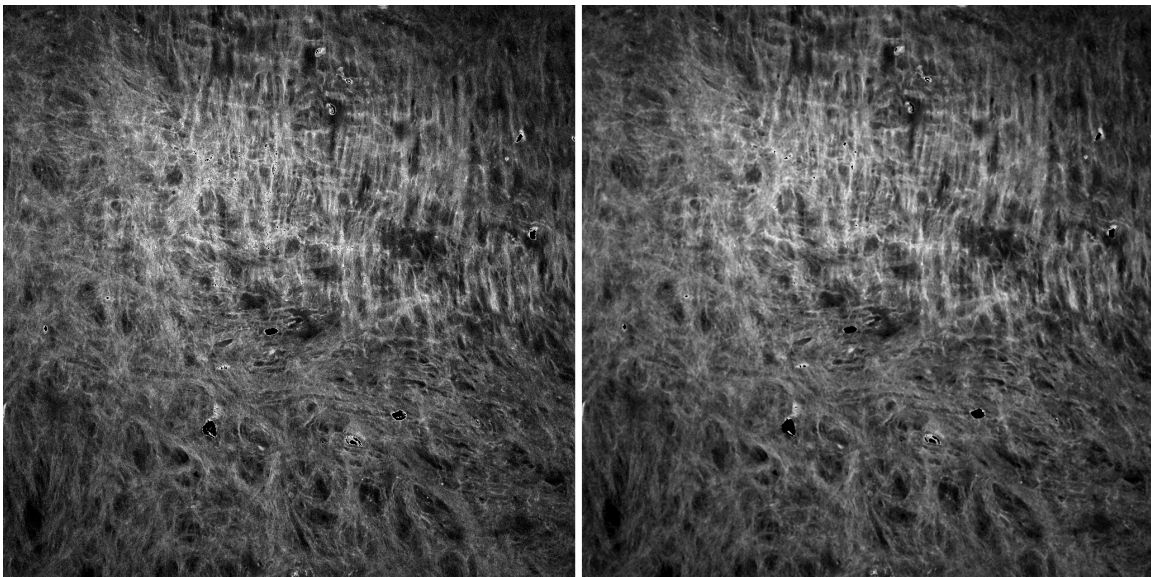


Figure 4.4: Input for the median filtering step is here the image after removing the maxima (seen on the left). After the median filtering less salt-and-pepper-like noise is present.

4.2.3 Normalization

To use as many gray-shades as possible at the end of the preprocessing steps a normalization to an 8bit range (0-255) is carried out using

$$n_f = \frac{255}{p_{\max} - p_{\min}} \quad (4.24)$$

to calculate the normalization factor n_f and

$$p_{\text{new}} = (p_{\text{old}} - p_{\min}) \cdot n_f \quad (4.25)$$

as the normalization step (p_{new} and p_{old} are the normalized and old pixel values respectively). Tab. 4.2 shows an example of a calculation for an image.

	old pixel value	new pixel value
minimum	12	0
maximum	250	255

Table 4.2: Using Eq. 4.24 the normalization factor $n_f = \frac{255}{238}$ and the new pixel values are calculated as shown in Eq. 4.25.

4.2.4 Cosine Mask

Because the Fourier Transform does not take spatial information into account and assumes periodicity a translation of the image as seen in Fig. 4.5 should not have any effect on the result. Due to the non-symmetric nature of the image artifacts occur in the Discrete Fourier Transform [36]. To avoid those discontinuities a symmetric cosine-based mask is applied to the image. The interval of $[-\frac{\pi}{2}, \frac{\pi}{2}]$ is an equally-spaced sampled with the number of samples being the number of pixels of the side length of the image and put into a vector. This vector is multiplied and transposed with itself and the result is a cosine-based mask with the dimension of the image (Fig. 4.6 to the left). Due to the nature of the cosine function it has a range of $[0, 1]$ with the maximum value being at the center and 0 at the borders. After a pixel-wise multiplication of the mask and the image the basis for further analysis is set (Fig. 4.6 to the right).

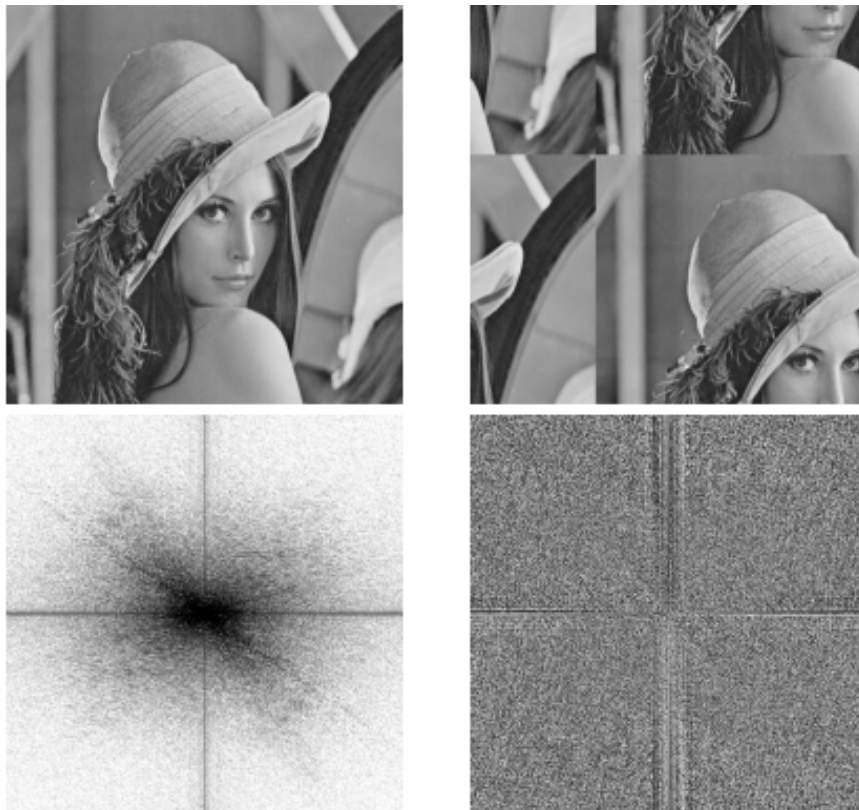


Figure 4.5: A test-image (top left) is assumed to be periodic for using the Discrete Fourier Transform. If an actual translation is carried out (top right) edges are created which show up as a very distinct cross in both components of the Discrete Fourier Transform (bottom left and right).

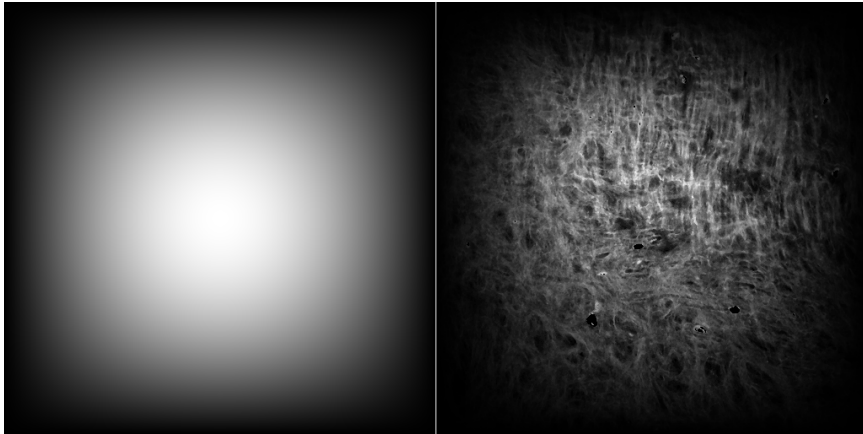


Figure 4.6: This mask on the left is applied to the image before the Fourier transform to avoid cross-like artifacts with high energy in the Fourier transform due to the non-periodic image. It has a value of 1 at the center and decreases to 0 at the borders according to a half-period of a cosine function. A pixel-wise multiplication with the image to be transformed is carried out afterwards. The image to be analyzed looks like the one to the right.

5 Theoretical Workflow

Because of the OOP and Java principles and setup discussed in Ch. 3 the actual workflow is not represented clearly in the program code. This chapter explains the process from a procedural viewpoint. The use of the mathematical methods of Ch. 4 in order to achieve the desired result of an accurate angle distribution analysis is also pointed out.

5.1 Preprocessing

If the user selects preprocessing steps (see Fig.7.2) they are carried out in the listed order. The cosine-based mask is a crucial part of the preprocessing. Therefore, it is not presented as a choice and carried out every time. The execution follows Sec. 4.2. Assuming all steps are selected the images are handled as followed:

1. Pixels in the image with an intensity larger than the user-chosen threshold are replaced by the smallest occurring intensity.
2. The image is smoothed with a 3x3 median filter to reduce possible speckle noise caused by the previous step.
3. To use the whole intensity values of $[0, 255]$ the image is normalized to this range.
4. At the end a 3x3 median filter is applied to smooth the image more. The median filter does not introduce new intensity values. Therefore, a second normalization is not necessary.
5. A cosine-based filter is applied to focus the analysis on the center of the image.

5.2 Analysis

After the preprocessing steps the actual angle analysis starts (see Sec. 4.1):

1. Execute the Fourier transformation and use the power spectrum for further analysis.
2. The pixels of the power spectrum are assigned to their corresponding angles.
3. Based on the user input angle wedges (see Fig.6.8 for an example) are created and pixel intensities are summed up within those wedges.
4. For comparable results the wedge-intensity sums are normalized.

5.3 Results

There are two ways the result is displayed to the user. First, a result table with the numerical values is shown. From there (see Fig. 5.1) the user can see the normalized intensity values of the wedges. One row represents one image and a column contains all summed intensities of the corresponding angles. This data can directly be saved to a file for further usage.

	0	10	20	30	40	50	60	70	80	90	100	110
1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
5	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
10	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
11	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
12	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
13	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
14	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
15	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
16	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
17	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
18	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
19	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
20	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Figure 5.1: The result table of a stack of 20 images with an angle width of 10° . One row represents one image and a column corresponds to the angles of this wedge. The wedges continue to 180° .

The second representation of the result is the graphical interpretation of the result table. In Subsec. 6.1.2 the normalized intensities are represented as an image for easy visual interpretation (see Fig. 8.6).

6 Execution

In this chapter a detailed pseudo-code of the whole plug-in is presented. The Angle Extraction Plugin has a total of 9 Java-methods, out of which 4 are private and 5 public (see Fig. 6.1). Private methods can be accessed within the code of the plug-in itself. Public methods can be accessed by external code, in this case the ImageJ program.



Figure 6.1: An overview of the Java-Methods of the Angle Extraction Plug-in. Private methods are used by the plug-in itself only and public methods can be accessed by external Java code (e.g. ImageJ). This figure does not say anything about the execution order but the accessibility of different methods.

6.1 Workflow

In Fig. 6.2 a flowchart is shown. `setNPasses` does not appear, because it is only used by ImageJ to inform the plug-in how many times the `run`-method will be called. After confirming the requirements by `setup` and checking for valid parameters by `showDialog`, `run` analyzes each image using `removeMaxima`, `normalize` and `generateBandpass`. Finally, `setup` is called again because of the `FINAL_PROCESSING`-flag and opens the result table and result image.

This workflow represents the different stages of the plug-in from a programmer's view. The processing-chain regarding the image analyzing only is explained in Sec. 5.

6.1.1 Class Variables

Because a plug-in is an implemented Java-class the class variables are available in all methods. Following variables are used:

- `private static int FLAGS = DOES_ALL | DOES_STACKS | FINAL_PROCESSING;`

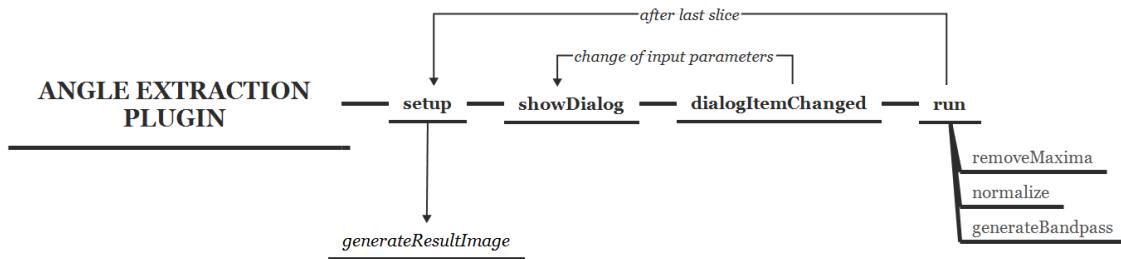


Figure 6.2: After the user started the plug-in the `setup`-method at first checks if an image of compatible type is opened (using the flags explained in Subsec. 6.1.1). If successfully started the plug-in will present calling `showDialog` the possibility to change the default parameters. It calls `dialogItemChanged` if the user enters some data to update the variables the code is working with and check for valid values. After confirming the parameters, `run` is called for each slice of the image-stack or just once if only one image should be analyzed. Within `run` the actual processing and analyzing is happening. At the end `setup` is called one last time to generate the result table and the result image.

According to the flags the `setup`-method checks if the requirements for starting the plug-in are met.

DOES_ALL: plug-in handles all types of images

DOES_STACKS: plug-in handles stacks of images

FINAL_PROCESSING: setting this flag results in a last call of `setup` with `arg="final"`

- `float pi`: used for creating the cosine mask
- `ResultTable resTable`: This is a provided data-type by ImageJ and uses strings as column headers and numbers as entries. One column represents intensities per angle-wedge through all images and one row represents an image with all its wedges.
- `int threshold`: radius of low pass filter for Fourier transformation
- `int wedgeWidth`: width of wedge in $^{\circ}$ (degrees)
The value has to be an integer-fraction of 360° to achieve full coverage.
- `String widths[]`: array, which contains integer fractions of 360° , used do fill drop-down selection
- `boolean[] preprocessing`: Array with boolean values to decide which pre-processing steps are executed in the following order: remove maxima, despeckle 1, normalize, despeckle 2
- `String[] prepStrings`: array, which contains strings to show in the GUI

- `int width`: width of the current image processed
- `int height`: height of the current image processed
- `int iCoord`: column-coordinate of maximum value in power spectrum of the Fourier transformation
- `int jCoord`: row-coordinate of maximum value in power spectrum of the Fourier transformation
- `float[][] bandpass`: Binary mask for applying bandpass to the Fourier transformation
- `float[][] angleMatrix`: each entry contains the angle in degrees with respect to the maximum of the Fourier transformation
- `int maxThreshold`: value for thresholding in `removeMaxima`

6.1.2 `public int setup(String arg, ImagePlus imp)`

This subsection will discuss the method `setup`. For clarity reasons some operations are not implemented directly within one of the public methods. They are separately coded in private functions, which are explained as well.

This method is called by ImageJ when the plug-in is loaded. `arg` can be empty, but after the last image processed `setup` is called again with `arg="final"`. This is used to decide when the result table and `generateResultImage()` is called to create the final image. `imp` is the current image being processed.

`private void generateResultImage()` This private function uses an already implemented functionality to create an image by using data from the result table. Because one entry in the result-table is the intensity of a wedge-span, resizing is necessary. Hitting `Ctrl+E` opens ImageJs scaling tool and the required size can be created (see Fig. 6.3). If interpolation is used in the resizing process, the result will not be the correct data anymore, because in the several interpolation options neighboring pixels and therefore neighboring data influences each other. The number of wedges stays the same but the number of pixels per wedge in the result image is increased. The intensity of a wedge must not change; therefore, it is crucial that no interpolation is executed.

By default, the result image has the Fire-look up table (LUT) applied. A LUT does not change the data itself, but contains information how to map different values to different colors. Depending on the current version, ImageJ has already a few LUTs installed (see Fig. 6.4).

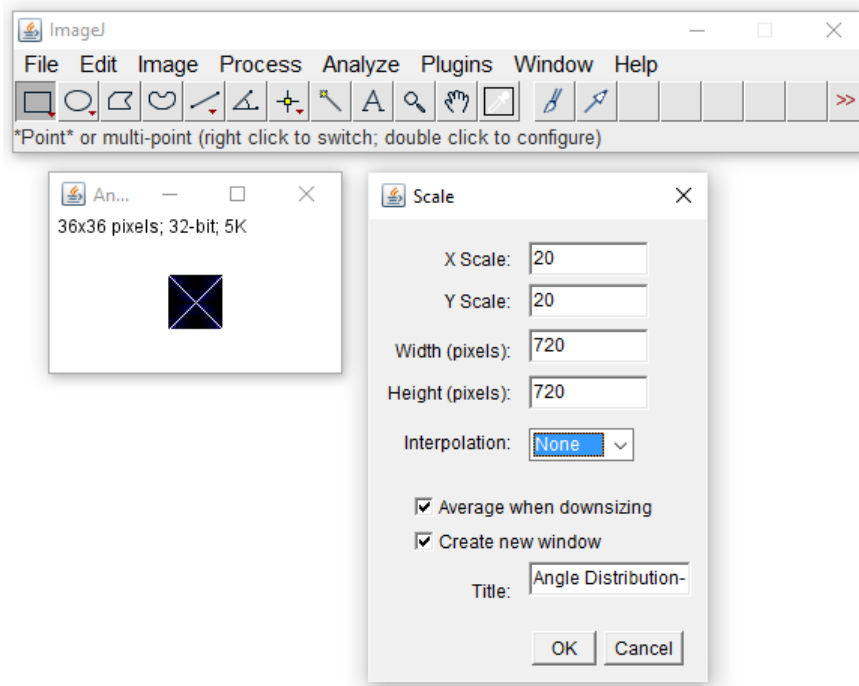


Figure 6.3: The 36 by 36 result image is composed of 36 images (rows) and $\frac{180}{5} = 36$ wedges. The scaling tool (Ctrl+E) allows to resize it. To avoid falsifying the intensities the interpolation option "None" has to be selected. Otherwise in the upscaling process intensities from neighboring pixels influence each other.

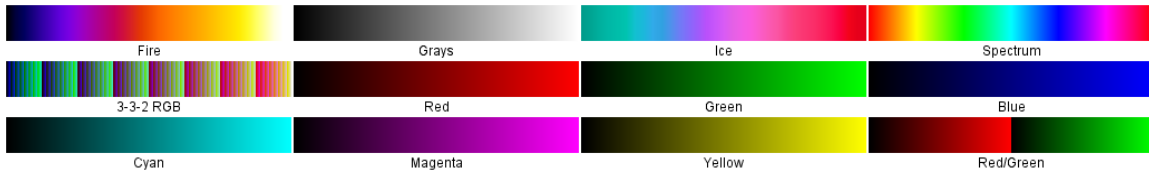


Figure 6.4: Selection of LUTs in ImageJ. They do not interact with the data itself, but represent the mapping operation from data-value to shown color.

6.1.3 `public int showDialog(ImagePlus imp, String command, PlugInFilterRunner pfr)`

This subsection describes the method `showDialog`. After setup is finished the `showDialog`-method is called to build the GUI. `imp` is the current image, `command` can be used to find out what command started the plug-in and `pfr` is needed if a preview function is required. Since the preview requires an execution of the `run`-method, which can be quite time consuming, it is not used here. If it would be implemented, every time the user changes a parameter a new analysis is started.

The `GenericDialog`-class has already implemented the features to build a GUI and an instance of this class is set up to build a user interface (see Fig. 7.2) and automatically invokes `dialogItemChanged` if any changes are made.

public boolean dialogItemChanged(GenericDialog gd, AWTEvent e)

This method updates the variables `threshold`, `maxThreshold`, `wedgeWidth`, `preprocessing[]` and checks for valid values of `maxThreshold` (0 – 255). `gd` is used to get access to the shown user interface and `e` describes the event that caused the call of `dialogItemChanged`.

6.1.4 public void run(ImageProcessor ip)

In this subsection the method `run` is discussed. Here the actual image processing is carried out. Its only parameter `ip` allows the developer to access, transform and analyze the image's data. Fig. 6.5 shows the workflow and structural setup

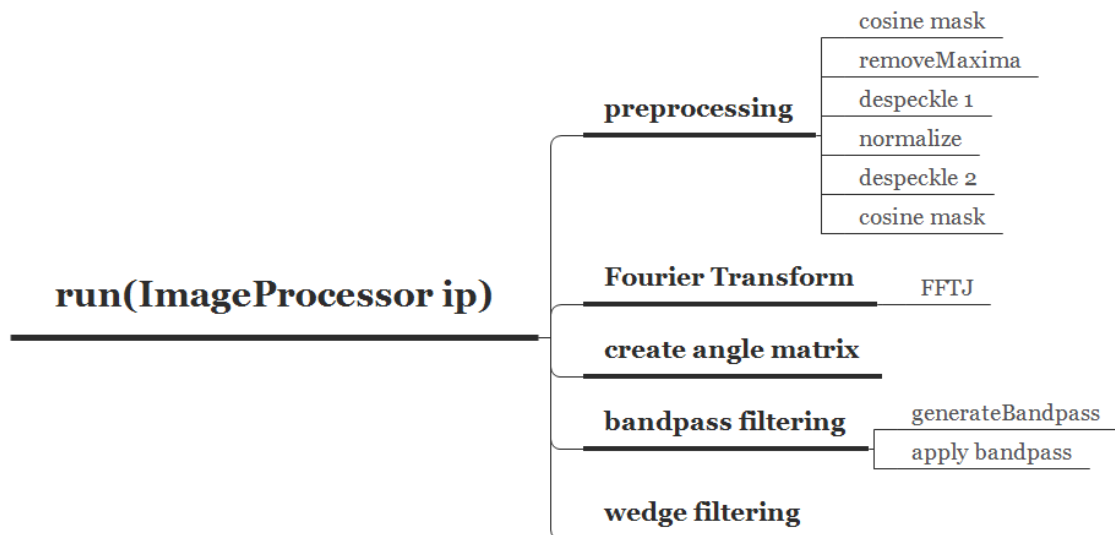


Figure 6.5: Workflow of the `run`-method. For detailed explanation of functional blocks please see Sec. 6.1.4.

Preprocessing Depending on which tasks the user selected following operations can be executed:

- **Cosine Mask**
To avoid artifacts in the Fourier transformation the image is multiplied with a mask based on the cosine function. It is calculated by discretizing a half-period of a cosine function in two dimensions (see Fig. 4.6).

- **Remove Maxima**
Each pixel is checked individually against `maxThreshold` and set to the lowest occurring intensity in the image.
- **Despeckle 1**
`RankFilter`-class is used to apply a standard 3 by 3 median filter as explained in Sec. 4.2.2.
- **Normalize**
After finding the maximum intensity in the image the normalization-factor is calculated and a pixel-wise multiplication is performed to use all values between 0 – 255.
- **Despeckle 2**
analogous to Despeckle 1

Fourier Transform Because ImageJ and its plug-ins are open source FFTJ [37] is included in the Java project to get access to its functionality. After the Fourier transformation the power spectrum of the Fourier transform of the current image (`ip`) is available in a two-dimensional array with data type `float`.

Angle Matrix The wedge-filtering process needs to know every angle of every pixel in relation to the center of the Fourier transform. The center coordinates (`iCoord`, `jCoord`) are found by looking for the highest value of the power spectrum. With this information two matrices with image dimensions are build containing the absolute x and y distances to the center respectively.

Using

$$\text{atan} \left(\frac{y_{i,j}}{x_{i,j}} \right) \cdot \frac{180}{\pi}, \quad (6.1)$$

where $y_{i,j}$ and $x_{i,j}$ are the absolute distances to the center of the current pixel investigated, the angle matrix itself is built. Since there are no negative angles allowed 360 is added to those values. Each entry contains now the angle with respect to the center of the power spectrum. In Fig. 6.6 the angles are shown in grayscale where 0° is black and 360° is white. The angles used in the result table and the result image correspond to this angle matrix.

grayscale where 0° is black and 360° is white. The angles used in the result table and the result image correspond to this angle matrix.

Bandpass Small structures do not represent the structures of interest and have to be filtered. They appear closer to the center of the Fourier transformation. To do this a thresholding-operation is executed. The user specified `threshold` as the radius in % of the image side length. A circle around the center with this radius is ignored during wedge filtering. To ensure an equal area for all wedges, another circle with the radius of the shorter image

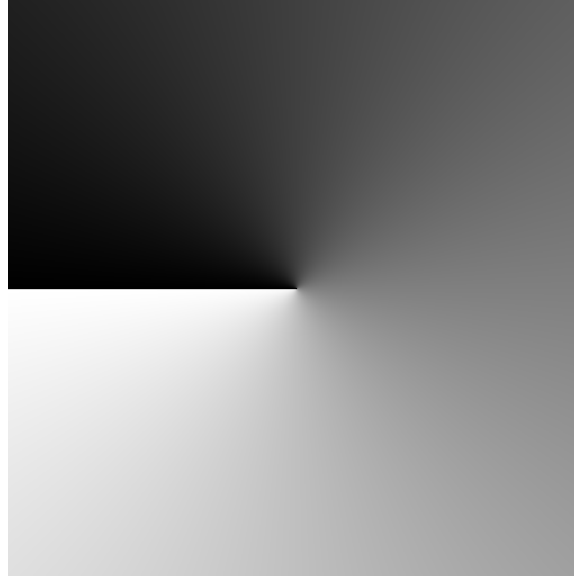


Figure 6.6: This image visualizes the angle matrix used in the wedge filtering process. Each pixel-intensity represents an angle at the current position. 0° maps to black and 360° to white.

side length is calculated. Everything outside this circle will be ignored as well. The inner circle is in the Fourier domain a low-pass filter and the outer circle a high-pass filter. With these combined a bandpass filter is created. To apply the bandpass a simple pixel-wise multiplication is sufficient, because the ignored areas in the mask are set to 0 and those areas relevant to the analysis to 1 (see Fig.6.7).

Wedge Filter This step is crucial to the angle analysis and uses the values of the power spectrum to establish a relationship between the Fourier data and angles of structures in the image. Instead of creating $\frac{360}{\#wedges}$ masks the data from the angle matrix is used to determine which pixel will be assigned to which wedge. Fig. 6.8 shows a wedge with an angle span of 5° .

The formula

$$index = \text{mod} \left(\left\lfloor \frac{angleMatrix_{i,j}}{wedgeWidth} \right\rfloor, \frac{360}{wedgeWidth} \right) \quad (6.2)$$

calculates the index in the intensity array. $angleMatrix_{i,j}$ is the angle of the pixel with the coordinates $[i, j]$ and $wedgeWidth$ describes how many degrees are combined. The example with a wedge width of 5° and the pixel with an angle of 276°

$$index = \text{mod} \left(\left\lfloor \frac{276}{5} \right\rfloor, \frac{360}{5} \right) = \text{mod} (55, 72) = 55 \quad (6.3)$$

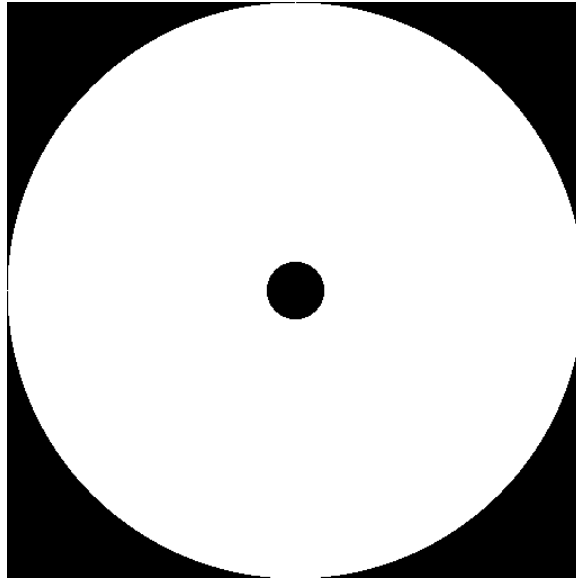


Figure 6.7: The inner circle has a radius of `threshold` and the outer radius is the value of smaller side of the image. This mask is multiplied pixel-wise with the power spectrum and the basis for wedge filtering is created.

would add the intensity of this pixel to the 55^{th} wedge of the intensity array. In the image it cannot be differentiated if a structure or fiber is rotated either 96° or 276° . Therefore, wedges, which are 180° apart contain information regarding the same structures and can be added. In example 6.3 the 55^{th} ($275^\circ - 280^\circ$) wedge is added to the 19^{th} ($95^\circ - 100^\circ$). At the end for each image the intensities are normalized to a range of $[0, 100]$, where 100 represents the maximum intensity for a wedge. Therefore, this wedge represents the most occurring angles in the image. A schematic overview is shown in Fig. 6.9. This data is added to the `resultTable`, which is later used to generate the visual output.

6.2 Summary

The ImageJ Angle Extraction Plug-In is capable of analyzing and visualizing angle distributions of any kind using Fourier transformation. The wedge-width is the most crucial parameter. In Fig. 6.10 the left structure is made of small parts at different angles. These parts will contribute to different areas in the Fourier spectrum. If the wedge is too small, intensities are mapped to the wrong angles for the total fiber. The right straight line will be captured by the corresponding wedge only, because this particular line will show up in one place of the Fourier transformation only.

As shown in Subsec. 6.1.4 the central part of the power spectrum is not used for the analysis. Intensities close to the center are caused by very small structures like the one marked

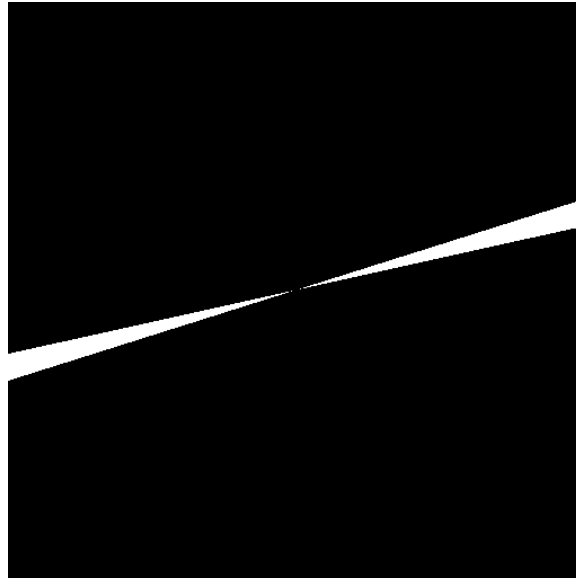


Figure 6.8: Depending on the angle-span currently analyzed a wedge-mask is created figuratively and multiplied with the Fourier data pixel-wise. The remaining intensities belong to the current angles evaluated. This step is left out in the implementation, because the affiliation of a pixel to its angle can be calculated by using the angle matrix directly.

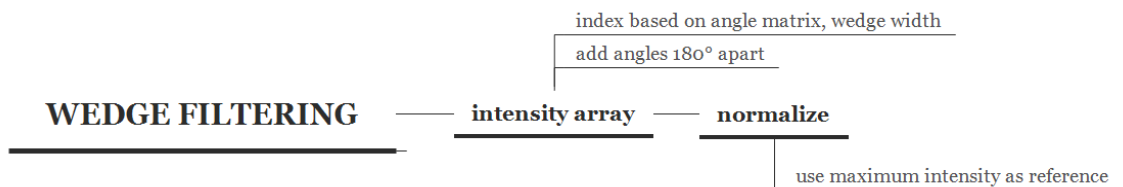


Figure 6.9: The intensity for each pixel of the power spectrum is added to the index, which is calculated based on the angle and wedge width (see Eq. 6.2). Before storing the results, a normalization based on the maximum intensity to a range of $[0, 100]$ is executed.

by the red rectangle in Fig. 6.10. This threshold determines the length of the structures to be filtered out and therefore more realistic results regarding the angle of the whole fiber and not the individual pieces can be achieved.

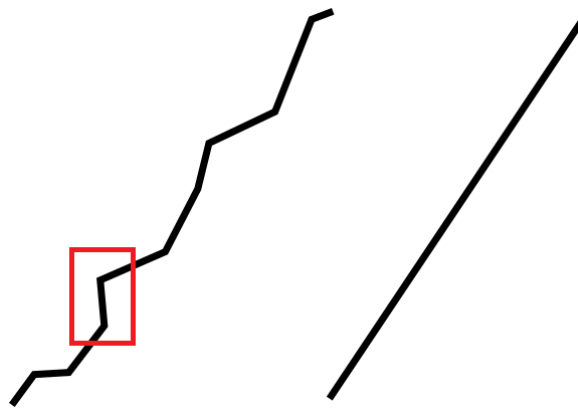


Figure 6.10: Because the left structure is made of different smaller structures at varying angles it is going to show at more than one place in the Fourier spectrum. The right line with only one specific angle is going to be at a certain spot in the power spectrum and will be captured by the right wedge. If the wedge-width is too small, it is possible that different parts of the left structure contribute to wrong wedges. The marked segment will show up close to the center of the power spectrum and can falsify the intensity for this angle, even though the whole fiber is rotated in a completely different way.

7 Manual

In this chapter the handling of ImageJ, the Angle Extraction Plug-In and plug-ins in general are explained.

7.1 ImageJ

ImageJ is an open-source software based on Java. Therefore, the current Java Runtime Environment (JRE) has to be installed. Additionally, if the user wants to develop and modify plug-ins, the Java Software Development Kit (JDK), which already includes the JRE has to be installed instead.

Assuming a basic usage of ImageJ is wanted and no further programming will take place (for developing manual see next section). In this case depending on the operating system of the computer, the correct files have to be downloaded from the website ¹.

7.2 Plug-Ins

Plug-ins are either `.java` or already compiled `.class`-files. It is recognized as a plug-in by a `_` (underscore) in its file name. In order to add a new plug-in to ImageJ two different methods can be used:

- Copy `plugin_.java`-file to a subfolder in the ImageJ/Plugins
select the copied file via Plugins → Compile and Run... command (see Fig. 7.1 on the top).
- Copy `plugin_.class` to a subfolder in the ImageJ/Plugins folder

After either method ImageJ needs to be restarted and the plug-in will appear in the menu (see Fig. 7.1 to the bottom).

7.3 Angle Extraction

To start the plug-in from the menu an image or an image-stack has to be opened. Once it is started the input mask shows as in Fig. 7.2.

¹<https://imagej.nih.gov/ij/download.html>

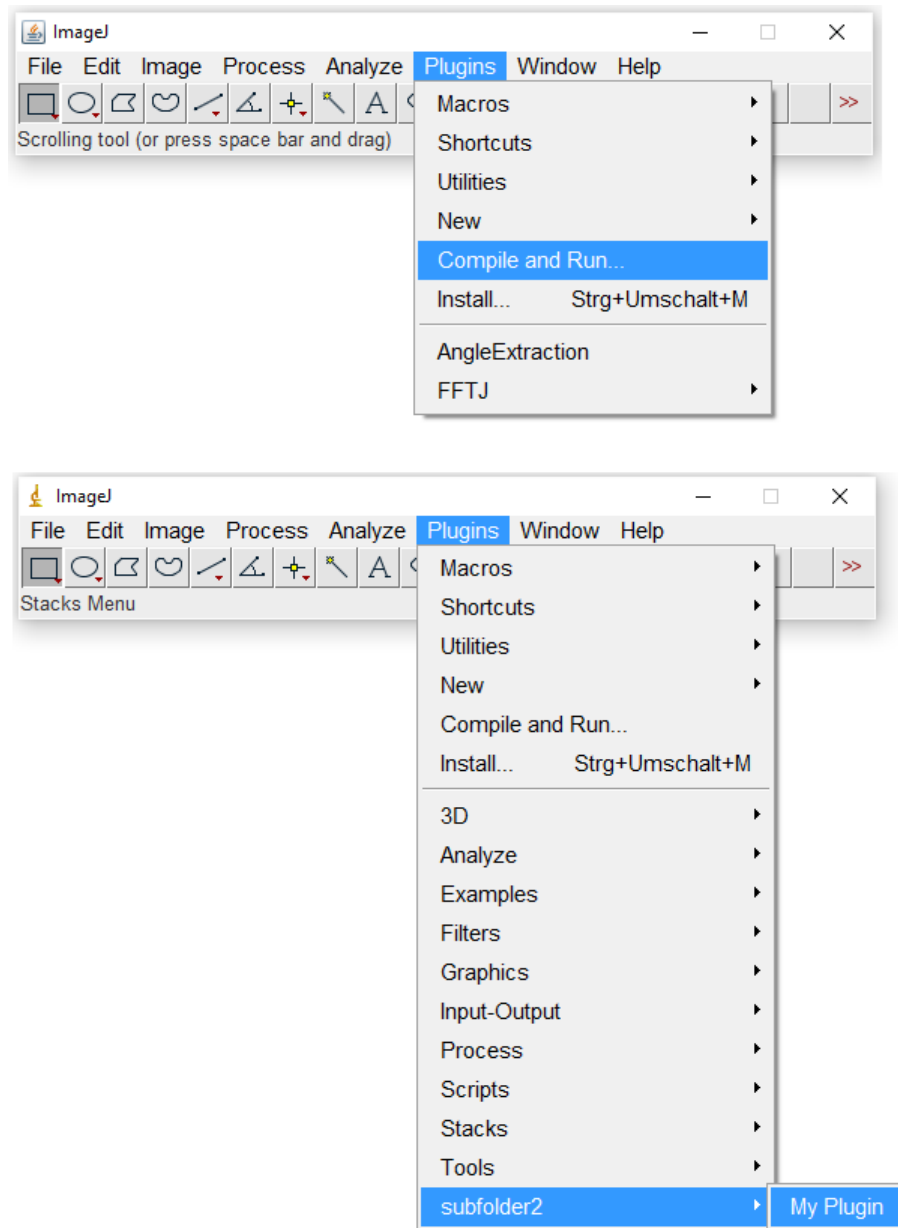


Figure 7.1: Menu entry to compile and install a new plug-in on the top and the plug-in appears after a restart in the menu (on the bottom).

After setting the parameters the processing itself begins and the progress can be observed in ImageJ's progress bar (marked with red rectangle in Fig. 7.3).

At the end two new windows appear:

- result table
contains the normalized intensity values in a range from $[0, 100]$

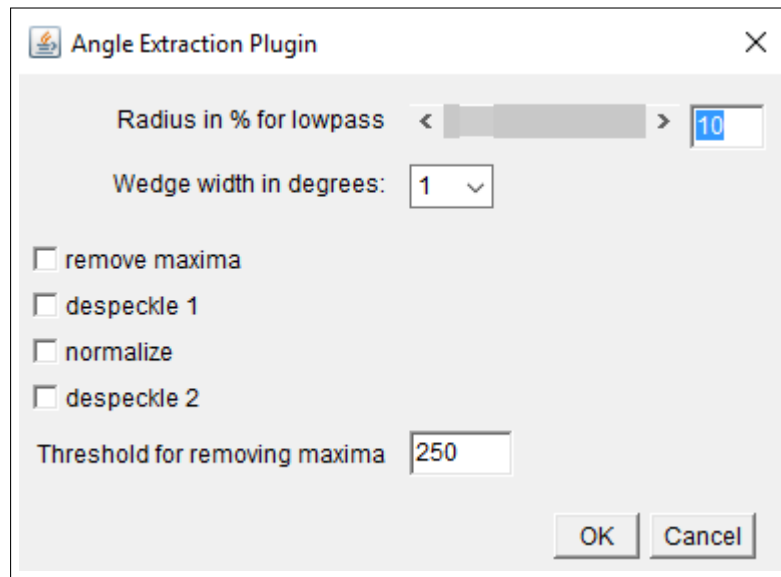


Figure 7.2: Input mask for the Angle Extraction plug-in with its standard values. The radius for the lowpass can either be changed by entering the value or using the slider. Wedge widths are restricted to fractions of 180, because otherwise a full rotation is not possible. The four preprocessing steps are executed from top to bottom, if selected. The threshold for removing the maxima has to be a value between 0 and 255 due to the gray value image type.

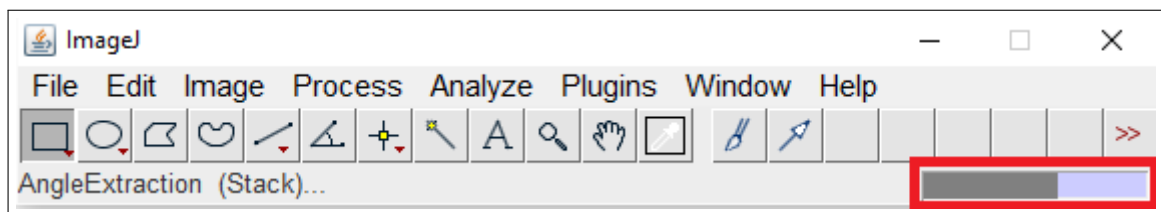


Figure 7.3: ImageJ's main window during execution of the Angle Extraction plug-in. The progress bar is marked with a red rectangle.

- result image
intensities from the result table are mapped to a LUT and displayed as an image

8 Verification

In order to assess the plug-in a Matlab [38] script to generate test-images with well-defined parameters was written (see Fig. 8.1).

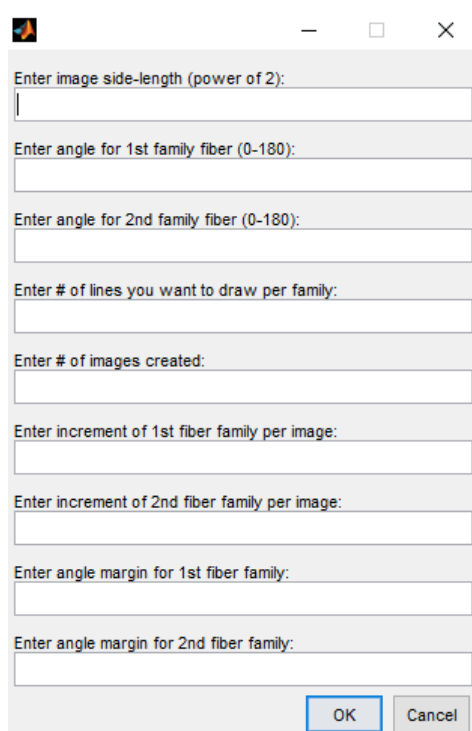


Figure 8.1: GUI for creating quadratic test-images with a power of 2 side-length. Parameters for fiber family 1 are used to create lines on the top half of the image and likewise for the lower half and fiber family 2. If more than one image is created the increment is added after each image. A margin is applied by adding a random value from the interval $[-margin, margin]$ to the original specified angle.

8.1 Generate Testimages

The script generates images with two sets of lines to mimic the two fiber families in the microscopic images that are to be analyzed. On the top half lines with parameters for fiber

family 1 is set up and on the lower lines with parameters for fiber family 2. The user needs to fill out following parameters to generate a set of test-images:

- image side-length
has to be a power of 2 for use of FFT
- angles for first and second fiber family
angles are positive in clockwise direction
have to lie between 0° and 180°
- number of lines per fiber family
- number of images created in total
- increments for first and second fiber family per image
Increments are applied after each image and positive values are clockwise rotations and negative values are counterclockwise rotations.
irrelevant if only 1 image is created
- angle margins for first and second fiber family
margin is multiplied with a random number $[-1, 1]$ and then added to the original angle

There is also the possibility to create more images and vary the angles with each image. This is to simulate changing fiber direction through depth. In addition to single image-files a stack-file containing all images is created, which is evaluated by the plug-in. In Fig. 8.2 a single test-image with angles of 45° and 135° is shown.

To identify the properties of each image they are stored in a folder containing the parameters in its name. Each parameter is represented by a string and are concatenated to form the full folder name in the following order:

- start angle fiber family 1 to end angle fiber family 1
- +- margin fiber family 1
- start angle fiber family 2 to end angle fiber family 2
- +- margin fiber family 2
- number of lines per fiber family
- increments for fiber family 1 and 2

This example: `0to180+-5_0to-180+-5_10lines_incr5_-5` contains images starting with both fiber families at 0° . Fiber families 1 and 2 have an angle margin of $\pm 5^\circ$. Lines

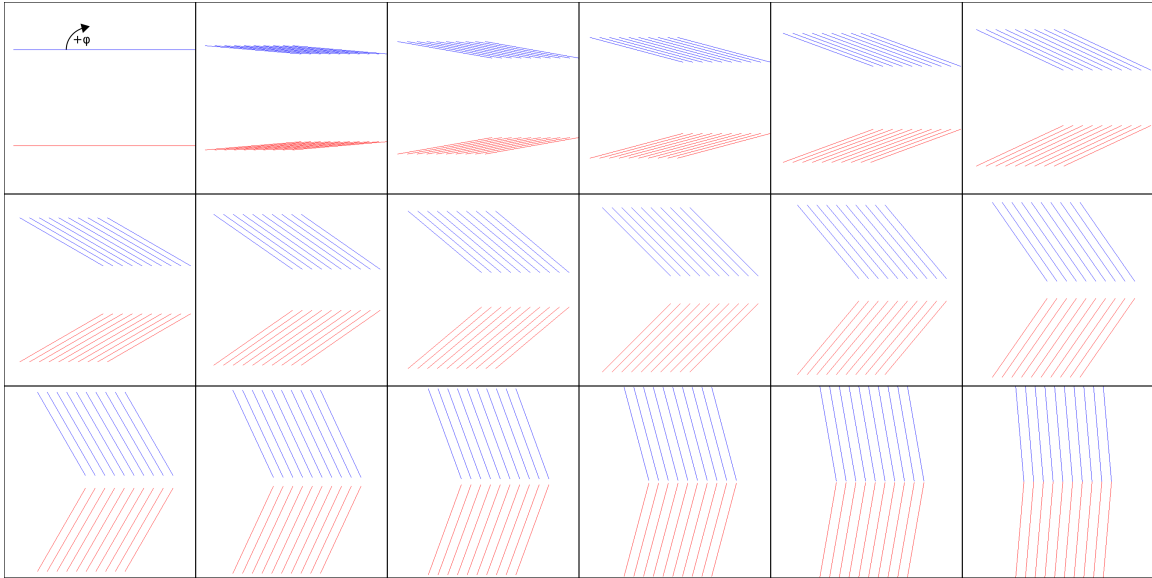


Figure 8.2: Part of a test image series created by the Matlab script with 10 lines per fiber family and $+5^\circ$ angle increase for fiber family 1 and -5° for fiber family 2 per image. The whole series continues until a total change of 180° is reached.

on the top half of the image are rotated by 5° clockwise and on the bottom half 5° counter-clockwise indicated by the negative sign. From this information the number of images total can be calculated by dividing the angle-span by the increment (here: $\frac{180-0}{5} = 36$ images).

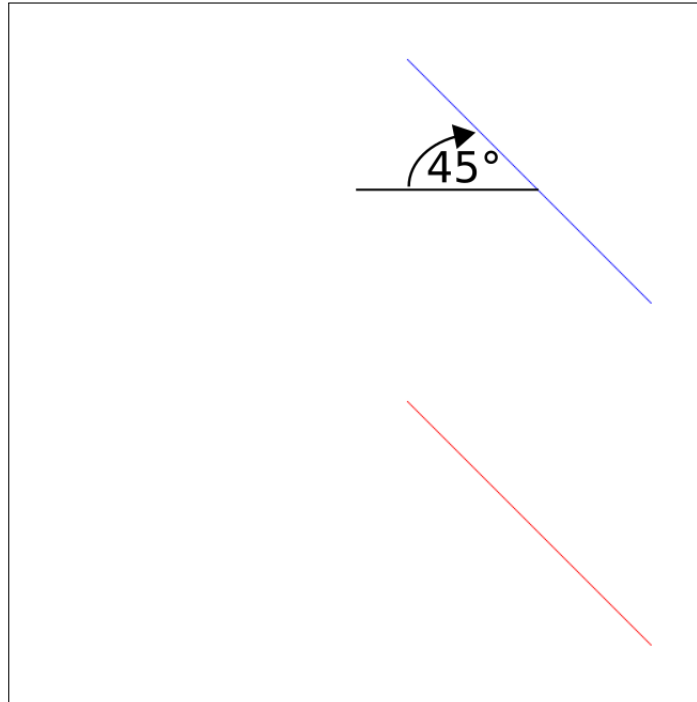
Within the folder single images have a file name with `angle_test_`, a counter and the base angles (without any margins) of both fiber families. `angle_test_3_10_-10.tif` is the third image of the series containing lines with angles of 10° and -10° .

8.2 Verification Results

Several sets of test-images were generated, analyzed with the plug-in and the results were compared with the input parameters. For all the verifications a 5° wedge-width and a low-pass radius of 10% was set. No preprocessing steps were carried out. In Fig. 8.3 the two fiber families contained a single line each at 45° . The resulting, normalized intensity values can be found in Tab. 8.1 and the visualized result in Fig. 8.4.

For further verification processes only the visualized output is shown. The resulting values can be reproduced by creating the desired test images with the Matlab script and analyzing those with the plug-in. After using one angle in one image another set was analyzed (see Fig.8.5 for detailed parameters).

The plug-in output for the image-series generated with Fig. 8.5 is shown in Fig. 8.6. The

Figure 8.3: Test image with two single lines at 45° .

angle	intensity	angle	intensity	angle	intensity
$0^\circ - 5^\circ$	0.553	$60^\circ - 65^\circ$	0.146	$120^\circ - 125^\circ$	0.016
$5^\circ - 10^\circ$	0.049	$65^\circ - 70^\circ$	0.085	$125^\circ - 130^\circ$	0.016
$10^\circ - 15^\circ$	0.032	$70^\circ - 75^\circ$	0.056	$130^\circ - 135^\circ$	0.016
$15^\circ - 20^\circ$	0.041	$75^\circ - 80^\circ$	0.042	$135^\circ - 140^\circ$	0.016
$20^\circ - 25^\circ$	0.056	$80^\circ - 85^\circ$	0.032	$140^\circ - 145^\circ$	0.016
$25^\circ - 30^\circ$	0.085	$85^\circ - 90^\circ$	0.029	$145^\circ - 150^\circ$	0.016
$30^\circ - 35^\circ$	0.149	$90^\circ - 95^\circ$	0.591	$150^\circ - 155^\circ$	0.016
$35^\circ - 40^\circ$	0.343	$95^\circ - 100^\circ$	0.023	$155^\circ - 160^\circ$	0.016
$40^\circ - 45^\circ$	2.267	$100^\circ - 105^\circ$	0.019	$160^\circ - 165^\circ$	0.017
$45^\circ - 50^\circ$	100.000	$105^\circ - 110^\circ$	0.018	$165^\circ - 170^\circ$	0.018
$50^\circ - 55^\circ$	2.260	$110^\circ - 115^\circ$	0.017	$170^\circ - 175^\circ$	0.019
$55^\circ - 60^\circ$	0.344	$115^\circ - 120^\circ$	0.017	$175^\circ - 180^\circ$	0.043

Table 8.1: Normalized result of the analysis of 8.3 by the plug-in. The maximal value is as expected between 45° and 50° , since the test image has only lines at 45° .

maxima (white boxes) match perfectly the input parameters and therefore represent a valid angle distribution.

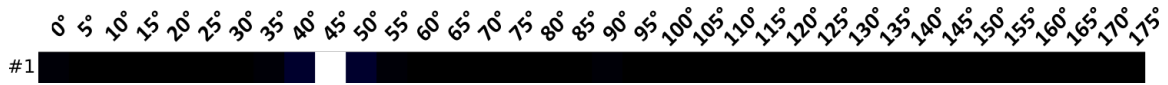


Figure 8.4: Plug-in output of the test image with two single lines at 45° . Wedge-width was 5° and the low-pass radius at 10%.

Figure 8.5: Parameters for the image set used to verify the plug-in. After each image fiber family 1 rotates by 5° clockwise and fiber family counter-clockwise. The expected output will be an x-shaped pattern, because both families span over $0^\circ - 180^\circ$. The angle margin is a randomly added/subtracted value to a lines angle before drawing in the range of the entered magnitude.

Finally a test-set of images with a margin of $\pm 5^\circ$ (see Fig. 8.7 for full parameters) was generated and analyzed. Because there are random numbers used in creating those images, one has to save them to be able to reproduce the results. Nevertheless, the main angle is expected to be found at the given values. In Fig. 8.8 the first 18 of 36 test-images are shown.

Interpreting the result (see Fig. 8.9) leads to following conclusions:

- maxima are found correctly

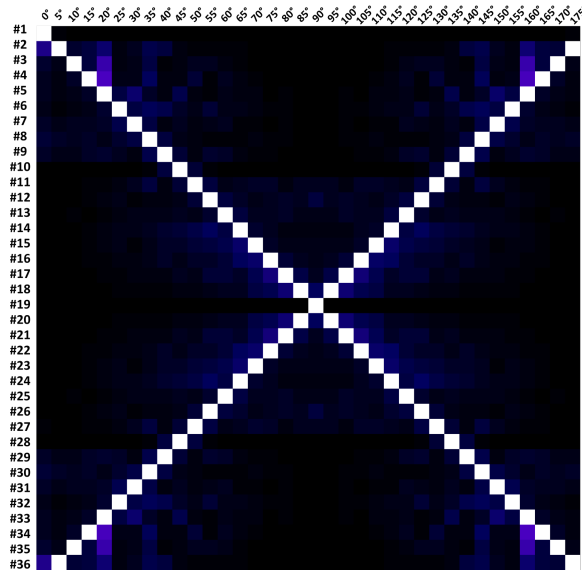
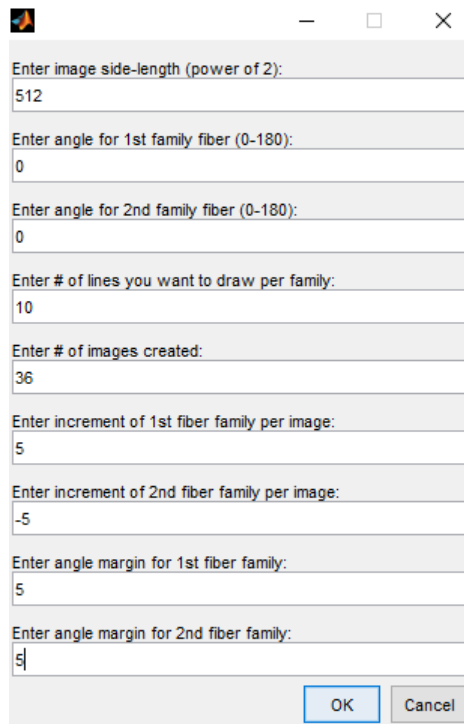


Figure 8.6: Calculated angle distribution for the image-set (total of 36 images) generated as defined in Fig. 8.5. The maxima of each image are exactly where they have to be in order to match the input parameters.

- neighboring angle-spans have higher intensities due to the margin

The green rectangle in Fig. 8.9 marks the individual angle distribution of Fig. 8.8 with the maxima at $45^\circ - 50^\circ$ and $-45^\circ - -50^\circ$ as expected.



A screenshot of a software dialog box with a title bar containing a small icon, a minus sign, a maximize button, and a close button. The dialog box contains several input fields with labels and values:

- Enter image side-length (power of 2): 512
- Enter angle for 1st family fiber (0-180): 0
- Enter angle for 2nd family fiber (0-180): 0
- Enter # of lines you want to draw per family: 10
- Enter # of images created: 36
- Enter increment of 1st fiber family per image: 5
- Enter increment of 2nd fiber family per image: -5
- Enter angle margin for 1st fiber family: 5
- Enter angle margin for 2nd fiber family: 5

At the bottom right of the dialog box are two buttons: "OK" and "Cancel".

Figure 8.7: Parameters for the final image set used to verify the plug-in. After each image fiber family 1 rotates by 5° clockwise and fiber family counter-clockwise. The expected output will be an x-shaped pattern, because both families span over $0^\circ - 180^\circ$. Here a random value between -5° and 5° was added to each single line before drawing.

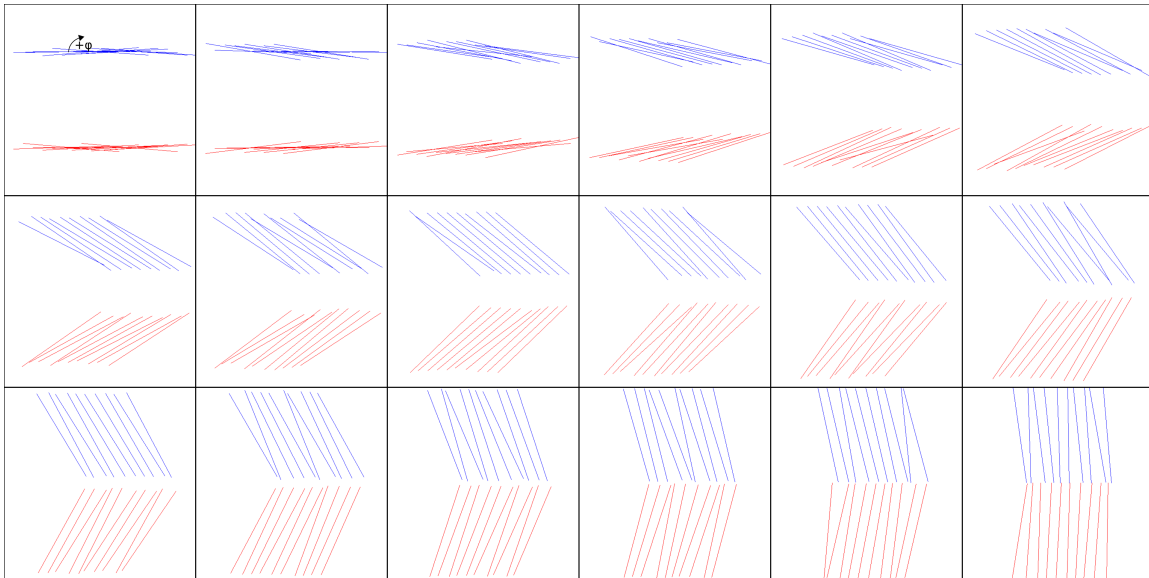


Figure 8.8: These are the first 18 images from the series generated with parameters seen in Fig. 8.7. The first fiber family has angles of $45^\circ \pm 5^\circ$ and the second fiber family $-45^\circ \pm 5^\circ$.

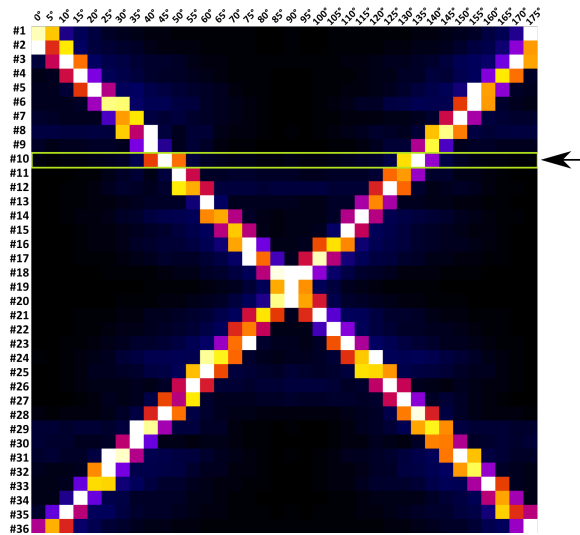


Figure 8.9: The result of the image-set generated with Fig. 8.7. Fig. 8.8's result is highlighted with a green rectangle. The maxima (white boxes) are still at the right place but one can see, that due to the margin of $\pm 5^\circ$ and a wedge-width of 5° neighboring fields have higher values than the corresponding fields in Fig. 8.6.

9 Conclusion

The Angle Extraction Plug-In introduced in this thesis presents a user-friendly method of a fiber distribution analysis. The results are displayed in a numerical result table and a visualization to be able to use in further work.

Due to the nature of the underlying images not just a single fiber angle but a distribution of angles and the dominant direction is sought. The Fourier transformation has the properties and methods to analyze structures and their corresponding orientation. For verification sets of test-images with clearly defined parameters are generated and analyzed. Those results show a high precision regarding the occurrence of angle changes and two different angle families.

Comparing the Matlab script and the new plug-in several improvements can be observed:

- The original script consists of two parts: The angle extraction script creates a file with the results and the plotting script visualizes the data as seen in Fig. 9.1. The image is normalized by the global maxima therefore only one wedge with a value of 100 is displayed. This makes it more difficult to locate the maximum angle at slices with low values because the ratio to the global maxima is similar and therefore of similar color. The resulting image of the plug-in is normalized slice by slice and therefore not a global maxima but a maxima per slice is displayed and the dominating angle spans can be picked out very easily.
- If the user wants to change the parameters, the user has to change them in the according lines of Matlab code whereas in the plug-in a user interface presents all possibilities of input and options to choose from.
- In the Matlab version the data is stored as comma-separated-values (CSV) in a file without extension and never directly displayed to the user. ImageJ has a result table which is shown additionally to the graphical representation thereof. This result table can be saved in an Excel-file and used by all compatible programs for further evaluation and computations if necessary.
- Last but not least the plug-in is much faster than the Matlab script and allows, due to this advantage, a more effective way of experimenting with the settings.

Considering all the mentioned advantages the Angle Extraction Plug-In is a significant improvement in the evaluation of microscopic images containing fibers.

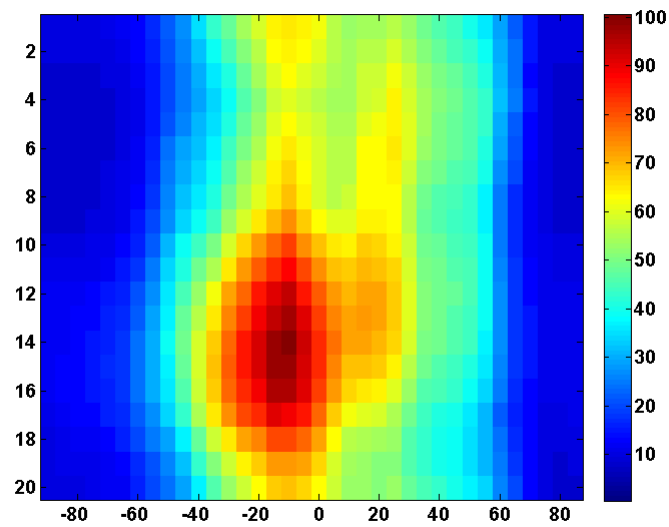


Figure 9.1: This is an example of the visual representation of the result of the angle extraction script generated by the plotting script. The scale to the left shows the globally normalized color-scale.

After finalizing the Angle Extraction Plug-In it will be made available to the public via the ImageJ platform using the web tool at the ImageJ Documentation Wiki ¹.

¹http://imagejdocu.tudor.lu/doku.php?id=create_new_content

Bibliography

- [1] C. A. Schneider, W. S. Rasband, and K. W. Eliceiri, “NIH image to ImageJ: 25 years of image analysis,” *Nature Methods*, 2012.
- [2] A. J. Schriebl, A. J. Reinisch, S. Sankaran, D. M. Pierce, and G. A. Holzapfel, “Quantitative assessment of collagen fibre orientations from two-dimensional images of soft biological tissues,” *Journal of The Royal Society Interface*, 2012.
- [3] E. Foundation, *Eclipse documentation*. Eclipse Foundation, 2015.
- [4] W. H. Organization, “Cardiovascular diseases (cvds),” 2016. Fact sheet.
- [5] G. A. Holzapfel and T. C. Gasser, “A viscoelastic model for fiber-reinforced composites at finite strains: Continuum basis, computational aspects and applications,” *Computer Methods in Applied Mechanics and Engineering*, 2001.
- [6] G. A. Holzapfel, M. Stadler, and C. A. J. Schulze-Bauer, “A layer-specific three-dimensional model for the simulation of balloon angioplasty using magnetic resonance imaging and mechanical testing,” *Annals of Biomedical Engineering*, 2002.
- [7] T. C. Gasser and G. A. Holzapfel, “A rate-independent elastoplastic constitutive model for biological fiber-reinforced composites at finite strains: continuum basis, algorithmic formulation and finite element implementation,” *Computational Mechanics*, 2002.
- [8] C. A. J. Schulze-Bauer, C. Morth, and G. A. Holzapfel, “Passive biaxial mechanical response of aged human iliac arteries,” *Journal of Biomechanical Engineering*, 2003.
- [9] G. A. Holzapfel, G. Sommer, and P. Regitnig, “Anisotropic mechanical properties of tissue components in human atherosclerotic plaques,” *Journal of Biomechanical Engineering*, 2004.
- [10] M. Walsh, E. Cunnane, J. Mulvihill, A. Akyildiz, F. Gijsen, and G. Holzapfel, “Uniaxial tensile testing approaches for characterisation of atherosclerotic plaques,” *Journal of Biomechanics*, 2014.
- [11] G. Sommer, D. C. Haspinger, M. Andrä, M. Sacherer, C. Viertler, P. Regitnig, and G. A. Holzapfel, “Quantification of shear deformations and corresponding stresses in the biaxially tested human myocardium,” *Annals of Biomedical Engineering*, 2015.

-
- [12] J. Tong, T. Cohnert, P. Regitnig, and G. Holzapfel, "Effects of age on the elastic properties of the intraluminal thrombus and the thrombus-covered wall in abdominal aortic aneurysms: Biaxial extension behaviour and material modelling," *European Journal of Vascular and Endovascular Surgery*, 2011.
- [13] G. Sommer, S. Sherifova, P. J. Oberwalder, O. E. Dapunt, P. A. Ursomanno, A. De-Anda, B. E. Griffith, and G. A. Holzapfel, "Mechanical strength of aneurysmatic and dissected human thoracic aortas at different shear loading modes," *Journal of Biomechanics*, 2016.
- [14] H. Weisbecker, M. J. Unterberger, and G. A. Holzapfel, "Constitutive modelling of arteries considering fibre recruitment and three-dimensional fibre distribution," *Journal of The Royal Society Interface*, 2015.
- [15] D. Balzani, S. Brinkhues, and G. A. Holzapfel, "Constitutive framework for the modeling of damage in collagenous soft tissues with application to arterial walls," *Computer Methods in Applied Mechanics and Engineering*, 2012.
- [16] J. Humphrey and G. Holzapfel, "Mechanics, mechanobiology, and modeling of human abdominal aorta and aneurysms," *Journal of Biomechanics*, 2012.
- [17] J. A. G. Rhodin, "Architecture of the vessel wall," 1980. *Handbook of Physiology*.
- [18] G. A. Holzapfel, T. C. Gasser, and R. W. Ogden, "A new constitutive framework for arterial wall mechanics and a comparative study of material models," *Journal of elasticity and the physical science of solids*, 2000.
- [19] G. A. Holzapfel, J. A. Niestrawska, R. W. Ogden, A. J. Reinisch, and A. J. Schriefl, "Modelling non-symmetric collagen fibre dispersion in arterial walls," *Journal of The Royal Society Interface*, 2015.
- [20] D. M. Pierce, F. Maier, H. Weisbecker, C. Viertler, P. Verbrugghe, N. Famaey, I. Fourneau, P. Herijgers, and G. A. Holzapfel, "Human thoracic and abdominal aortic aneurysmal tissues: Damage experiments, statistical analysis and constitutive modeling," *Journal of the Mechanical Behavior of Biomedical Materials*, 2015.
- [21] G. Sommer, T. C. Gasser, P. Regitnig, M. Auer, and G. A. Holzapfel, "Dissection properties of the human aortic media: An experimental study," *Journal of Biomechanical Engineering*, 2008.
- [22] D. Sage, "Orientationj." <http://bigwww.epfl.ch/demo/orientation/index.html>, 2012.
- [23] J. Schindelin, I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, J.-Y. Tinevez, D. J. White, V. Hartenstein, K. Eliceiri, P. Tomancak, and A. Cardona, "Fiji: an open-source platform for biological-image analysis," *Nature Methods*, 2012.

-
- [24] J.-Y. Tinevez, “Directionality.” <https://github.com/fiji/Directionality/>, 2010.
- [25] Z.-Q. Liu, “Scale space approach to directional analysis of images,” *Appl. Opt.*, vol. 30, no. 11, p. 1369, 1991.
- [26] I. Sobel, “An isotropic 3x3 image gradient operator,” 1968. Researchgate.
- [27] J. S. Perry, “Java language basics - object-oriented programming on the java platform,” 2015. Introduction to Java programming.
- [28] W. R. T. Ferreira, *ImageJ User Guide - IJ 1.46*, 2010-2012.
- [29] J. Gosling, “Java platform standard edition 8 documentation,” 1996.
- [30] P. Pirotte, “The imagej eclipse how-to,” 2010. A guide on how to include ImageJ into Eclipse and develop plugins using this IDE.
- [31] E. W. Weisstein, “Fourier series.” MathWorld—A Wolfram Web Resource.
- [32] E. W. Weisstein, “Fourier transform.” MathWorld—A Wolfram Web Resource.
- [33] E. W. Weisstein, “Discrete fourier transform.” MathWorld—A Wolfram Web Resource.
- [34] D. Lyon, “The discrete fourier transform, part 2: Radix 2 fft,” *Journal of Object Technology*, 2009.
- [35] P. Bourke, “2 dimensional fft,” 1993. Online Document.
- [36] L. Moisan, “Periodic plus smooth image decomposition,” *Journal of Mathematical Imaging and Vision*, vol. 39, pp. 161–179, oct 2010.
- [37] N. Linnenbruegger, “Fftj and deconvolutionj.” ImageJ - Plug-In, 2001.
- [38] MATLAB, *MATLAB version 8.5.0.197613 (R2015a)*. The Mathworks, Inc., Natick, Massachusetts, 2015.