SerialFIB Documentation

Sven Klumpe, Sara Katrin Goetz, Herman Fung

August 25, 2021

Contents

1 Introduction

The (cryo-)FIB/SEM microscope is a versatile tool for ultrastructural analysis, both as imaging tool in FIB/SEM tomography as well as in sample preparation for subsequent *in situ* cryo-electron tomography. The process of imaging and/or sample preparation is time consuming. SerialFIB is providing a customizable automation toolkit in order to automate repetitive and time-intensive tasks for developing and streamlining workflows. We hope it will help you in studying the biology you are interested in!

1.1 General architecture

The software is divided into a graphical user interface, image processing scripts and a driver that provides the functionalities of the FIB/SEM microscope via commercial Advanced Programming Interfaces (API). So far, only the Thermo Fisher Scientific driver, based on AutoScript4, has been developed.

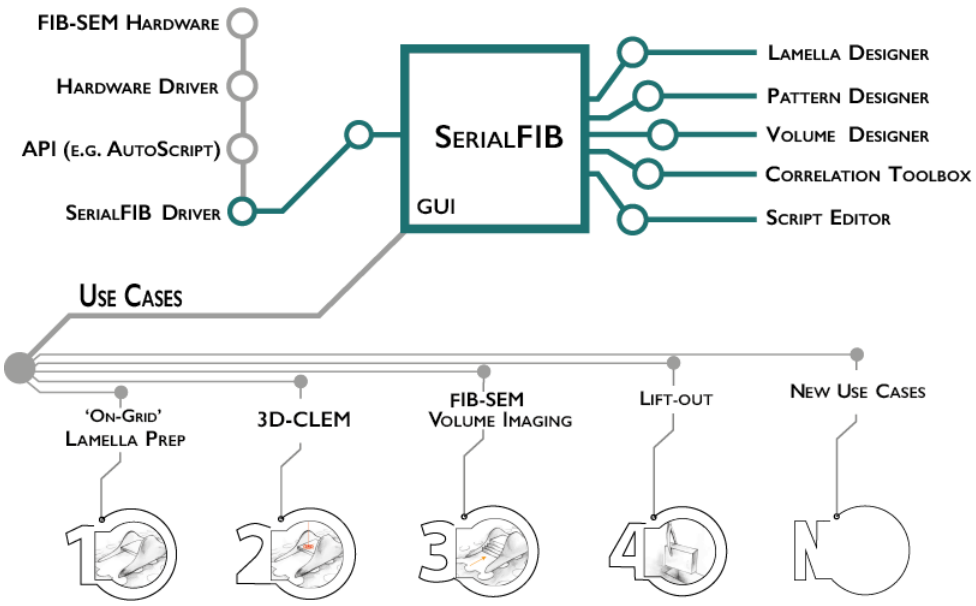


Figure 1: General software architecture

2 Installation

Once AutoScript4 (version 4.0 and higher) has been installed on your system , the only package that is missing should be PyQt5. It can be install simply via pip by typing

```
pip install PyQt5
```

into the command line. Then, on the PC used to run SerialFIB (usually the Support PC on TFS DualBeam systems), simply clone the repository

```
git clone http://github.com/sklumpe/SerialFIB
```

and start the program by typing

```
python SerialFIB.py
```

Alternatively, you can create a shortcut on the Desktop. Simply create one that points first to the python executable of your AutoScript4 installation and then the path to the repository, pointing at the SerialFIB.py file. On our systems, this would e.g. look like this:

```
"C:\Program Files\Python35\envs\AutoScript\python.exe"
D:\SharedData\SerialFIB\SerialFIB.py
```

The SerialFIB GUI can also be run locally, either with a virtual AutoScript4 machine to enable response from the virtual microscope, or simply by starting

SerialFIB (python SerialFIB.py). Currently, the AutoScript4 wheels are still needed for offline operation, this will change in the future once we have established drivers for other systems. You should have access to the wheels through your license. In offline mode, the software will load dummy images found in the ./DummyImages/ directory of the repository. For local installation without AutoScript4, the dependencies are:

```
PyQt5
numpy
cv2
pickle
scikit-image
```

They can be installed by typing

```
pip install PyQt5 numpy cv2 pickle skimage
```

into the command-line of your OS.

3 Quick guide

For general operations, please see the SerialFIB tutorial in the repository. Scripting functions are documented below. Main functions that are needed for scripting are the functions starting with "run_" as they directly call a routine given pattern sequence files, SAVparams or lamellae protocols. The only thing that needs to be prepared before each run are the output directories using the "write_patterns" function. Plans are to include a simple "prepare_run" command to make that point easier. If you have any other suggestions, please contact us! We are grateful for feedback!

4 Scripting example

Here is a little walkthrough for a sample scripting case running one of the protocols in the ScriptEditor rather than from the GUI.

```
### Testing cryo-FIB protocol milling ###

### User Input ###
output_dir=r'D:/SharedData/Sven/20210224_Testing/Test10'
img_index=0
stagepos_index=0
pattern_index=0
protocol=r'D:/SharedData/Sven/Developing/SerialFIB/testprotocol.pro'

##############

### Definition of variables ###

fibsem.output_dir=output_dir+'/'
label=stagepositions[stagepos_index]['label']
alignment_image=images[img_index]
pattern_dir=output_dir+'/'+str(label)+'/'
stagepos=stagepositions[stagepos_index]

fibsem.write_patterns(label,patterns[pattern_index],alignment_image,output_dir)

### Creating patternfile ###
fibsem.run_milling_protocol(label,alignment_image,stagepos,pattern_dir,protocol)
```

Let's walk through the code. We have some user inputs that are usually given by the GUI such as the output directory as well as the protocolfile that we want to use. Img_index, stagepos_index and pattern_index provide us with a position we want to work with. If you want to do it for all positions, simply loop through all stagepositions and images, which are loaded as a list.
This concludes the first part of the script.

```
### Testing cryo-FIB protocol milling ###

### User Input ###
output_dir=r'D:/SharedData/Sven/20210224_Testing/Test10'
img_index=0
stagepos_index=0
pattern_index=0
protocol=r'D:/SharedData/Sven/Developing/SerialFIB/testprotocol.pro'

##############
```

Next, those indices are used to call from the lists and get the elements needed for the run, e.g. alignment_image, label etc.

```
### Definition of variables ###

fibsem.output_dir=output_dir+'/'
label=stagepositions[stagepos_index]['label']
alignment_image=images[img_index]
pattern_dir=output_dir+'/'+str(label)+'/'
stagepos=stagepositions[stagepos_index]
```

Finally, the patterns (in XT server format) needed for the run are written using the fibsem.write_patterns() command.

```
fibsem.write_patterns(label,patterns[pattern_index],alignment_image,output_dir)
```

and lastly, the protocol is started by calling the fibsem.run_milling_protocol function.

```
### Creating patternfile ###
fibsem.run_milling_protocol(label,alignment_image,stagepos,pattern_dir,protocol)
```

While this a very rudimentary introduction, the functions available from the FIBSEM driver are listed in alphabetical order at the end of this document. Feedback on how to improve on the procedure and make it more user-friendly is greatly appreciated. If you have any questions or want to contribute, this is ! HIGHLY ! appreciated. Feel free to contact klumpe (at) biochem.mpg.de.

5 Scripting example: preliminary Waffle method Automation

An additional example for customized scripting aimed at the development of fully automated Waffle method. The script can be found on the GitHub repository in the directory Scripting examples. The additional pattern sequence files and protocols needed are in the subdirectory FilesForWaffleDev. First the script below.

```
### Testing cryo-FIB custom milling ###

### User Input ###
output_dir=r'D:/SharedData/Sven/20210805_WaffleDev/Test4/'

patternfile_trench=r'D:/SharedData/Sven/20210805_WaffleDev/patternfiles/TopCut2.pf'

patternfile_notch=r'D:/SharedData/Sven/20210805_WaffleDev/patternfiles/notch.pf'

lamella_protocol=r'D:/SharedData/Sven/20210805_WaffleDev/patternfiles/lamellamill.pro'

img_index_trench=2
stagepos_index_trench=0
pattern_index_trench=2

img_index_notch=0
stagepos_index_notch=1
pattern_index_notch=0

img_index_lamella=1
stagepos_index_lamella=1
pattern_index_lamella=1

##############


#### Initial Trench from Above ####

### Definition of variables ###

fibsem.output_dir=output_dir+'/'
label=stagepositions[stagepos_index_trench]['label']
alignment_image=images[img_index_trench]
pattern_dir=output_dir+'/'+str(label)+'/'
stagepos=stagepositions[stagepos_index_trench]

fibsem.write_patterns(label,patterns[pattern_index_trench],alignment_image,output_dir)


fibsem.run_milling_custom(label,alignment_image,stagepos,pattern_dir,patternfile_trench)


#### Notch Milling ####

label=stagepositions[stagepos_index_notch]['label']
alignment_image=images[img_index_notch]
```

```
pattern_dir=output_dir+'/'+str(label)+'/'
stagepos=stagepositions[stagepos_index_notch]

fibsem.write_patterns(label,patterns[pattern_index_notch],alignment_image,output_dir)

fibsem.run_milling_custom(label,alignment_image,stagepos,pattern_dir,patternfile_notch)



#### Lamella Milling ####



##############

### Definition of variables ###

fibsem.output_dir=output_dir+'/'
label=stagepositions[stagepos_index_lamella]['label']
alignment_image=images[img_index_lamella]
pattern_dir=output_dir+'/'+str(label)+'/'
stagepos=stagepositions[stagepos_index_lamella]

#print(patterns)
fibsem.write_patterns(label,patterns[pattern_index_lamella],alignment_image,output_dir)


### Creating patternfile ###



fibsem.run_milling_protocol(label,alignment_image,stagepos,pattern_dir,lamella_protocol)
```

Let's look at it again section by section. First, let's have a look at the user input. It starts off by defining the output directory as well as the pattern sequence files and protocols needed for the task. Furthermore, indices for certain images are defined. The index 2 corresponds to the image of the actual site prior to any milling operations (img_index_trench and pattern_index_trench). Thus, the stageposition index 0 (stagepos_index_trench) corresponds to the stage position at 0 degree relative rotation to the loading rotation and 7 degree tilt, to allow the FIB to come at a perpendicular angle to the sample's surface.

Next, the 0th image is an image loaded via the correlation module (simply press cancel when it asks for the 3DCT output, only the FIB image is going

to be loaded) that shows an arbitrary Waffle site after the trench (or TopCut) in order to have a reference on which to align the position after the trench has been performed. Same goes for the 1st image, corresponding to the site after trench and notch milling. Stage position 1 is the Waffle position at the normal milling angle for lamella preparation, meaning 180 degree relative rotation to the loading rotation and e.g. 20 degree tilt.

```
### User Input ###
output_dir=r'D:/SharedData/Sven/20210805_WaffleDev/Test4/'

patternfile_trench=r'D:/SharedData/Sven/20210805_WaffleDev/patternfiles/TopCut2.pf'

patternfile_notch=r'D:/SharedData/Sven/20210805_WaffleDev/patternfiles/notch.pf'

lamella_protocol=r'D:/SharedData/Sven/20210805_WaffleDev/patternfiles/lamellamill.pro'

img_index_trench=2
stagepos_index_trench=0
pattern_index_trench=2

img_index_notch=0
stagepos_index_notch=1
pattern_index_notch=0

img_index_lamella=1
stagepos_index_lamella=1
pattern_index_lamella=1

##############
```

The initial milling steps corresponds to the trench milling perpendicular to the sample's surface. The alignment image is taken before, shown in figure 2 As shown in the previous scripting example above, we simply define our labels, alignment_image, pattern_dir and stagepos based on the indices for trench milling. fibsem.write then takes care of writing the folder structure as well as relevant patterns into the output directory and fibsem.run_milling_custom then runs the actual milling job based on the pattern sequence file 'patternfile_trench'.

```
#### Initial Trench from Above ####

### Definition of variables ###

fibsem.output_dir=output_dir+'/'
```

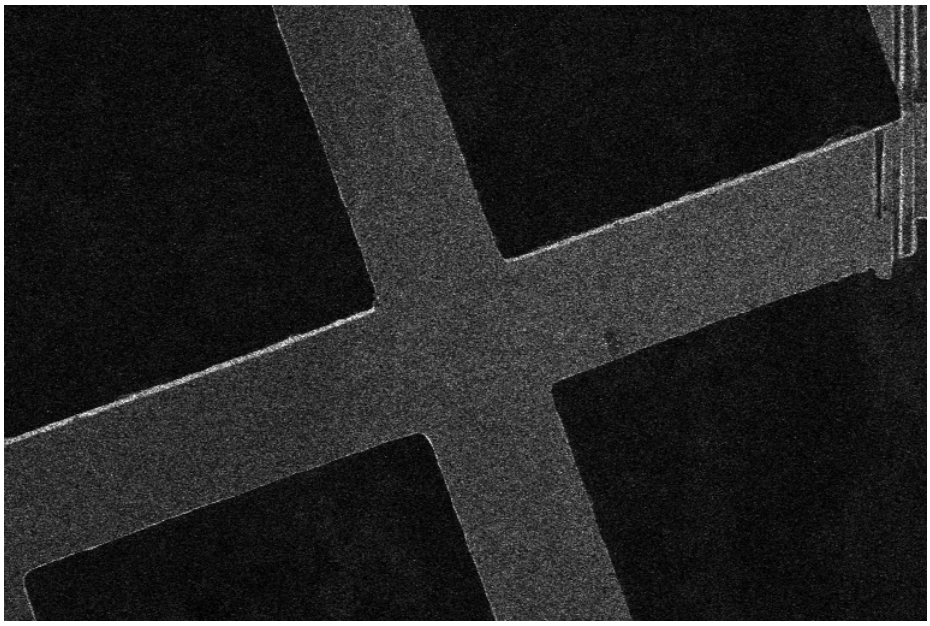


Figure 2: Reference Image for inital trench (or TopCut) milling. No scale bars since this is the exact image SerialFIB took.

```
label=stagepositions[stagepos_index_trench]['label']
alignment_image=images[img_index_trench]
pattern_dir=output_dir+'/'+str(label)+'/'
stagepos=stagepositions[stagepos_index_trench]

fibsem.write_patterns(label,patterns[pattern_index_trench],alignment_image,output_dir)


fibsem.run_milling_custom(label,alignment_image,stagepos,pattern_dir,patternfile_trench)
```

Next we are going to do the notch milling based on the reference image of an arbitrary trench milled site shown in figure 3. For testing, this has been performed on a simple empty copper grid. The reference image is shown after the code. Note that we again change the references based on the notch indices now and fibsem.write_patterns and fibsem.run_milling_custom are used as described above.

```
#### Notch Milling ####

label=stagepositions[stagepos_index_notch]['label']
```

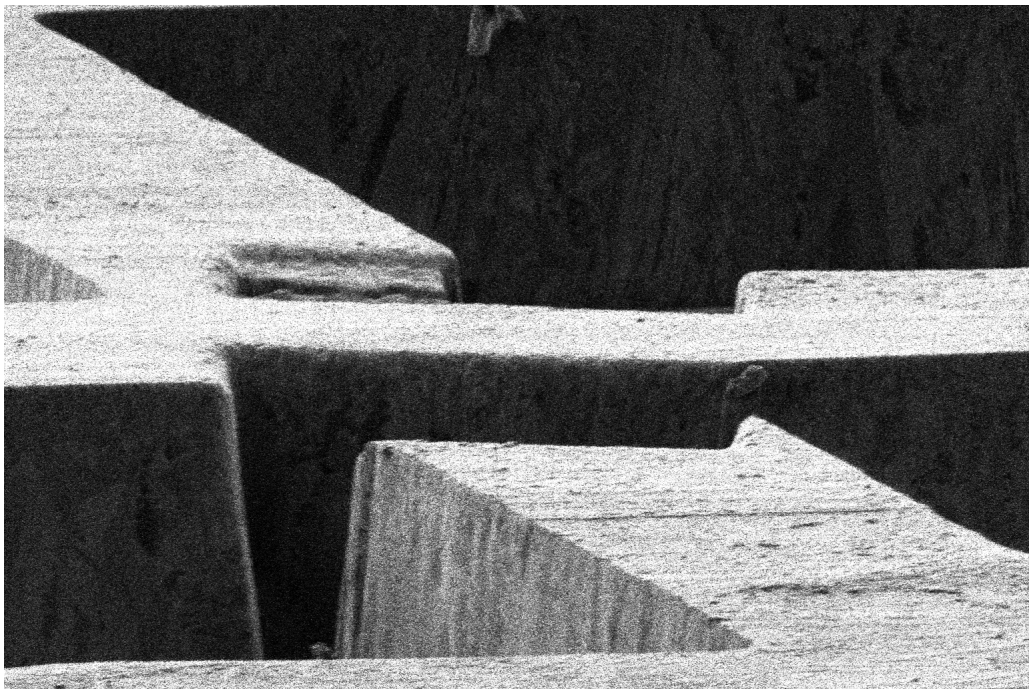


Figure 3: Reference Image for notch milling of an abitrary previously milled position loaded into SerialFIB via the load correlation function (canceling when asked for 3DCT output). No scale bars since this is the exact image SerialFIB took.

```
alignment_image=images[img_index_notch]
pattern_dir=output_dir+'/'+str(label)+'/'
stagepos=stagepositions[stagepos_index_notch]

fibsem.write_patterns(label,patterns[pattern_index_notch],alignment_image,output_dir)

fibsem.run_milling_custom(label,alignment_image,stagepos,pattern_dir,patternfile_notch)
```

Finally, we have to produce our actual lamella. For this, we are going to take another reference image loaded into the GUI via the correlation module (again pressing cancel for the 3DCT output file). The image is shown again after the code in figure 4. This image corresponds to an image after notch milling. Rather than calling fibsem.run_milling_custom, we call fibsem.run_milling_protocol since we now want to create a lamella and this is easiest done using the lamella protocol files.

```
#### Lamella Milling ####



##############

### Definition of variables ###

fibsem.output_dir=output_dir+'/'
label=stagepositions[stagepos_index_lamella]['label']
alignment_image=images[img_index_lamella]
pattern_dir=output_dir+'/'+str(label)+'/'
stagepos=stagepositions[stagepos_index_lamella]

#print(patterns)
fibsem.write_patterns(label,patterns[pattern_index_lamella],alignment_image,output_dir)


### Creating patternfile ###



fibsem.run_milling_protocol(label,alignment_image,stagepos,pattern_dir,lamella_protocol)
```

Based on our defined protocol, the lamella will be prepared. The result on the copper grid test sample is shown in figure 5.

We do realize there are missing steps (mainly GIS Pt Deposition prior to notch and lamella production). This example is given to illustrate the ScriptEditor further rather than to provide a full solution to automated lamella milling using the Waffle method. We are however currently working to implement that feature into SerialFIB, so stay tuned!

6 Analysis functions: processSEM.py

The analysis script for SEM images produced by SerialFIB's volume imaging module can be found in the directory ./analysis/ . Generaly usage is

```
usage: processSEM.py [-h] -indir --input_directory [--input_directory ...]
-outdir --output_directory [--output_directory ...]
[-l --level [--level ...]] [-s --sigma [--sigma ...]]
[-wname --wavelet_name [--wavelet_name ...]]
[-sb --sigma_blur [--sigma_blur ...]]
[-offset --offset_blur [--offset_blur ...]]
[-iter --iterations_erosion [--iterations_erosion ...]]
```

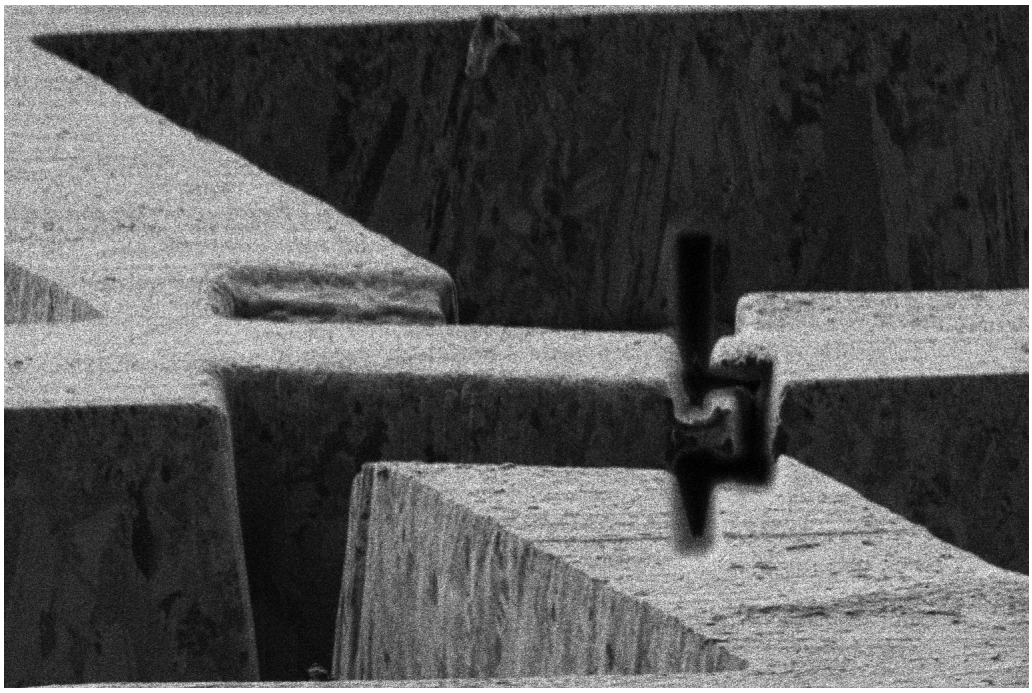


Figure 4: Reference Image for lamella milling of an abitrarily position after notch milling previously prepared and loaded into SerialFIB. No scale bars since this is the exact image SerialFIB took.

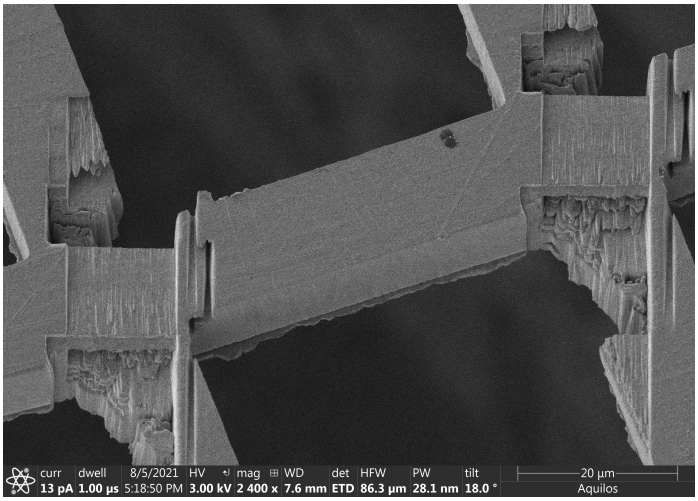
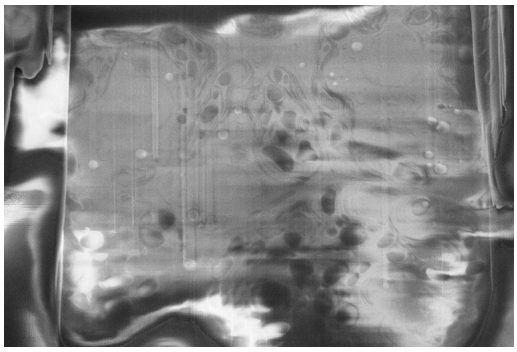


Figure 5: SEM image of the position after the preliminary Waffle script.

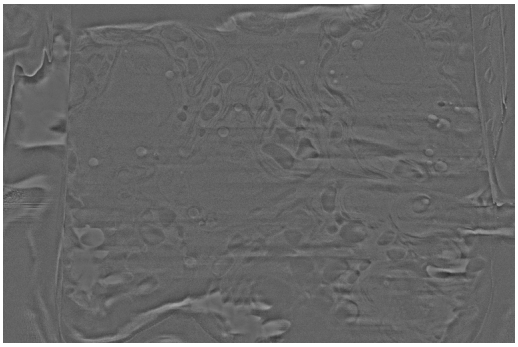
```
Analysis for raw images from SAV data.

optional arguments:
-h, --help            show this help message and exit
-indir --input_directory [--input_directory ...]
Input Directory with Raw Images
-outdir --output_directory [--output_directory ...]
Output Directory
-l --level [--level ...]
Level of wavelet decomposition. Default is 8.
-s --sigma [--sigma ...]
Sigma of gaussian for stripe dampening in wavelet
decomposition. Default is 6.
-wname --wavelet_name [--wavelet_name ...]
Wavelet for decomposition. Default is coif3.
-sb --sigma_blur [--sigma_blur ...]
Sigma for blurred image to compensate charging.
Default 35.
-offset --offset_blur [--offset_blur ...]
Offset for blurred image mask. Default 100.
-iter --iterations_erosion [--iterations_erosion ...]
Number of iterations of image erosion for charge
compensation. Default 3.
```

where input_directory holds the images to be processed.

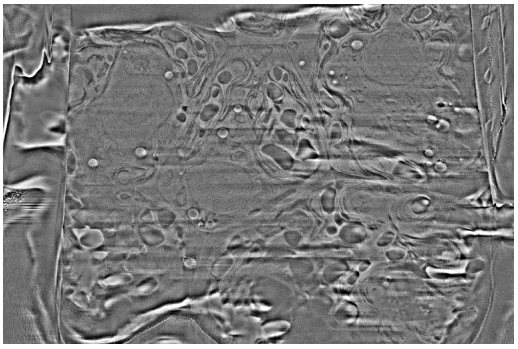
An example before and after processing is shown in Figure 6 Finally, the images is brightness contrast adjusted and contrast enhanced in FIJI using CLAHE filter.



(a) Raw Image



(b) Image after post processing in processSEM.py



(c) Image after post processing in processSEM.py and contrast enhancement using CLAHE

Figure 6: Processing of SEM images using processSEM.py and I

7 FIB/SEM driver functions

Functions in the FIB/SEM driver callable through the ScriptEditor as fibsem.commandname in alphabetical order are documented here:

- align(image,beam,current=1.0e-11)
  - Input: image as AdornedImage e.g. from ImageBuffer, beam="ION" or "ELECTRON"
  - Output: -
  - Action: Aligning current view to provided image
  - Comment: Down to 2 um, stage move is used, then beam shift
- align_current(new_current, beam="ION")
  - Input: new current as float, beam="ION" (default) or "ELECTRON"
  - Output: -
  - Action: Aligning current view after changing beam current
  - Comment: An Image is taken, aperture is changed and alignment to taken image
- auto_cb()
  - Input: -
  - Output: -
  - Action: Runs auto contrast brightness
  - Comment: -
- auto_focus(beam="ELECTRON)
  - Input: beam "ELECTRON" (default) or "ION"
  - Output: -
  - Action: Auto focusses on current region
  - Comment: -
- connect()
  - Input: None
  - Output: None
  - Action: Connects to microscope server
  - Comment: -
- create_SAV_patterns(self, directory, pattern_lamella, pattern_above, pattern_below)

  - Input: path to the directory of the lamella position, name of the three definition patterns from the SerialFIB GUI
  - Output: list of patterns and list of currents to be used for milling
  - Action: Creates patterns for volume imaging jobs
  - Comment: -

- create_custom_patterns(self, directory, pattern_lamella, pattern_above, pattern_below,custom_filename)
  - Input: Directory path as string, filename of lamella pattern, and extreme point patterns as string pattern sequence file path as string, custom_filename path as string
  - Output: 33 lists. One for the patterns, one for the beam currents of the steps in the protocol, and one for the time variable of the steps in the protocol
  - Action: -
  - Comment: -

- create_custom_protocol(self, directory, pattern_lamella, pattern_above, pattern_below, protocol_filename,mode="fine")
  - Input: Directory path as string, filename of lamella pattern, and extreme point patterns as string protocol file path as string, optionally mode ("rough" or "fine"). "Rough" takes extreme points into account, "fine" does not.
  - Output: 3 lists. One for the patterns, one for the beam currents of the steps in the protocol, and one for the time variable of the steps in the protocol
  - Action: -
  - Comment: -

- create_pattern(x,y,h,w,d=10e-06)
  - Input: x,y,h,w,d as float, depth gets default value of 10 um
  - Output: pattern class
  - Action: microscope draws pattern in ion beam view
  - Comment: default depth 10e-06

- create_trench_patterns(directory,pattern_lamella,pattern_above,pattern_below)
  - Input: Directory containing the user input from the SerialFIB GUI as xT patterns
  - Output: AutoScript4 "pattern" objects for trench milling
  - Action: -

  - Comment: -

- custom_file_parser
  - Input: filename as str
  - Output: pattern_dict, list of currents per step
  - Action: -
  - Comment: Parser for patterns created using the PatternEditor

- custom_file_parser(self, custom_filename)
  - Input: ath to pattern sequence file
  - Output: Dictionnary of AutoScript4 patterns corresponding to step names, list of step names, list of ion beam currents corresponding to the step names
  - Action: -
  - Comment: -

- define_SAV_params_file(file)
  - Input: Path to SAVparams file as str
  - Output: None
  - Action: sets variable fibsem.SAVparamsfile to provided path
  - Comment: -

- define_output_directory(directory)
  - Input: directory as str
  - Output: None
  - Action: defines output directory for images and patterns
  - Comment: -

- disconnect()
  - Input: None
  - Output: None
  - Action: Disconnects from microscope server
  - Comment: -

- getStagePosition()
  - Input: None
  - Output: stageposition as dict
  - Action: -

  - Comment: -

- get_current()
  - Input: None
  - Output: Current as float
  - Action: Grabs current value for the ion beam current from the microscope server
  - Comment: -

- is_idle()
  - Input: None
  - Output: Boolean
  - Action: Checks whether microscope is milling or not. Returns True for idle, False for milling
  - Comment: -

- makePatterns_SAV(self,y_start, y_end, slice_thickness, width, pattern_type, scan_direction, milling_current, output_dir)
  - Input: Start and end position from SerialFIB GUI, parameters from SAV params file (slice thickness int, width float, pattern_type string, scan_direction string, milling_current float)
  - Output: -
  - Action: Creates pattern sequence file for volume imaging jobs, writes it out in lamella output directory as "SAV_pattern_file.pf"
  - Comment: -

- moveStageAbsolute(stageposition)
  - Input: stageposition as dict
  - Output: -
  - Action: moves stage to the stageposition values provided
  - Comment:

- moveStageRelative(stageposition)
  - Input: stageposition as dict
  - Output: -
  - Action: stage is moved by the values provided
  - Comment: e.g. if z=-0.1, stage is moved 0.1 mm down

- pattern_directory_parser(directory)

  - Input: directory as str
  - Output: list of patterns
  - Action: draws patterns from directory in active view
  - Comment: -

- pattern_parser(directory,filename)
  - Input: directory, filename of xT server .ptf as str
  - Output: pattern class
  - Action: microscope draws pattern from filename in active view
  - Comment: -

- run_SAV
  - Input: lamella_name as str, iamge from ImageBuffer, stagepos as dict, pattern_directory as str, custom_filename as str
  - Output: -
  - Action: runs SAV job
  - Comment: images are taken with the take_image_EB_SAV function

- run_milling
  - Input: pattern_directory, top_pattern, bottom_pattern as str, milling time as int
  - Output: -
  - Action: Run patterning with given patterns and milling time
  - Comment: time-based milling

- run_milling_custom(self,lamella_name,alignment_image,stagepos,pattern_ref_directory,custom_filename)
  - Input: Lamella name as string, alignment image as numpy array, stageposition as dictionary, path to the pattern directory as string, path to pattern sequence file (custom_filename) as string
  - Output: Log for printing
  - Action: Runs provided pattern sequence file at the given position
  - Comment: -

- run_milling_protocol(self, lamella_name, alignment_image, stagepos, pattern_ref_directory, protocol_filename,mode="fine"
  - Input: lamella name from positions, alignment image as numpy array, stageposition as dictionary, site definition directory from the Serial-FIB GUI, path to protocol file as string, optional mode , "Rough" takes extreme positions for material ablation into account, "fine" does not

  - Output: Log for printing
  - Action: Runs milling defined by given protocol file at the provided lamella position
  - Comment: -

- run_trench_milling(lamella_name,alignment_image,stagepos,pattern_ref_directory)
  - Input: Lamella Name from positions list, Alignment image as Numpy array, stageposition as dictionary, Directory of the patterns defined through the SerialFIB GUI
  - Output: log for printing
  - Action: Runs the trench milling for the provided position
  - Comment: -

- save_pattern(directory,filename,Pattern)
  - Input: directory,filename as str, Pattern as pattern class
  - Output: -
  - Action: saves pattern to xT server .ptf file
  - Comment: -

- stop()
  - Input: None
  - Output: None
  - Action: sets variable fibsem.continuerun to False to end running protocol thread
  - Comment: -

- stop_patterning()
  - Input: None
  - Output: None
  - Action: If patterning is running, it is stopped
  - Comment: -

- take_image_EB()
  - Input: None
  - Output: EB image as AdornedImage
  - Action: -
  - Comment: -

- take_image_EB_SAV()

  - Input: None
  - Output: EB image as AdornedImage
  - Action:
  - Comment: Higher resolution and dwell times, line integration all defined through the provided SAV params file

- take_image_IB()
  - Input: None
  - Output: IB image as AdornedImage
  - Action: -
  - Comment: -

- write_patterns(self,label,patterns,alignment_image,output_dir)
  - Input: Input: Label of the lamella position, List of patterns as AutoScript4 objects, alignment image as numpy array, output directory path as string
  - Output: -
  - Action: Writes the patterns as xT .ptf files
  - Comment: -