Documentation

written with Evernote 10.68.3

Overview

This Matlab code performs image processing and alignment tasks on *serial section images* with the goal to prepare 3D data for segmentation.

Why Scripting, Why Matlab

Processing routes vary considerably from run to run. For example

- data are recorded on various FIB platforms which use different recording strategies and distinguish in their raw data structure,
- a large variety of detectors are used, sometimes multiples simultaneously in a single run,
- certain detector geometries lead to image gradients,
- others generate noisy data that necessitate the application of denoising algorithms to facilitate the segmentation process,
- image series suffer to a different degree under imperfections, such as image-to-image scan distortions, depending on the instrument and lab conditions,
- materials with strong local inhomogeneity in their milling behavior show curtaining artifacts.

A script-based approach allows to tailor the *workflow of processing steps* to the specific needs of a given image series and flexibly react to unforeseen events. The *Matlab* software is well curated and its various toolboxes allow to tackle major computational challenges.

Script Structure

Albeit being flexible, scripting caries various disadvantages. It affords a considerable knowledge of the corresponding language by the user. Overall script structure can become quickly unclear and hard to decipher. Despite possible variability there is a large number of routine processes that are used regularly. This code tries to address some of those issues.

- It uses *Matlab sections* (denoted by the double-percent symbol '%%' in the text) as structuring element. Each section represents one process step and the overview of currently used processes can be conveniently seen using the 'GoTo' button in the Matlab editor.
- As a result, each Matlab script represents *one particular linear workflow* of image processing tasks. The entire analysis is compactly captured in *one* script file.
- Sections themselves are built following a strict structure. They begin with user input data, followed by definition of parameters and code for the given process, and finish saving results into a data structure.
- A *template section* is provided with instructions for the user which allows to add new processes while maintaining the overall structure of the script.

Ease of Use

If a given workflow can be accomplished solely with preexisting processes that are provided in a template file, the user interaction is limited to adapting the variables defined in the *UPDATE* text block which is clearly marked. It is hoped that this approach will make the script accessible and usable to novice Matlab users. Advanced programmers can incorporate their own functions and use the script as scaffold.

Workflow Documentation

The general goal of this code is to streamline the scripting process and provide at the end a *well-documented workflow*. The code tries to accomplish this task in several ways.

- Intermediate image data are saved using a naming convention that reflects the list of processes applied to the given data set on the operating system level.
- Salient results of each section are saved into a *Matlab structure*. Since such a structure is sequentially built, the order of its various sub structures reflect the workflow of the given script.
- A *Matlab table* is generated with elements documenting the various processing steps, the dimensions of the image series as they change along the way, names of result sub structures and time needed for the execution of each step.
- Wherever useful, analysis results are plotted. In cases where plots are generated for an entire series, avi files are used and saved into the *Figure* folder. Series of images are stored in the MRC format which can be read with image processing tools such as ImageJ. Most single plots are at this moment only shown on the screen and the user can decide to save those as files from the plot window.

The ultimate goal is to provide a certain degree of *transparency*. The documentation elements allow a user to keep track of processes and parameters in a larger project, facilitate the transfer of information to colleagues, and ultimately make the data analysis transparent during the review process of a publication.

Version 1.0

At this moment only the most commonly used processes are implemented. They are all stored in one big template script. A new script is built and modified using copy and paste. A few example scripts are provided as a starting point. They can handle images recorded on a Helios FIB (via Slice & View) and a Crossbeam FIB (via Atlas tomography widget). *Working with the provided code affords some patience during the setup but will hopefully help to keep track of the applied sequence of image processing tasks and relevant parameters*.

Contact

For questions, bug reports and general comments please contact

Stephan Kraemer (skraemer@fas.harvard.edu)
Center for Nanoscale Systems, Harvard University

Script Design

BUILDING A LINEAR WORKFLOW

The script makes use of a series of *Matlab sections* (denoted by the double-percent symbol '%%' in the text). Every section represents *one process step* which follows *strictly* from the previous ones. As a result, each script file contains a specific sequence of processes which build one analysis *session*. Once a session is complete, it is possible to make use of the resulting data that are stored in the 'info' structure to build further sequences.

One potential scenario is to use the same sequence of processes on another signal, if a multi-signal run was performed. In this case, run a second session with a copy of the first script and change the input data from signal one to two. In order to apply the same image alignments that were used on the first data set, skip all pre-processing steps and immediately apply the values from the first session, such as the beam shift values .

Another scenario is to evaluate alternate processing sequences, for example to test the influence of denoising on segmentation results. In this case it is possible to build deviate using section 'SECTION X' and start the script by adding sections with processes that build on the copied ones.

SCRIPT STRUCTURE OF THE FIRST SESSION

The script is designed as a *working tool* and is not meant to be executed as a whole. As such it is divided into *four groups* of sections.

Group 1 . Setup

The first group deals with setting up the script execution and saving information that is pertinent to the specific serial section run. It has to be executed in the following order when a script is run the first time. The sections set up the 'info' structure and define *project folders*.

- 'SESSION 1'
    - This section is the first to be executed when the first session is started. It initializes the info structure and allows to give the script session a clear identifier, for example a combination of user name, run date and sample name. The code checks whether an *info* variable is already present and provides a warning. *Clear the workspace in order to omit overwriting existing data.*
    - Each session operates at this moment only on the data set of *one* detector.
    - The user can set variables that define the global behavior of the script, for example if image indices are printed during a calculation or whether the info structure should be saved automatically at the end of each section that produces relevant data. *The latter is recommended to make sure that the 'info' file is always up to date.*
- 'RUN DATA'
    - Allows to save experimental details of the specific run and formatting of the numbered image files
- 'PROJECT FOLDERS'
    - Defines the working folders. The user has to define the *project folder* in which all the processing results are stored. Additional folders can be added if so desired, for example a storage folder on another local hard drive or a linked network drive. See more information in the chapter *Project Organization*.
- 'PROJECT NOTES'

- 'PROCESS LIST'
  - This section adds the Matlab table *ProcessList* to the info structure. During the execution of relevant sections a corresponding row is added.
  - Once a process list has been generate, it is possible to save the table as an Excel spreadsheet into the project folder. *This allows to see the sequence of processes applied in this specific case without having to open Matlab and loading the info structure. The Excel spreadsheets can then be collected in one big project file if desired.*

Group 2 . Load and Save

The second group of sections deals with saving and retrieving of *info* structure and image data

- 'INFO . Load'
  - This section allows to continue the work at any given time.
  - As in section 'SESSION 1' the user can set variables that define the global behavior of the script.
- 'INFO . Save'
  - Allows to save the *info* structure manually, in case the variable *autoSaveInfo* is set to *false*.
- 'INFO . Edit'
  - This sections contains *instructions* on how to remove the information of the *last* process step in case the user decides to redo the analysis or chooses another analysis route. Information to be removed is the *sub structure* generated by the given process and the corresponding *row* in the ProcessList table.
  - Note: *Deleting the information of an intermediate process step will break the workflow and is not possible within the concept of this script. If a user recognizes a mistake during the execution of an intermediate step, all following processes have to be repeated as well in order to insure that there is no discontinuity of information flow.*
- 'IMAGES . Load from folder'
  - Load image series into the variable IMS.
  - The section also allows to load single images form a series, skip images and reduce dynamic range to 8 bit in case the original images are stored as 16 bit.
- 'IMAGES . Save to folder'

- Save image series to folder, either as TIF in 16 or 8 bit, or as JPG with the option to set the compression factor.

Group 3 . Importing data from raw image folders

The third section group parses the raw data and extracts the region of interest. Currently code is provided to handle data recorded with *Slice & View* on a Helios FIB and with the *Atlas tomography wizard* on a Zeiss Crossbeam FIB. Importing data happens in two to three steps depending on the given case.

1. First, the script reads the *file list* in the image folder and saves the list into a *string array* and orders them following the naming syntax used when the images were saved during the serial section run. The script will use that list to read the corresponding image files. Note that the corresponding image indices used in the script will *not* correspond to the image numbers in the filename if file numbering does not start with 1 and if intermediate images are missing.
2. In a second step the images are extracted from the raw image file, say 'ESB', and saved into 'ESB_Ext' following the naming convention introduced in chapter 'Data Storage'. *This step is specifically needed when the XROI feature is used in the Atlas tomography software. In this case the actual image area is placed inside of the field of view of the electron beam.* For consistency purposes this approach is also used when the image folder contains already images of equal size and containing the actual region of interest. Following processes will use information in the *Extract* sub structure.

The specific set of sections dealing with Helios data is as follows

- 'PREPARE . Helios . Parse file list'
  - Image files stored by the *Slice & View* software were found to have varying image sizes. Furthermore, if the imaging position is changed during the run it is possible that the actual pixel size changes. This section therefore does not only reads the file list but also opens every image file using the *imfinfo* function and extracts image dimensions and pixel sizes from the *meta data*, and saves them into the *Image* sub structure.
- 'PREPARE . Helios . Scale images >> Scl'
  - In case the the pixel size changes during during the serial section run, it is possible to scale and interpolate all images to a common pixel size.
- 'PREPARE . Helios . Extract images >> Ext'
  - Saves images with a common image size.

Crossbeam data are extracted with the following set of functions. In this version the script assumes that the user uses the *XROI* feature in the Atlas tomography software.

- 'PREPARE . Crossbeam . Parse Atlas file list'
    - This section reads the image file list. Additionally it extracts the z-position of the given slice which is included by the Atlas software in the file name and calculates the slice thicknesses across the series.

The user can choose between two extraction methods.

- 'PREPARE . Crossbeam . Extract Manually >> Ext'
    - This section is useful for runs where the image position is not or infrequently changed or if a sub region is only of interest. The user simply defines the boundary box of the to be extracted image region.
- 'PREPARE . Crossbeam . Extract XROI >> Ext'
    - This section is useful if the XROI region was shifted during the run. The script automatically searches for the largest patch in the raw image using morphological operations and calculates either the common envelope or the least common area of all patches across the series, depending on the user's choice.

A section 'PREPARE . Crossbeam . Parse simple file list' can be used if image files just contain slice numbers and if all images have the same dimensions. Following the above rule the extraction section 'PREPARE . Crossbeam . Extract Manually >> Ext' has to be executed by either using the original image dimensions as bounding box or by providing a cropped region.

Group 4 . Processing

The last section contains the actual image processing steps. Chapter 'Key Processes' describes the more advanced processes.

SCRIPT STRUCTURE OF FOLLOWING SESSIONS

As noted at the beginning of this chapter, it is possible to generate alternate workflows that build on previous ones. In this case the info structure of the preceding session will be loaded into the workspace variable 'infoPrev', and relevant sub structures that are needed for the following calculations are copied into the current 'info' variable. A corresponding script contains three groups.

Group 1 . Setup

The info structure is initialized using the information from the preceding section.

- 'INFO . Load'

    - Set here the variable 'previousSession' to *true* and provide the number of the session from which the current one will be derived. The corresponding info structure will be loaded into the workspace variable 'infoPrev'.
- 'SESSION X'
    - This session allows to set the process number up to which the corresponding information should be copied. Then the section will initialize the new *info* structure in two steps
        - Step 1: copy the *Run* and *Project* sub sections and initialize a new *Notes* table.
        - Step 2: Generate a *ProcessList* table copying all entries of the previous list up to the set process number, copy sub sections for all processes, again up to the given process number.

Group 2 . Process Notes and List

The second group contains the section 'PROJECT NOTES' and 'PROCESS LIST' in order to add more notes if necessary and to save notes and process list to an Excel spread sheet at the end of the analysis.

Group 3 . Load and Save

The third group is *identical* to the one described in section 'SCRIPT STRUCTURE OF THE FIRST SESSION'. It handles intermittent saving and retrieving of *info* structure and image data.

Group 4 . Processing

Then processing can start immediately and a new sequence of process steps can be built.

Section Design

Each section represents a specific processing task (such as cropping the images series, denoising, etc.).

- Sections that start with a *number* in the title represent a task that extracts necessary information from or modifies the image series. The numbering allows to guide the reader through the sequence of analysis steps.
- Sections marked with *PLOT* visualize a certain aspect of a previous process step.

STRUCTURAL ELEMENTS OF A SECTION

All sections have the same *layout* of structural elements:

- The 'TASK' line briefly describes the process.
- A 'NEED' line indicates if the analysis needs specific variables, for example the image series, if it is loaded into memory. In general sections were designed to be as autonomous as possible where most variables can be retrieved from the *info* structure and *ProcessList* table (see above). This allows to interrup the analysis and resume at another time. It also allows to apply the same script to a second signal with minimal changes, which can be useful for example in cases where both secondary and backscatter images were recorded during the run. Restrict the use of workspace variables.
- A 'PROC' line describes the intended handling of the sections if more specific instructions are necessary.
- An 'UPDATE' text block contains the input parameters needed for the given task. It is distinguished between WORKFLOW, INPUT/OUTPUT and PROCESS parameters. The user only needs to interact with this part of the section. All following code should run without intervention.
- At the end of the section the *info* structure and *ProcessList table* are updated with input parameters, results of the process and potential changes to the size of the data set.
- The sections use *tic/toc* command pairs to monitor the time spent for each processing step which is saved into the ProcessList table.
- A 'Dates' substructure keeps track of when the script was created, modified, loaded and saved.

BUILDING A NEW PROCESS USING THE TEMPLATE SECTION

As mentioned above, a user can easily generate a new processes using the 'TEMPLATE' section, located at the end of the template file. It contains the main building blocks and instructions on how to modify text and code:

1. Replace text snippets marked by brackets < ... > with the proper text relevant for the given application.
2. Code that needs to be adjusted is marked with instructions % TODO X:...'.

Working with a Script

SCRIPT TEMPLATES

Four example scripts are provided containing a typical set of sections: reading and saving of the info structure, reading and saving of images, initialize the session and info structure, parsing the image folders, denoising and aligning the images.
HANDLING OF HELIOS DATA IS NOT YET UPLOADED TO GITHUB.

- 'SerialDataEval_Session1.m' (in folder 'Helios Templates')
  - The first script parses and loads images recorded on a Helios FIB. The instrument at CNS produces images with negligible scan distortions and basic rigid body displacement via cross correlation of entire images is sufficient to correct image-to-image drift. (See 'GLOBAL DRIFT' section.)
- 'SerialDataEval_Session1.m' (in folder 'Crossbeam Templates')
  - Images recorded on the Crossbeam 550 FIB are prone to scan distortions that vary as function of the slow scan direction y. The section group 'Y-DEP DRIFT' divides the image into horizontal stripes, calculates the cross correlation in each stripe, builds a displacement matrix which is used by the *imwarp* function to correct the displacements. Beforehand images are denoised using the non-local means filter. For more details see corresponding sections in 'Key Operations' chapter.
- 'SerialDataEval_Session2.m' (in folder 'Helios Templates')
  - Example 1 represents a case where two signals (secondary and back-scattered electrons) were recorded simultaneously. This script demonstrates how to build a second session where the alignments of session 1 are applied to the second set of images.
- 'SerialDataEval_Session2.m' (in folder 'Crossbeam Templates')
  - The last example demonstrates how in a session an alternate workflow is applied. It builds on example 2 (Crossbeam data) and applies the scan distortions on raw, un-denoised images.

ADDING SECTIONS

At this moment all available processes are bundled in one file *SerialDataAlignTemplate_All.m*.

1. Build a script by copying Matlab sections from this template file in the sequence as needed by the given data set. Each script represents a processing *session* with *one* given series of processes. Build separate scripts for alternate sequences.

3. Modify all the values for the variables defined in the *UPDATE text block* as needed by the given problem. *Minimize the use of workspace variables. Save all variables of a process that are needed by consecutive processes in the info structure. This allows to end and resume a session at any point.*

SCRIPT NAVIGATION AND EXECUTION

A script can be best viewed in the *Matlab Editor* and easily navigated with the 'Go To' button in the Editor control tab. Use the <Ctrl>-Down and <Ctrl>-Up keyboard shortcut to move from one section to the next, or the corresponding function in the 'GoTo' context menu.
The individual section can be *executed* either by clicking on the *blue ribbon* on the left edge of the text editor that appears when the cursor is placed inside of it or via the 'Run Section' command in the SECTION panel of the Editor tab.

HANDLING OF SERIAL SECTION DATA

Image data can be handled in two different ways, depending on the amount of *memory* available on the computer.

- In the first approach images are successively loaded from the hard drive *at each processing step*, and resulting images are stored back onto the hard drive after each task. In order to speed up the process one can set up a work folder on a SSD hard drive.
- For the second approach the entire data set is loaded into memory. The latter case is intended for speedy evaluation.
    - The initial image series is loaded into the workspace variable 'IMS' using a section 'IMAGES . Load from folder'.
    - The first process that modifies images saves the results into the variable 'IMSP'.
    - Any following processes will transfer the content of 'IMSP' back into 'IMS', again operating on 'IMS' and saving the result into 'IMSP'.
    - Note: There is currently no memory management. If the dataset is loaded into memory, the memory has to be large enough to hold the given number of Matlab clones, two representations of the data, before and after processing, plus overhead for the actual calculations.

STARTING A PROJECT

A project is initialized by executing the 'Session 1' script as described in chapter 'Script Design'. It defines the *project folder(s)*, initializes the *info* structure, *Notes* and

ProcessList table, extracts a region of interest from the raw image data and allows to build a first sequence of processes.

STOPPING A SESSION

Before exiting Matlab the user must ensure that both the latest version of the *info* structure and the *last image series* was saved to the hard drive.

- The info structure is automatically saved at the end of a process if the variable 'autoSaveInfo' is set to *true*. Otherwise it has to be saved manually using section 'INFO . Save'.
- As described above, image series can be handled either by loading and saving intermittent results from and to folders or by loading the initial image series into the workspace variable 'IMS'. In the former case the latest image series is automatically saved, in the latter case the variable 'IMSP' has to be saved using section 'IMAGES . Save'.

CONTINUE A SESSION

If *info* structure and image series were saved properly as described above, the analysis of a dataset can be resumed at any time. Just use the 'INFO . Load' section to load the info structure back into memory. Reviewing the *ProcessList* table allows to ascertain after which process step the analysis was halted.

STARTING ANOTHER SESSION

A new workflow can be started by loading the info structure of the preceding session into variable 'infoPrev' and initializing the new info structure using section 'SESSION X'. Then processing can begin.

Data Storage

SERIAL SECTION DATA

Serial section data are currently solely stored as a *series of TIFF files in a folder*. Two sections ('IMAGES . Load from folder' and 'IMAGES . Save to folder') handle loading and saving of entire data cubes. For inspection purposes, it is possible to load a

NAMING CONVENTION FOR INTERMEDIATE RESULTS

The image series can be saved after significant process steps allowing to analyize the effect of a given processing task. Each series is stored in a folder which starts with the acronym of the currently used signal, e.g. 'ESB','SE2', 'InLens' for Crossbeam data, 'TLD','ICD','CBS' for Helios data. Each process step is described by a three-letter acronym. The acronym is saved in the ProcessList. The folder name contains the sequence of applied processing steps.

As example, the folder 'ESB_ExtDenCroYdd' contains an image series that has undergone the following sequence of processing steps

1. Ext: extract the 'XROI' from a Crossbeam dataset
2. Den: denoise via non-local means averaging
3. Crop: crop series to region of interest
4. Ydd: correct image drift, taking into account y-dependent drift values in case images suffer from drift-induced local scan distortions.

Each section that saves an intermediate result is marked in the title by '>> <ACRONYM>'

PROCESSING PARAMETERS

Processing parameters and results are saved in the *info* structure, which can be stored and reloaded via section 'INFO . Load', 'INFO . Save'. Each section saves parameters into a corresponding sub structures. The info structure is the core element of this script that stores analysis results and contains parameters determining the workflow of this script.

Whenever a process (apart from intermediate steps) is finished, a line in the *ProcessList* table is added, documenting the processing steps so far applied to the dataset. The table also contains information about the data volume size at each step, the name of sub structures in the info structure which contains input parameters and results for the given process step. Notes can be added and the duration of each process is stored.

Project Organization

PROJECT FOLDERS

All data related to one serial section run are stored in a *project folder*. The script allows to distribute the data over several platforms. Their *absolute locations* are

analysis of the serial section run is finished to free up space on the local active drive.
2. local *project* drive: contains currently active projects. It stores the script, info *.mat* file and all intermediate results. Once the analysis of a run is finished the user can decide whether to delete intermediate results or move them to the storage drive to free up space.
3. *work* drive: a drive that allows fast file reading and saving such as a SSD drive. The entire project folder can be moved/copied there temporarily for the time-intensive part of the analysis. This can be useful if the local memory is small and images are continuously read from the hard drive (See chapter 'Working with a script'.)
Examples

```
info.Project.storageFolder = 'F:\User\Stephan\Serial\Projects\2022-12-09 Copper
single particle I';
info.Project.projectFolder = 'D:\User\Stephan\Serial\Projects\2022-12-09 Copper
singel particle I';
info.Project.workFolder = 'C:\WORK\Current Project';
```

PROJECT FOLDER STRUCTURE

The project folder is divided into a few sub folders:

- *Run*
    - All supplementary data that are produced during the preparation and execution of the run, such as log files, Atlas 3D setup files, etc. Typically stored on storage drive.
- *Align*
    - Matlab script 'SerialDataAlign_Session1.m', etc. Corresponding 'info' structure stored as *.mat file 'info-Session1.mat', etc.
- *Figures*
    - Any images and plots that describe analysis results.
- *Segmentation*
    - further Matlab scripts that contain segmentation-related processes such as morphological operations, maybe at some point Deep Learning based segmentation.

- *Image folders*
    - as described above, intermediate serial section data, marked by the name of the detector and applied processes, symbolized by 3-letter acronyms. For example: 'ESB_Ext', 'ESB_ExtCroDen', 'ESB_ExtDenYdd', ...

Standard operating procedures

INSTALL CODE

- ☐ Load repository *zip* file from GitHub
  - ☐ Go to website below
  - ☐ click *Code* button, select 'Download ZIP'

```
https://github.com/StephanKraemer/FIB-SEM.git
```

- ☐ Extract and save repository folder to location of choice
- ☐ Add 'Functions' folder with sub folders to *Matlab path*
  - ☐ Click 'Set Path' in ENVIRONMENT panel of the HOME tab
  - ☐ Use 'Add with Subfolders' in 'Set Path' control pop-up window
- ☐ Generate MEX function from code files stored in 'MEX' folder
  - ☐ Use the *Matlab Coder App* (see Matlab documentation)
  - ☐ Add folder containing MEX functions to *Matlab path*

SET UP PROJECT FOLDER(S)

The instructions below correspond to a setup that distributes the project over two folders, a storage folder on an external drive and the project folder on a local hard drive in which the script results are saved during execution. This selection is arbitrary and can be modified by the user as preferred. In every script section the user specifically defines the folder(s) for reading and saving data.

- ☐ Generate project folder at desired location on local hard drive (assuming the use of a workstation)
- ☐ Generate the sub folders 'Align' and 'Figures'
- ☐ Save the folder(s) containing the raw images into the project folder. The folder name represents the detector(s) used, for example 'ESB', 'SE2', 'InLens' on a Crossbeam or 'ETD','TLD','ICD' on a Helios.

- ☐ Generate a folder with the same name on the external storage drive
- ☐ Generate a folder 'Run' in the project folder on the storage drive
- ☐ Save all files and folders that are generated by the Atlas software on a Crossbeam or by Slice&View on a Helios into the 'Run' folder

BUILD FIRST SESSION SCRIPT

- [ ] Copy one of the script files from the 'Example Script' folder into the 'Align' folder. Choose the one that fits best to your experimental setup.
- [ ] Add more sections from the template file 'SerialEvalTemplate_All.m' if needed.
    - [ ] For every section that was added, change its *section number* in the section title and the corresponding *processStep* variable to generate a coherent series of processes.

START FIRST SESSION AND BUILD INFO STRUCTURE

- [ ] Select the 'Align' folder as the *Current Folder* in the Matlab UI
- [ ] Clear the workspace using *clearvars* function

From now on change all variables in the 'UPDATE' text blocks in every section and execute the section to *commit* the changes:

- [ ] In the first section 'SESSION 1' update *project identifier*, *description* and *detector*
- [ ] Define the desired behavior of the script by setting the *verbose* and *autoSaveInfo* variables. (The latter does only produce minimal time overhead in the execution of each script section and ensures that the saved info structure stays up to date).
- [ ] Executing the section will generate the *info* structure. Either
    - [ ] use keyboard shortcut <Ctrl> + <Enter>
    - [ ] click on the blue ribbon the left side of the section in the Editor (appears after clicking into section to render active)
    - [ ] select 'Run Selection' in the SECTION panel of the EDITOR tab

- [ ] Collect all experimental parameters in the 'RUN DATA' section. Make sure that the variable *nDigits* is properly set in order for the program to load image files later properly.
- [ ] Copy the *absolute path* of the *project* and *storage folders* from the *Windows File Explorer* window into the 'PROJECT FOLDERS'.
- [ ] Execute the 'PROJECT NOTES' section to generate the *Notes* table in the *info* structure.

The info structure is now ready to be used.

PARSE IMAGE FILES ON A CROSSBEAM

In the next step the raw data folders have to be parsed and the images extracted. This process is instrument dependent. A typical workflow for a scenario where the image region was unchanged during the run and the images were recorded with the Atlas software on a Crossbeam is as follows:

- [ ] In 'PREPARE . Crossbeam . Parse Atlas file list' set *sliceIn.projectFolder* to *info.Project.projetFolder. (* Executing the section will generate the *sliceName* variable, a string array that contains all image file names in the raw image folder. The slice thicknesses will be extracted from the filenames and plotted.)
- [ ] (optional) Save the plot as BMP and EPS file in the 'Figures' folder
- [ ] In 'PREPARE . Crossbeam . Extract Manually >> Ext' define the origin and dimension of the image region

PARSE IMAGE FILES ON A HELIOS

A similar process is applied to image series recorded on the Helios FIB, adapted to that instrument. In this example all images have the same pixel size. Therefore a resizing with section 'PREPARE . Helios . Scale images >> Scl' is not necessary.

- [ ] In 'PREPARE . Helios . Parse file list' set *projectFolderRead* to the proper location set in the variable *info.Project.* A structure *Image* is generated with the dimensions of all images in the series. A plot will show the corresponding data.
- [ ] In 'PREPARE . Helios . Extract images >> Ext' set input and output location of the image files in the 'UPDATE -- IN/OUT' text block. In the 'UPDATE -- Process' block define how the images should be extracted setting the variables *area*, *centerX*, *centerY* and *imageClass*.

APPLY PROCESSES

Simply set the variables in the 'UPDATE' text blocks in each section and execute.

INTERMITTENTLY END A SESSION

If the user sets the *autoSaveInfo* variable to *true* and uses the approach saving the intermittent image series results, all the necessary data are already stored on the hard drive. If the user chose to load the image series into memory, the IMSP variable has to be saved as follows

- ☐ In 'IMAGES . Save to folder' set all variables to fit to the image series that is to be saved
  - ☐ set *pStep* to the process number that generated the variable *IMSP*
  - ☐ Set *useFolder* to *false*
  - ☐ design *sliceFolderSave* string following the naming convention given in chapter 'Date Storage': <Detector name>_ followed by the sequence of process acronyms that are listed in the 'Save' column of the *ProcessList*.
  - ☐ set all image dimensions to fit the variable *IMSP*.

CONTINUE A SESSION
All the relevant information is stored in the info structure.

- ☐ Select the 'Align' folder as the *Current Folder* in the Matlab UI
- ☐ load info structure using section 'INFO . Load'

Now processing can resume.

BUILD CONTINUING SESSION

In case an alternate processing workflow is desired the script has to start with sections 'INFO . Load' to load the info structure of the session that the new one is built on, then build the new info structure with section 'SECTION X'. After that processing can resume with a new sequence of processes. See chapter 'Script Design' for the typical structure of a continuing session script.

- ☐ In 'INFO . Load' set *currentSession* to *false*
- ☐ define *session* number which the new session is built on
- ☐ set process control parameters verbose and autoSaveInfo as desired
- ☐ In 'SESSION X' replace X in title with the proper session number and set corresponding variable *info.Session*
- ☐ Give *description* that captures the goals of this session

- [ ] give the process number *procLas*t of the previous session from which to continue the analysis
- [ ] Set the number of the first process in the new session to *procLast* +1 and continue as usual

---

Conventions

COORDINATE SYSTEM

Axes are named following the conventions of the Image Processing Toolkit (and most other image processing software):

image origin: top-left corner of image

x: horizontal cross section image axis

y: vertical cross section image axis (positive down)

z: slice direction

NOTE: The *indices* of the corresponding 3D data cube variable correspondingly represent the y,x, and z axis, e.g. [ny,nx,nz] = size(IMS);

SLICE INDEXING

The script reads and saves the filenames of a data folder in the 'sliceName' string array. The initial set of files contains 'nSlicesRecorded' slices, with indices

```
iStart = 1;
iEnd = nSlicesRecorded;
```

NOTE: The above slice indexing only follows the order of files, given by the formatting of their names. The numbers can differ from the ones in the filenames if those start with a number different from 1.

BOUNDING BOX OF AN IMAGE FRAME

A cropped region is typically defined by its origin *[ox,oy]*, width *w* and height *h*. The origin is defined as the pixel above the top-left corner of the image region, such that the following code accesses the region

```
R = I( oy+1:oy+h, ox+1:ox+w );
```

Alternatively, in some instances the user is asked to provide the *start and end pixel* in both dimensions. The selection is correspondingly

```
R = I( yS:yE, xS:xE )
```

KEY PROCESSES

NON-LOCAL-MEANS DENOISING

The original Matlab function by Buades et al (2006) 'A non-local algorithm for image denoising'

IMAGE ALIGNMENT VIA IMAGE-TO-IMAGE CROSS CORRELATION

GLOBAL DRIFT - Including separation of jumps

Use this set of sections in case images do not suffer from scan distortions and a simple image-to-image drift correction is sufficient. It allows to locate isolated larger jumps that are treated as a separate entity. This step can be skipped if the cumulative drift contains only a slowly varying continuous function with additive noise. The analysis contains the following steps. Key variables are noted

1. Cross correlation

The function 'SerialCrossCorrSeries' uses a Fourier-based cross correlation function to determine the image-to-image drift. A polynomial fit of the cross correlation peak allows sub-pixel resolution.

Two sub-pixel results are provided. The first one does a general polynomial fit and interpolates the peak on a user-defined grid. A second fit uses a polynomial function that contains the peak center as a parameter. The first fit is used to calculate starting values for the fit of the latter function. The second fit allows to estimate the error in finding the peak position. Due to the inherent smoothness of the cross correlation function the error is typically better than a tenth of a pixel.

The function also provides the cumulative drift based on values of either fit 1 or fit 2, depending on the user's choice. The corresponding variables are

```
dfl1 (sub-pixel image-to-image drift on fixed grid)
dfl2 (sub-pixel image-to-image drift parameterized)
dflcum (cumulative sum of either fit 1 or fit 2)
```

2. Separate single jumps from continuous drift
Plotting a histogram of the image-to-image drifts (for example dfl2) leads to a Gaussian distribution resulting from the statistical nature of the drifts occurring during recording of the series. Jumps show up as outliers.

The user provides a threshold value to separate drifts that are considered jumps versus noise given as multiples of the standard deviation of a Normal distribution, e.g. 3sigma or 5sigma. Jumps should only show up sporadically. The resulting variables are stored in the substructure 'info.ContDrift'

```
d1Step, d2Step singular image jumps (1:y-axis, 2:x-axis)
d1Cont, d2Cont continuous drift part with drift noise
```

3. Integrate along image series
The image-to-image drifts are combined via cumulative sum. Individual jumps produce step functions, the continuous part a slowly undulating function. The long-range undulations are a consequence of the sensitivity of the cross correlation to spatial changes of the overall intensity distribution.

```
d1StepCum, d1ContCum
d2StepCum, d2ContCum
d1Comb, d2Comb (sum of both cumulative drifts)
```

4. Remove undulations of continuous cumulative drift
A spline fit is used to separate the slowly varying drift component described above from the actual image-to-image drift which is more statistical in nature. The fit result are stored under

```
info.SplineFit.d1Fit, info.SplineFit.d2Fit
```

NOTE 1
This correction ultimately assumes that there is no systematic shift along the z direction, i.e. that the sample surface is perfectly flat and perpendicular to the ion beam. A deviation of this condition will lead to a distortion of the data cube. Nevertheless, such a systematic variation, if measurable by other means, can be easily added to the shift correction introducing a seperate variable.

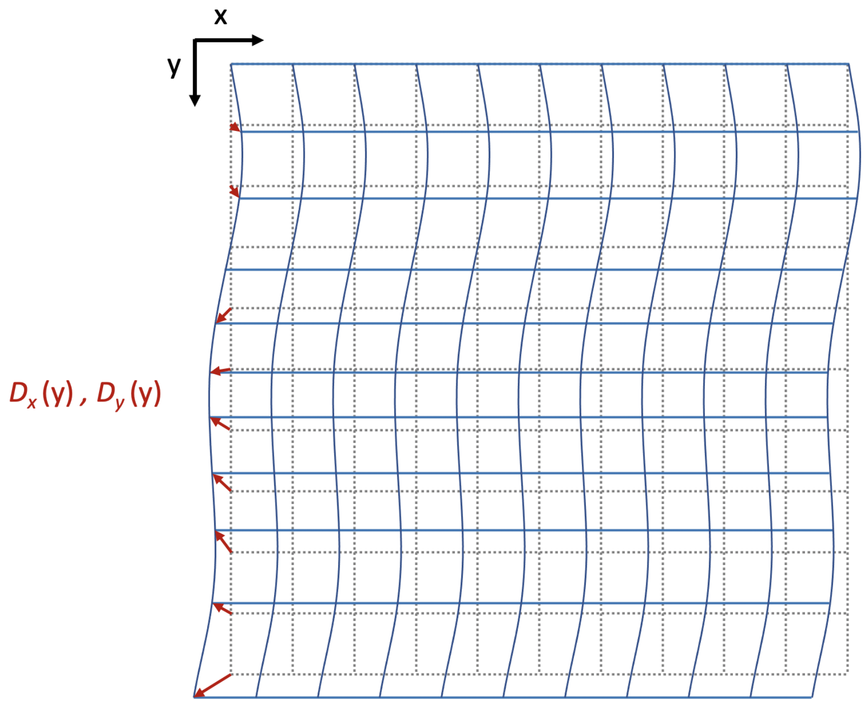
NOTE 2
There is a degree of arbitrariness in the choice of the smoothing factor.

5. Correct shift via circshift function
The ultimate drift correction is calculated by subtracting the spline fit from the total cumulative drift. In this implementation a basic *circshift* is used for the shift. This approach leaves the intensities unaltered due to the pixel-wise shift. The bounds of the corresponding precision of the drift corretion is +-0.5 pixels which appears to be perfectly adequate.

CORRECTION OF Y-DEPENDENT SCAN DISTORTIONS

Serial section image series recorded on the Crossbeam 550 FIB show scan distortions which cannot be described with a simple shear transform. Instead of showing a linear dependence, they rather vary as function of the slow scan scan direction (y) showing independent displacements along the x- and y-direction.
The rationale behind the following group of process steps is that the image acquisition can be divided into a fast scan along the horizontal (x) on order of milliseconds and a slow san along the vertical (y) on order seconds or minutes. Furthermore it can be be observed that (a) the absolute values of the scan distortions are small (a few pixels), (b) vary slowly along the y-direction.
The above conditions allow to split the image into a set of thin horizontal stripes. The function *SerialCrossCorrSeries()* is applied to each individual stripe. For each stripe the same spline fit is used as in the above case of the global drift correction, with the goal to remove the artificial evolution of the cumulative drift and regain the actual image-to-image displacements.
Stripe heights on order 100-200 pixels have been employed and appear to sample the scan distortions on a fine enough grid.
The obtained short range distortions (dx,dy) are collected as function of the center of each stripe along the y-direction and interpolated to pixel-resolution. The corresponding section produces *movies* for both x and y-displacements that allow to cross check whether the y dependence is smooth. So far, this has been the case for all analyzed serial section runs indicating that the above mentioned conditions appear to be valid.
For the actual corrections of the scan distortion it is possible to employ the Image Toolbox function *imwarp()*, which uses a displacement field that can be constructed

from the interpolated local scan distortions. The figure below is illustrating the definition of the displacement field.



Note 1

A necessary condition for the successful use of the above proposed stripe-based cross correlation approach is that each stripe contains a large enough number of objects to average over individual displacements that might be artificially introduced due to the shape change of a dominant object as the cross sections traverse the object. Fortunately, past experiments indicate that the images can be rather coarse in nature with only few objects to still produce reliable displacement values from the cross correlation.

Note 2

Using a reference image calculating the cumulative drift across the entire series inadvertently leaves any potential scan distortion in this image unaltered and imprints it onto the entire series. Fortunately, these distortions are small on the absolute scale and have no effect on the the main objective of the drift correction, i.e. to remove local jitter in order to ensure smooth transition of intensity contours across adjacent slices. A potential strategy to minimize this artifact is to search for three consecutive images with the least relative distortions and choose the central image as reference.

Demo Project

A complete project folder is provided representing a multi-session workflow on data recorded on a Crossbeam FIB. It can be downloaded from
[FIB-SEM Demos]()

The sessions distinguish in the degree of processing: In session 1 images are denoised, drift-corrected and binned to generate isotropic voxels, session 2 omits denoising and in session 3 only scan distortions are corrected. The project contains three sub folders

- 'Align'
  - scripts 'SerialDataAlign_Session1.m', etc.
  - corresponding MAT files 'info-Session1.mat' etc, containing the info structures collecting the input data and results of the corresponding processes
  - Excel files 'ProcessList_Session1.xlsx', etc, tables summarizing the applied processes, image dimensions and duration of each analysis
  - A PowerPoint file 'Summary.pptx', containing run parameters, notes and figures produced during the analysis, illustrating the chosen degree of denoising, and a comparison of yz-slices before and after drift correction.
- 'Figures'
  - Images, plots and movies produced during the analysis
  - MRC files that contain series of images in the original 16-bit format, testing the influence of denoising parameters. They can be opened using ImageJ.
- Image folders
  - The folder names illustrate the syntax described in chapter 'Data Storage'.

Data belong to a research project by Yamilex Acevedo-Sanchez and Rebecca Lamason from the MIT Biology Department, 'An obligate intracellular bacterial pathogen forms extensive and stable interkingdom contacts with the endoplasmic reticulum'. Due to space constraints only three consecutive images containing a region of interest are uploaded to GitHub. The original 16-bit images have been reduced to 8 bit.

An essential contractile ring protein controls cell division in Plasmodium falciparum.
Nature Communications (2019) 10:2181

Rudlaff, Kraemer, Marshman, and Dvorin.
Three-dimensional ultrastructure of Plasmodium falciparum throughout cytokinesis.
PLOS Pathogens (2020) 16(6) e1008587

Moverman, Liberman, Kraemer, Corfas, Liberman
Ultrastructure of noise-induced cochlear synaptopathy.
Scientific Reports (2023) 13:19456

Acknowledgements

For denoising the script uses the original function by Buades et al (2006) 'A non-local algorithm for image denoising'. Handling of MRC image files is based on the EMIODist package, available on GitHub and originally developed by Fred Sigworth and Liguo Wang at Yale University, distributed by Mathworks.