bmiptools: BioMaterials Image Processing Tools

Luca Curcuraci[1¶], **Richard Weinkamer**[1], **and Luca Bertinetti**[2]

1 Max-Planck Institute of collodis and interfaces, Potsdam, Germany **2** B CUBE - Center for Molecular Bioengineering -Technische Universität Dresden, Germany ¶ Corresponding author

DOI: 10.21105/joss.04859

Software

- Review
- Repository
- Archive

Editor: Aoife Hughes

Reviewers:

- @taw10
- @mooniean

Submitted: 26 September 2022
Published: 27 November 2022

Summary

Image-processing is a fundamental step in many scientific imaging techniques, and it is particularly critical in tomographic techniques. In most cases, the aim of the processing is to 'clean' the images from artifacts inherent to the technique and to facilitate a smooth and fast segmentation of the volumetric datasets. Focused ion beam/scanning electron microscopy (FIB/SEM) based tomography, which is gaining more and more importance in the fields of structural biology and biological materials, is one of these techniques where the 'digital cleaning' of the images becomes a fundamental step to obtain visually understandable and easy-to-segment volumetric datasets. In this case, the need for the digital cleaning arises from the sample inherent mechanical properties, the sample preparation and from the imaging workflow. Especially, FIB/SEM-based volume imaging performed under cryogenic conditions (cryo-FIB/SEM) is afflicted by certain systematic artifacts that heavily compromise the readability of the images. Bmiptools (BioMaterials Image Processing Tools) is a Python package designed for addressing this issue, with a series of plugins tailored to remove/mitigate the typical artifacts present in (cryo-)FIB/SEM image stacks. Bmiptools can be easily installed and used both via a Python API or via a simple GUI, making it accessible both to expert and non-expert users.

Statement of needs

Bmiptools is a Python package which can be used to perform image-processing of FIB/SEM image stacks typically acquired on biological tissues and organisms (although it is not restricted to this context). Bmiptools has been developed to meet the following three main requirements:

- Bmiptools implements a series of correction methods specific for typical FIB/SEM artifacts like charging, curtaining, slow variation of brightness within an image, or the abrupt brightness variation between two consecutive slices in a 3D image (called stack).
- Whenever it is possible and makes sense, the parameters used in the correction algorithm to eliminate/reduce the artifacts should be found in an "objective manner", in order to reduce the "human bias" in the selection of the parameters used for the image-processing.
- Bmiptools should be simple to use for those having minimal coding experience with Python. Moreover, users without coding experience should be able to use most of the bmiptools functionalities by means of a suitable GUI.

In addition to these three main requirements, other practical criteria are that the parameters describing the applied transformations need to be stored in an ordered and human understandable way, so that the user can easily follow the 'history' of the modifications the images have been subjected to. To fulfill a basic principle of (computational) reproducibility of the results, the sequence of transformations applied to the images (called pipeline, from now on), can be easily saved and loaded in bmiptools. When a pipeline is saved, all the transformations applied are stored in a proper file. When the pipeline is loaded, all the transformation are applied in the same order and with the same parameters to the data. If applied to the original set of images, consequently the exact the same result is produced. This ensures reproducibility by any other scientist, and allows to save storage space, as in principle only the pipeline object and initial images are needed to reproduce the output.

Bmiptools is open to external contributions: even a user with a basic coding experience should be able to implement their own custom extensions, integrable in bmiptools with minimal efforts. The user can decide to keep the original metadata of the input images during the processing, since this information can be used in the transformations applied to an image. Once the processing is done, the metadata are updated with all the image-processing parameters and settings used in bmiptools, so that the image-processing information does not get lost. However, due to the vendor/machine/experimenter/institute-dependent nature of the metadata, only limited support is available at the moment. In the current software ecosystem for image-processing of FIB/SEM images, one can find commercial software, like Amira (Stalling et al., 2005) or Dragonfly (Makovetsky et al., 2018), or open source alternative for image-processing, like Fiji (Schindelin et al., 2012), that implement many (but not all) of the algorithms of bmiptools. Some transformations implemented in bmiptools can be clearly realized also with these programs. However, the optimization procedures, which in bmiptools are used to find the transformation parameters in an objective manner, are missing in the above-mentioned packages. Popular libraries like scikit-image (Van der Walt et al., 2014) or openCV (Bradski, 2000) can be used to work at low-level on FIB-SEM images, but they are (correctly) conceived as general-purpose image-processing tools. Therefore, they neither have specific transformations for FIB-SEM images, nor possess the high-level functionalities of bmiptools (despite they are extremely useful tools in FIB-SEM image processing as bmiptools shows).

A brief overview of the software

Bmiptools can be easily installed via PyPI. It has an extensive documentation, which contains both the theoretical explanation of each correction method implemented and practical information, like how to install it, how to use its functionalities, or how to contribute with new custom tools. Examples, tutorials, and case studies are also provided in the bmiptools documentation.

Stacks and pipelines

Bmiptools is organized around two basic objects: the stack, and the pipeline objects. A stack is an object containing the collection of images, which have to be corrected, corresponding to the raw output of the microscope. Bmiptools assumes the images to be in tiff format. A pipeline is simply an object which applies (and eventually optimize) automatically, a series of transformations (referred to as Plugins) selected by the user to perform the necessary corrections.

Plugins

Bmiptools is equipped with a series of plugins, which can be used to apply a correction method to a stack. These plugins are the basic building block of a pipeline, but they can be applied directly to a stack (without the necessity of using any pipeline), allowing a more low-level control of the various transformation steps. The currently available plugins are:

- `Standardizer`: rescales the gray level histograms of the slices of a stack to a target range.
- `HistogramMatcher`: matches the histograms of the slices of a stack, effectively removing sudden brightness variations between slices.
- `Denoiser`: reduces the noise level of the slices using classical denoising techniques (Buades et al., 2011; Chambolle, 2004; Chang et al., 2000; Donoho & Johnstone, 1994;

Paris et al., 2009; Sudha et al., 2007).

- `DenoiserDNN`: reduces the noise level of the slices using the Noise2Void approach (Krull et al., 2019).
- `Destriper`: eliminates curtaining artifacts, typical of FIB-SEM images (Münch et al., 2009).
- `Flatter`: corrects for smooth brightness changes within a slice.
- `Decharger`: reduces charging artifacts, typical of cryo-FIB/SEM images.
- `Registrator`: aligns the slices of a stack to ensure 3D structural continuity of objects (Evangelidis & Psarakis, 2008; Le Besnerais & Champagnat, 2005; Reddy & Chatterji, 1996).
- `Affine`: performs a generic affine transformation (e.g., a rotation) on a stack.
- `Cropper`: crops a specific region of a stack.
- `Equalizer`: enhances the contrast in an image using the CLAHE algorithm (Zuiderveld, 1994).

Bmiptools allows users to extend this list by creating their custom plugins and to integrate them permanently via a simple plugin installation procedure.

The bmiptools GUI

The bmiptools GUI is simple and intuitive and, therefore, should be useable almost without explanation once the basic functioning of bmiptools is clear. Information about how to use the GUI is available in the documentation. There one can also find videos showing how to perform basic operations.

Plugin optimization in bmiptools

Most of the bmiptools plugins can be optimized, in the sense that the software is able to find the "best" parameter combination automatically once the parameters space has been defined. "Best" refers to a loss function, which is chosen and then subsequently minimized during the optimization. The loss function of each transformation is discussed in the documentation of each plugin. Some of these loss functions are known from literature (e.g. the denoiser plugins are optimized according to the self-supervised approach "noise2self" (Batson & Royer, 2019)), while other loss functions are proposed for the first time. For newly proposed functions, a systematic study of the loss behavior is presented. For example, for the selection of the decharger and destriper parameters, novel self-supervised losses are employed, and in the "Miscellaneous" section of the documentation two corresponding case studies are discussed. There, all the implementation details about these newly proposed losses can be found, and examples of application to typical use case are discussed.

Acknowledgements

The authors acknowledge support from BiGmax, the Max Planck Society's Research Network on Big-Data-Driven Materials Science. The autors thank Peter Fratzl for helpful input. LC thanks Emeline Raguin for the time spent in the early software debugging, Emeline Raguin and Franziska Jehle for some sample images used in the test phase, and René Genz for the useful discussion about software licences.

References

Batson, J., & Royer, L. (2019). Noise2self: Blind denoising by self-supervision. *International Conference on Machine Learning*, 524–533. https://doi.org/10.48550/arXiv.1901.11365

Bradski, G. (2000). The openCV library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, *25*(11), 120–123. https://elibrary.ru/item.asp?id=4934581

Buades, A., Coll, B., & Morel, J.-M. (2011). Non-local means denoising. *Image Processing On Line*, *1*, 208–212. https://doi.org/10.5201/ipol.2011.bcm_nlm

Chambolle, A. (2004). An algorithm for total variation minimization and applications. *Journal of Mathematical Imaging and Vision*, *20*(1), 89–97. https://doi.org/10.1023/B:JMIV.0000011325.36760.1e

Chang, S. G., Yu, B., & Vetterli, M. (2000). Adaptive wavelet thresholding for image denoising and compression. *IEEE Transactions on Image Processing*, *9*(9), 1532–1546. https://doi.org/10.1109/83.862633

Donoho, D. L., & Johnstone, J. M. (1994). Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, *81*(3), 425–455. https://doi.org/10.1093/biomet/81.3.425

Evangelidis, G. D., & Psarakis, E. Z. (2008). Parametric image alignment using enhanced correlation coefficient maximization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *30*(10), 1858–1865. https://doi.org/10.1109/TPAMI.2008.113

Krull, A., Buchholz, T.-O., & Jug, F. (2019). Noise2void-learning denoising from single noisy images. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2129–2137. https://doi.org/10.48550/arXiv.1811.10980

Le Besnerais, G., & Champagnat, F. (2005). Dense optical flow by iterative local window registration. *IEEE International Conference on Image Processing 2005*, *1*, I–137. https://doi.org/10.1109/ICIP.2005.1529706

Makovetsky, R., Piche, N., & Marsh, M. (2018). Dragonfly as a platform for easy image-based deep learning applications. *Microscopy and Microanalysis*, *24*(S1), 532–533. https://doi.org/10.1017/S143192761800315X

Münch, B., Trtik, P., Marone, F., & Stampanoni, M. (2009). Stripe and ring artifact removal with combined wavelet—fourier filtering. *Optics Express*, *17*(10), 8567–8591. https://doi.org/10.1364/OE.17.008567

Paris, S., Kornprobst, P., Tumblin, J., Durand, F., & others. (2009). Bilateral filtering: Theory and applications. *Foundations and Trends® in Computer Graphics and Vision*, *4*(1), 1–73. https://doi.org/10.1561/0600000020

Reddy, B. S., & Chatterji, B. N. (1996). An FFT-based technique for translation, rotation, and scale-invariant image registration. *IEEE Transactions on Image Processing*, *5*(8), 1266–1271. https://doi.org/10.1109/83.506761

Schindelin, J., Arganda-Carreras, I., Frise, E., Kaynig, V., Longair, M., Pietzsch, T., Preibisch, S., Rueden, C., Saalfeld, S., Schmid, B., & others. (2012). Fiji: An open-source platform for biological-image analysis. *Nature Methods*, *9*(7), 676–682. https://doi.org/10.1038/nmeth.2019

Stalling, D., Westerhoff, M., Hege, H.-C., & others. (2005). Amira: A highly interactive system for visual data analysis. *The Visualization Handbook*, *38*, 749–767. https://doi.org/10.1016/B978-012387582-2/50040-X

Sudha, S., Suresh, G., & Sukanesh, R. (2007). Wavelet based image denoising using adaptive thresholding. *International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007)*, *3*, 296–300. https://doi.org/10.1109/ICCIMA.2007.305

Van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., & Yu, T. (2014). Scikit-image: Image processing in python. *PeerJ*, *2*, e453. https://doi.org/10.7717/peerj.453

Zuiderveld, K. (1994). Contrast limited adaptive histogram equalization. *Graphics Gems*, 474–485. https://doi.org/10.1109/ICACCI.2014.6968381