Super Quick Image Deconvolution with SQUID

Algorithm Engineering 2025 Project Paper

Christoph Manitz

Friedrich Schiller University Jena

Germany

christoph.manitz@uni-jena.de

ABSTRACT

This work presents the development of SQUID, an open-source command-line-based tool designed for super quick deconvolution of microscopy images. Our tool is written in C++ and allows users to input image data and point spread functions (PSFs) via command-line arguments or JSON- formatted configuration files. The tool supports both measured and synthetic PSFs. It allows to set up image grid processing, to select various deconvolution algorithms, and to handle Tagged Image File Format (TIF) images, the most commonly available image format of microscopy data. The tool implements the Richardson-Lucy and Richardson-Lucy with Total Variation algorithm, along with options for parameter tuning, such as number of iterations. While SQUID provides flexibility in image processing, performance evaluations indicate that it keeps up with existing tools like Huygens Professional and DeconvolutionLab2, or is even faster, though it introduces artifacts in certain scenarios. Future works will aim to optimize performance by using GPU acceleration. Overall, our tool should serve as a fast, modular, and open-source alternative to other existing tools. The source code can be found at https://github.com/Griestopff/squid.

KEYWORDS

deconvolution, grid processing, fast fourier transform

1 INTRODUCTION

A microscope magnifies objects that are smaller than the resolution of the human eye. With the application of cameras and other photon detection systems, microscopes also allow the recording of such magnified view. The resulting images are discrete sequences of numerical values arranged in image points, also known as pixels, recorded as a 2- or 3-dimensional function $f[n]$, where n is the number of pixels that are arranged in a matrix [2]. The images acquired by a microscope are never of complete fidelity. Optical systems by default cause aberrations(Figure 2), and the sample itself contributes to such effects by absorbing and scattering photons(Figure 3). Due to their complex biological structure optical aberrations can be especially strong in Organ-on-Chip (OoC) samples for example, where different cell layers and cell types exhibit different optical behavior. All these effects lead to lowered fidelity image, which does not reproduce the ground truth image. The resulting image aberrations can be mathematically described using the concept of convolution, which can be described by a point spread function (PSF). In this paper, we develope a open-source tool which handle this problem even or faster than exisiting tools (Figure 1).

1.1 Related Work and our contribution

The Huygens Professional Software [4] is well known for its fast processing and uses graphical processing unit (GPU) acceleration [5]. However, acquisition costs are high, ranging from one time purchases around $50000 to monthly license rental fees around $110 [13]. DeconvolutionLab2 [12] by the Biomedical Imaging Group of EPFL is written in Java, providing an open-source software. It has to be linked to imaging software platforms like ImageJ [4], Fiji [3] or ICY [1], and runs as a stand-alone application. Our contribution with this project is to develop a deconvolution tool that combines high-performance and open-source. It will consider the FAIR Guiding Principles [7] for the reuse of the data and the software to make it accessible and modifiable for everyone.

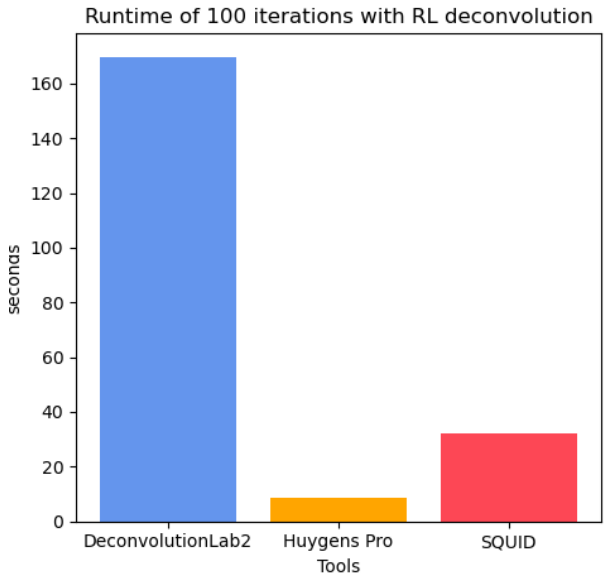


Figure 1: Runtime in seconds vs the number of iterations. (Table 1)

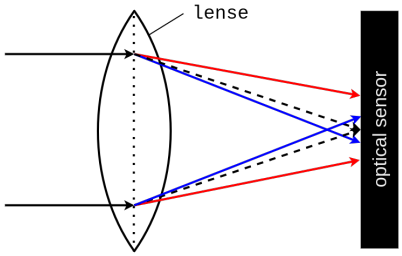


Figure 2: Effect of chromatic aberration on polychromatic light. Polychromatic light (black arrows) hit a lense. Red light (red arrows) is refracted differently compared to blue light (blue arrows). The refracted light does not concentrate on one point on the optical sensor (black rectangle) and produces a blurred image where the various wavelengths are focused at various points along the optical axis (black dotted arrows).

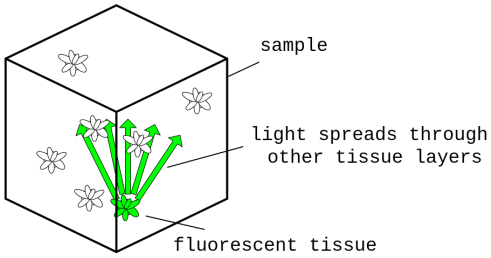


Figure 3: Fluorescence light spreading through multiple tissue layers. A tissue sample (cube) contains fluorescent molecules (white and green objects). If a molecule located in a deeper layer is excited (green object), the resulting light (green arrows) needs to travel through the thick tissue in order to reach the optical sensor.

1.2 Background

1.2.1 Point spread function. The point spread function (PSF) characterizes how a point source of light is represented in the image. The term is derived from the fact that every physical optical system extends a point of light (Figure 4). The PSF is the kernel for the convolution operation and the characteristics depend on various factors such as the numerical aperture of the microscope lenses, the wavelength of light and the refractive index of the medium [2, 5.5]. Accurate modeling or estimation of the PSF is crucial for effective restoration of an image for the most deconvolution algorithms.

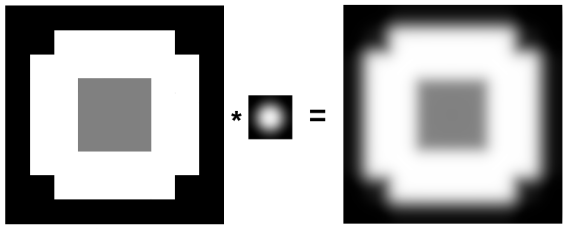


Figure 4: Convolution process. Convolution of an image (left) with a gaussian PSF (middle), where * denotes the convolution operator. The result is a convolved image (right).

1.2.2 Convolution. Convolution of two functions $f(t)$ and $g(t)$ is a mathematical operation that describes how the shape of one function is modified by another and is defined as

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau \text{ [2, 4.2.5].}$$

In the context of image processing, convolution is used to apply filters to images. A filter or kernel in image processing is a small matrix applied to an image to modify it or extract specific features, like edges or textures. It works by performing a convolution operation, sliding across the image and computing weighted sums to enhance or detect patterns. For discrete number sequences $f[n]$, the convolution involves the multiplication and summation of overlapping sections of the array with a kernel as it moves across the array

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m] \text{ [2, 4.2.5].}$$

If g is the observed image, f is the original image, and h is the PSF, the convolution of the image can be expressed as

$$g = f * h.$$

This relationship is mathematically basic but computationally slow. For a 3D image of size $n \times n \times n$ and a kernel of size $k \times k \times k$, there are k^3 operations per voxel, leading to a computational complexity of $O(n^3 \cdot k^3)$. However, the process can be significantly accelerated using the frequency domain, as discussed in the following sections.

1.2.3 Fourier transform. The Fourier transform (FT) [2, 4.2.4] is a mathematical tool used to transform a function from its original domain, for example time or space, into a representation in the frequency domain. Every function in the frequency domain can be split up into different sine and cosine waves with coefficients. Any complex wave can be broken down into a sum of simpler, individual waves to approximate the original function (see Figure 5).

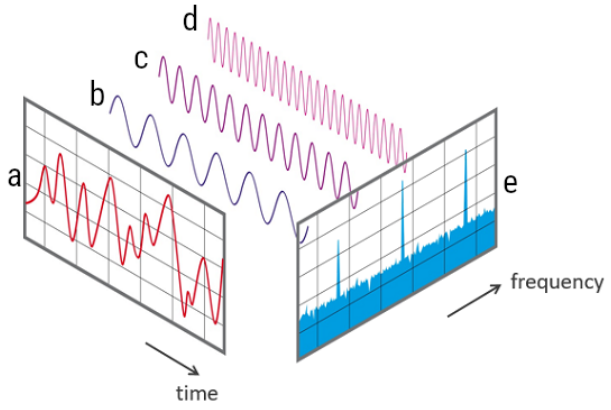


Figure 5: Visualization of signal composition. The time domain shows a signal (a) composed of other wave functions (b, c and d). The frequency domain (e) shows the contribution of the individual wave functions (b, c, and d) to the total signal in the time domain (a). Adapted from [8].

The exponential kernel $e^{-2\pi ikx}$ is used in the continuous Fourier transform to convert a function $f(x)$ from the time (or spatial) domain to the frequency domain. In the Fourier transform formula

$$\mathcal{F}(k) = \int_{-\infty}^{\infty} f(x) e^{-2\pi ikx}\, dx$$

the term $e^{-2\pi ikx}$ acts as a complex exponential kernel that helps decompose the function $f(x)$ into its frequency components. This kernel is complex because it combines sine and cosine waves through Euler's formula, $e^{i\theta} = \cos(\theta) + i\sin(\theta)$, where $\cos(\theta)$ and $\sin(\theta)$ are represented by coefficients a and b, as the structure of a complex number. By using complex exponentials, the Fourier transform efficiently represents and manipulates these waves, simplifying the process of analyzing different frequency components within the function. k can be interpreted as the frequency at which we are analyzing the signal. For continuous FT, k is a continuous variable representing the frequency.

For discrete Fourier transform (DFT), k typically represents discrete frequencies or frequency bins [2, 4.4]. For discrete data, which is typical in digital image processing, the DFT is used. The DFT of a sequence $f[n]$ is defined as

$$\mathcal{F}[k] = \sum_{n=0}^{N-1} f[n] e^{-2\pi ikn/N},$$

where $\mathcal{F}[k]$ represents the DFT of f[n], *f[n]* is the discrete sequence in the spatial domain, k is the frequency index, N is the number of data points in the sequence and $e^{-2\pi ikn/N}$ is again the complex exponential kernel.

The inverse discrete Fourier transform (IDFT) allows to transform data back from the frequency domain to the spatial domain. The Fast Fourier transform (FFT) is an efficient algorithm to compute the DFT and IDFT, significantly reducing the computational complexity from $O(n^2)$ to $O(n \log n)$ [2, 4.11.3] in 1D. This leads to a computationally complexity of $O(n^3 \log n)$ for a 3D image. The convolution can be performed more efficient in the frequency domain with this algorithm, as explained in the next section.

1.2.4 Convolution theorem. The convolution theorem [2, 4.2.5] is a fundamental result in the field of Fourier analysis. It states that the FT of a convolution of two functions

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$$

is the pointwise product of their individual FT. f and g are functions and $\mathcal{F}\{x\}$ denotes the FT of a function x.

Instead of performing a direct convolution in the spatial domain, which can be computationally expensive, it can be applied in the frequency domain. A matrix multiplication can be executed more efficiently with a computational complexity of $O(n^3)$ for a 3D image of nxnxn pixel. After that it is possible to apply the inverse FT to obtain the result in the spatial domain.

Because the complexity of the DFT and the IDFT is $O(n^3 \log n)$ and of the matrix multiplication is $O(n^3)$, the complete complexity of the convolution in the frequency domain corresponds to that of the DFT and is lower than the complexity of $O(n^3 \cdot k^3)$ of the convolution in the time domain.

The property of a multiplication also allows a deconvolution in the frequency domain to be simpler than in the spatial or time domain.

1.2.5 Deconvolution. Deconvolution is the process of reversing the effects of convolution on recorded data. In microscopy, this means removing the blurring introduced by the PSF to recover the original image. The goal of deconvolution is to solve for the ground truth of an observed image. The components in 3D are defined as continuous functions

$$f : \mathbb{R}^3 \to \mathbb{R}, \quad (x, y, z) \mapsto f(x, y, z)$$
$$g : \mathbb{R}^3 \to \mathbb{R}, \quad (x, y, z) \mapsto g(x, y, z)$$
$$h : \mathbb{R}^3 \to \mathbb{R}, \quad (x, y, z) \mapsto h(x, y, z),$$

where f denotes the ground truth image, g the observed image and h the PSF.

The convolution, with the convolution operator *, of the ground truth image f and the PSF h creates the observed image g

$$g = f * h.$$

The FTs of the three components are expressed as

$$F = \mathcal{F}\{f\}$$
$$G = \mathcal{F}\{g\}$$
$$H = \mathcal{F}\{h\}.$$

With these components the image can be recovered from the observation using a naive inverse filter.

According to the convolution theorem, the convolution can then be represented in the frequency domain as

$$G = F \cdot H.$$

Thus, we can recover the original image by dividing the transformed disturbed image G by the transformed PSF H

$$F = \frac{G}{H}.$$

The naive inverse filtering approach however does not generally yield the expected results [2, 5.7], due to small (close to zero) values of the PSF H. Division by these small values lead to very large numbers, which cause hardware-related issues due to limited precision. Additionally small amounts of noise in the recorded image g or PSF h can lead to large errors in the recovered image f. To mitigate this, a modified version of the inverse filter can be used, where a small constant ϵ is added to the denominator

$$F = \frac{G}{H + \epsilon}.$$

However, the convolution theorem is defined in continuous space and 3D images are represented as a collection of voxels in a discrete functions

$$V[x, y, z] \in \mathbb{R}.$$

This implies that, theoretically, perfect recovery cannot be achieved with this approach. Therefore, naive inverse filtering is rarely used in practice [2, 5.7]. Instead, more sophisticated methods for example iterative deconvolution techniques are employed to stabilize the inversion process and produce more accurate and robust results by approximation.

The Richardson-Lucy algorithm is an iterative method based on Bayesian inference [6, 11]. It aims to maximize the likelihood of the observed data given the PSF. The Richardson-Lucy algorithm iteratively updates the estimate of the original image using the update rule

$$f_{n+1} = f_n \left[\frac{g * h^*}{f_n * h} \right],$$

where f_n is the estimate of the original image at iteration n, g is the observed image, h is the PSF, and h^* is the transposed PSF. Transposing a matrix involves reversing its spatial coordinates along each axis. The division and convolution are performed element-wise.

Using the convolution theorem this can be written in the frequency domain as

$$F_{n+1} = F_n \left[\frac{G \cdot H^*}{F_n \cdot H} \right].$$

The algorithm typically starts with an initial guess, often a uniform or blurred version of the observed image, and iteratively refines this estimate. The Richardson-Lucy algorithm is particularly effective for Poisson noise, which is common in low-light microscopy. In each iteration, the FT of the estimated image f_n is computed, allowing it to be multiplied by the PSF in the frequency domain to form the convolution in the spatial domain. The result is then inverse-transformed back to the spatial domain for element-wise division with the observed image. The resulting quotient is subsequently convolved with the flipped of h. For this, the quotient is again transformed into the frequency domain, where it is multiplied by the conjugate PSF, because flipping in the spatial domain results in complex conjugation in the frequency domain. This so-called correction factor is then transformed back into the spatial domain and multiplied with the estimated image to update it. The following pseudocode shows a detailed look at the implementation:

(1) **Initialization**

- g = observed image
- h = Point Spread Function (PSF)
- f_n = estimated image after n iterations (initialized with g)

(2) **FFT of the PSF and the observed image**

- $H = \mathcal{F}\{h\}$
- $G = \mathcal{F}\{g\}$

(3) **Iterations**

(a) **First transformation**

- $F_n = \mathcal{F}\{f_n\}$
- $F_n' = F_n \cdot H$
- $f_n' = \mathcal{F}^{-1}\{F_n'\}$

(b) **Calculation of the Correction Factor**

- $c = \frac{g}{f_n'}$ (element-wise division)
- $c = \max(c, \epsilon)$ (to avoid division by zero)

(c) **Second transformation**

- $C = \mathcal{F}\{c\}$
- $C' = C \cdot \mathrm{conj}(H)$
- $c' = \mathcal{F}^{-1}\{C'\}$

(d) **Update the estimated image**

- $f_{n+1} = f_n \cdot c'$ (element-wise multiplication)

(4) **Result**

- f_{n+1} is the deconvolved image after the iterations.

For better understanding C represents $\frac{G}{F_n \cdot H}$ and C' represents $C \cdot H^*$ from the formula at the beginning of the section. The main advantage of the Richardson-Lucy algorithm is its ability to handle significant levels of noise and its basis in a well-defined statistical framework [14]. However, it can be computationally intensive, requiring many iterations to converge to a satisfactory solution.

1.2.6 Image comparison. The quality and precision of a deconvolution process can be estimated if a reference or ground truth image is available. A ground truth image shows the real view of a sample. The quality can be checked with synthetic generated images and their distorted variants. Two images can be compared pixelwise using the the Wasserstein distance [9], also known as the Earth Mover's Distance. It is a metric that quantifies the difference between two probability distributions. It can be particularly useful for comparing digital images by treating their pixel intensity distributions as probability distributions. The approach quantifies how an image can be transformed into another

The Wasserstein distance between two one-dimensional probability distributions of images P and Q is defined by the formula

$$W(P,Q) = \inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{(x,y)\sim\gamma} \left[\|x - y\| \right].$$

In this formula, P and Q represent the probability distributions of pixel intensities in two images. The marginal distributions are the frequency distributions of these intensities. The set $\Pi(P,Q)$ denotes all possible couplings, meaning all possible assignments of intensity values from P to intensity values from Q, ensuring that the marginal distributions remain intact. The symbol γ represents a specific coupling, and $\mathbb{E}_{(x,y)\sim\gamma}$ denotes the expected value computed over the distances between the assigned intensities x and y. The expression $\|x - y\|$ indicates the distance between the intensity values x and y, which in this context is the absolute difference between the pixel intensities. By minimizing over all possible couplings, the "least effort" required to transform one distribution into the other is determined.

2 THE ALGORITHM

Initially, the arguments are parsed from the configuration file to set up the parameters required for the image processing workflow. This step ensures that all relevant settings, such as algorithm configurations and file paths, are correctly read and utilized throughout the process. To ensure consistency and facilitate processing, all images are converted to a uniform data type. Specifically, they are normalized to a floating-point representation, where pixel values are scaled to a range from zero to one. This normalization process is crucial, as it standardizes the input data and enhances the performance of the deconvolution algorithms, allowing for more accurate and reliable results.

The images are stored as a `Hyperstack` object, which is designed to manage the complex data associated with multi-channel imaging. This `Hyperstack` object contains attributes such as a vector of channels and metadata that describe the characteristics of the dataset (Figure 14). Each channel within the `Hyperstack` is represented by a `Channel` object, which includes an identifier and an associated `Image3D` object. The `Image3D` object is crucial for handling three-dimensional data and features an attribute called `slices`, which is a vector of `cv::Mat` objects. This vector contains the various layers or slices of the 3D image, allowing for efficient access and manipulation of the individual layers within the volumetric dataset. Additionally, a `PSF` class has been implemented, which also includes an `Image3D` attribute.

In the modular design of the deconvolution algorithm, a generic class named `DeconvolutionAlgorithm` is utilized, enabling the seamless integration of various deconvolution algorithms. This class is structured to accept a specific algorithm type through templates, providing high flexibility in selecting and using different algorithms. The constructor takes a configuration (`DeconvolutionConfig`) and any additional arguments. It then creates an instance of the provided algorithm type using `std::make_unique`, managing the instance's lifetime through a `unique_ptr` (pointer). The algorithm's configuration is set by invoking the `configure` method, which establishes specific parameters for the deconvolution process. A key component is the abstract base class `BaseDeconvolutionAlgorithm`, which defines an interface that all specific deconvolution algorithms must implement. This class contains virtual methods for deconvolution and configuration, allowing for a clear separation between algorithms and their implementation. An example of such an algorithm is the `RLDeconvolutionAlgorithm` class, which inherits from `BaseDeconvolutionAlgorithm` and provides specific implementations for deconvolution and configuration (Figure 13). This class includes parameters such as the number of iterations and various settings relevant to the Richardson-Lucy algorithm. Through this modularity, users can select and customize different algorithms, offering significant flexibility and adaptability in image processing. Overall, this design facilitates easy extensibility and adaptation to diverse requirements in image processing while reducing complexity and dependencies among the various algorithms.

The general workflow for the image processing and preparation for the deconvolution starts with extending the input image (Figure 6) at its borders. The tool provides three different ways to extend the border. The first one is to repeat the last value of the border, the second is to mirror the image on the edge and the last on is to fill the extended space with zeros. After this the extended image will split into subimages. The subimages will be adjusted to the PSF with its safety border (Figure 7), which is a border around the image filled up with zeros. If its fits into the subimage size (Figure 7a) then the safety border will be adjusted to the subimage size. If the safety border or the PSF size is larger than the size of the image Figure 7b, than the area of the subimage will be adjusted and will extend the subimage size from the extended image. These subimages will be processed separately for the deconvolution. After the deconvolution the subimages will be cropped to the initial size of the subimage, if the PSF enlarged the subimage. In the end all subimages will be merged together into one image and be cropped to the original image size (Figure 6).

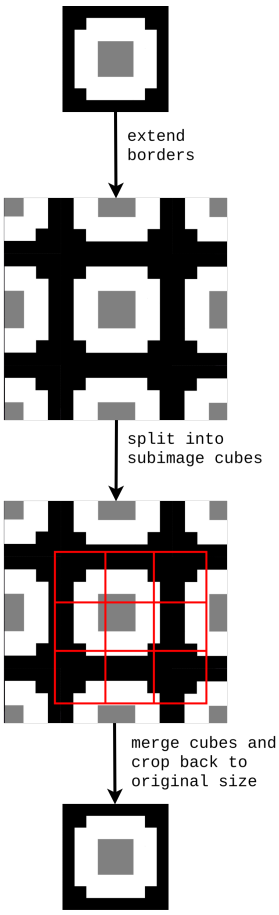
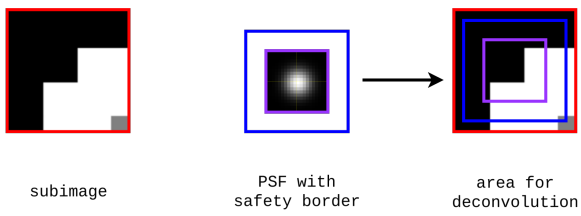
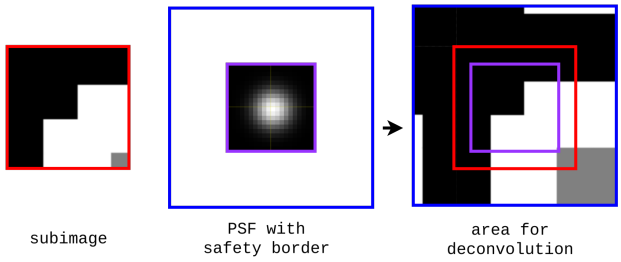


Figure 6: General workflow of Super Quick Image Deconvolution (SQUID) illustrated in 2D. The image at the top is the input image. The extended image will be split into subimages, which are represented by the red squares. The last image represents the result after merging the processed subimages back into a single image.



(a) No extension of the subimage if the PSF with the safetey border is smaller than the subimage.



(b) Extension of the subimage if the PSF with the safety border is larger than the subimage.

Figure 7: General workflow of subimage processing of SQUID illustrated in 2D. The red border represents the subimage boundary, the violet border represents the original PSF boundary, and the blue border represents the original PSF boundary including the safety border size. The deconvolution area is determined by the size of these borders.

Because the deconvolution process of a single subimage is independent from the other subimages, the subimages are processed in parallel on all available CPUs or threads to speed up processing. Additionally, all subimages are converted into contiguous one-dimensional $fftw\_complex$ data types to align the data and optimize cache access. To accelerate iteration over all subimages, the for loop iterates over an increasing integer used to index the subimages rather than directly looping over all subimage objects. Furthermore, wherever possible, functions use the $\_\_restrict\_\_$ macro to further optimize cache access.

For elementwise matrix operations, the compiler is provided with OpenMP hints to use single-instruction-multiple-data (SIMD) vectorization. Because of the contiguous one-dimensional data structure of the subimages, these operations can be easily vectorized by the compiler.

Due to the presence of many classical mathematical operations, such as elementwise matrix multiplication and division, it could be advantageous to compile the program with the -ffast-math flag to allow the compiler to optimize calculations.

3 EXPERIMENTS

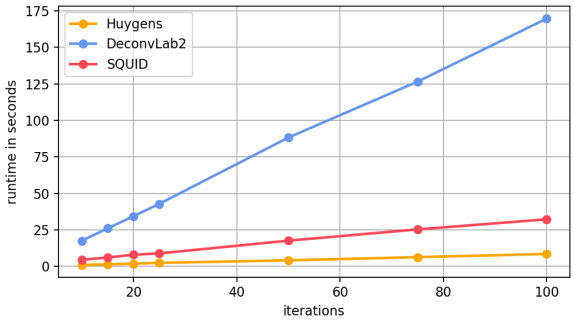
A synthetic PSF was generated with SQUID with the following configuration parameters. The subimage size was adjusted to the size of this generated PSF.

```
{
...
    "psf_path": "gauss",
    "savePsf": true,
    "psfx": 20,
    "psfy": 20,
    "psfz": 30,
    "sigmax": 5,
    "sigmay": 5,
    "sigmaz": 5,
...
}
```

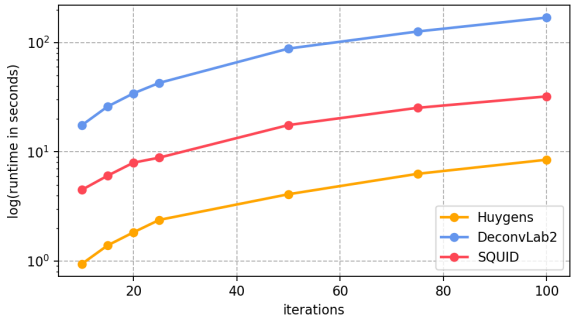
DeconvolutionLab2 (DeconvLab2) with 10 iterations the deconvolution process performed with Huygens took 0.94s, the deconvolution using DeconvLab2 took 17.50s and the deconvolution using SQUID took 4.48s. The amount of time required for deconvolution increased linearly with a growing number of iterations. At 100 iterations the deconvolution using Huygens took 9.47s, the deconvolution performed with DeconvLab2 169.70 and the deconvolution using SQUID 32.20s.

Table 1: Measured values for deconvolution. The table shows the number of iterations, the runtime in seconds and the Wasserstein distance calculated from the result images and the groundtruth image. All values of the runtime in this table are the minimum values with their standard deviation over five runs.

Tool	Iterations	Runtime in sec.	Wasserstein
Huygens	10	0.94 ±0.02	5.0785
DeconvLab2		17.50 ±1.37	6.4158
SQUID		4.48 ±0.14	4.8599
Huygens	15	1.39 ±0.05	5.0807
DeconvLab2		26.00 ±1.69	6.4158
SQUID		6.05 ±0.24	4.8598
Huygens	20	1.83 ±0.04	5.0812
DeconvLab2		34.30 ±0.28	6.4158
SQUID		7.96 ±0.65	4.8598
Huygens	25	2.38 ±0.21	5.0842
DeconvLab2		42.80 ±0.23	6.4158
SQUID		8.85 ±0.37	4.8597
Huygens	50	4.09 ±0.51	5.0860
DeconvLab2		88.20 ±1.22	6.4157
SQUID		17.59 ±0.56	4.8590
Huygens	75	6.29 ±0.64	5.0885
DeconvLab2		126.50 ±1.51	6.4157
SQUID		25.30 ±2.19	4.8585
Huygens	100	8.47 ±0.28	5.0884
DeconvLab2		169.70 ±36.88	6.4157
SQUID		32.20 ±1.08	4.8582



(a) Runtime in seconds vs the number of iterations.



(b) Logarithmic transformed runtime in seconds vs the number of iterations.

Figure 8: Comparison of runtime per iteration (Table 1). The green lines represent Huygens, the red lines DeconvLab2, and the blue lines SQUID. Figure 8b shows a logarithmic transformed runtime.

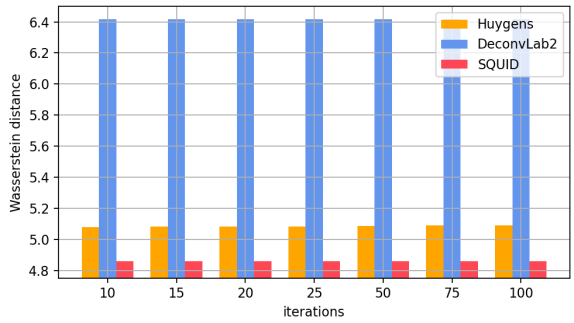
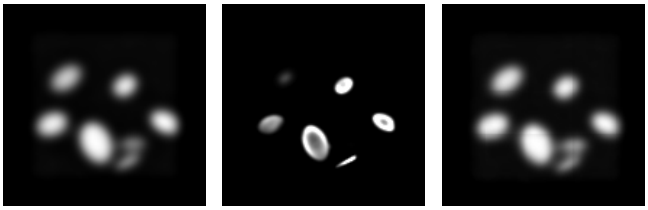


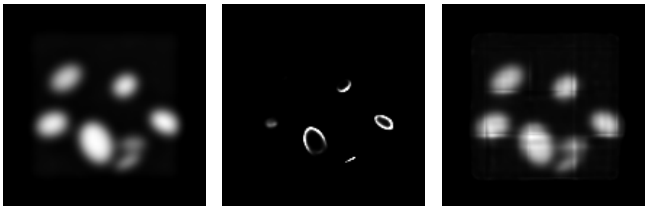
Figure 9: Wasserstein distance vs the number of iterations (see Table 1).

Comparing the resulting images of the three deconvolution tools revealed differences (Figures 9 to 11).



(a) Huygens **(b) DeconvLab2** **(c) SQUID**

Figure 10: Comparison of deconvolved images with 10 iterations. The subtitles of the panels show the tool used to deconvolve the input image (see Figure 12b).



(a) Huygens **(b) DeconvLab2** **(c) SQUID**

Figure 11: Comparison of deconvolved images with 100 iterations. The subtitles of the subfigures show the tool used to deconvolve the input image (see Figure 12b).

4 CONCLUSION

This project provides the groundwork for further implementations of our new deconvolution platform DeconvTool. The tool accepts TIF files, which are the most commonly available image format of microscopy data. The deconvolution algorithms are built upon a FT-based approach, which simplifies and accelerates the demanding calculations. In the next phase of the project, the execution speed will be increased via GPU acceleration, which will provide a major decrease in runtime. DeconvTool is currently a starting point for a fully open-source deconvolution platform, which is not yet an alternative to current commercial deconvolution tools like Huygens Pro, but provides a promising start. It offers a more flexible and scalable approach than existing open-source solutions like DeconvolutionLab2 and is approaching the performance of commercial tools. The platform also supports various possibilities for future extensions, making it a versatile solution for diverse deconvolution needs.

5 REFERENCES

[1] BioImage Analysis Lab at Institut Pasteur and France Bioimaging. 2024. *ICY*. https://icy.bioimageanalysis.org/ Accessed: 2024-09-13.

[2] Rafael C. Gonzalez and Richard E. Woods. 2008. *Digital Image Processing*. Pearson Education, Inc.

[3] ImageJ. 2024. *Fiji is just ImageJ*. https://imagej.net/software/fiji/downloads Accessed: 2024-09-13.

[4] Scientific Volume Imaging. 2024. *Huygens Software*. https://svi.nl/Huygens-Software Accessed: 2024-07-31.

[5] Scientific Volume Imaging. 2024. *Huygens Software GPU*. https://svi.nl/HuygensGPU Accessed: 2024-07-31.

[6] Leon B. Lucy. 1974. An iterative technique for the rectification of observed distributions. *Astronomical Journal* 79 (1974). https://doi.org/10.1086/111605

[7] Wilkinson M., Dumontier M., Aalbersberg I., and et al. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data* (2016). https://doi.org/10.1038/sdata.2016.18

[8] nti Audio. 2024. *Fast Fourier Transformation FFT - Basics*. https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft Accessed: 2024-09-13.

[9] Victor M. Panaretos and Yoav Zemel. 2019. Statistical Aspects of Wasserstein Distances. *Annual Review of Statistics and Its Application* 3 (2019).

[10] PlantUML. 2024. *PlantUML at a Glance*. https://plantuml.com/en/ Accessed: 2024-09-25.

[11] William Hadley Richardson. 1972. Bayesian-Based Iterative Method of Image Restoration. *Journal of the Optical Society of America* 62 (1972).

[12] Daniel Sage, Lauréne Donati, Ferréol Soulez, Denis Fortun, Guillaume Schmit, Arne Seitz, Romain Guiet, Cédric Vonesch, and Michael Unser. 2017. DeconvolutionLab2: An open-source software for deconvolution microscopy. *Methods* 115 (2017).

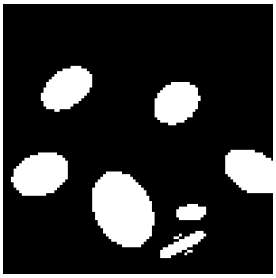
[13] SVI. 2024. *Huygens Compare Solutions*. https://svi.nl/Huygens-Compare-Solutions Accessed: 2024-07-31.

[14] Madri Thakur and Shilpa Datar. 2014. Image Restoration Based On Deconvolution by Richardson Lucy Algorithm. *International Journal of Engineering Trends and Technology* 14, 4 (2014).

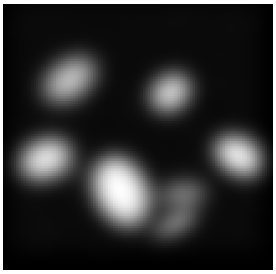
6 LIST OF ABBREVIATIONS

SQUID Super Quick Image Deconvolution
DeconvLab2 DeconvolutionLab2
EPFL École Polytechnique Fédérale de Lausanne
PSF point spread function
OoC Organ-on-Chip
FT Fourier transform
DFT discrete Fourier transform
IDFT inverse discrete Fourier transform
FFT Fast Fourier transform
GPU graphical processing unit

7 ATTACHMENT



(a) Original image



(b) Convolved image

Figure 12: Generated synthetic cells with DeconvTest. Figure 12a and Figure 12b show layer 25 (of 50) of the synthetic generated 3D cell images.

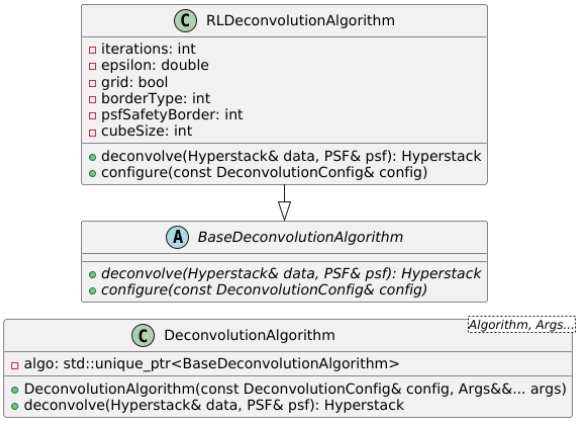


Figure 13: UML class diagramm of algorithm classes. The diagram illustrates three classes: `BaseDeconvolutionAlgorithm`, `RLDeconvolutionAlgorithm`, and `DeconvolutionAlgorithm`. The `RLDeconvolutionAlgorithm` class inherits from `BaseDeconvolutionAlgorithm`, indicating that it implements the abstract methods defined in the base class. The `DeconvolutionAlgorithm` class is a template class that takes two parameters, `Algorithm` and `Args...`, highlighting its generic nature. Arrows represent the relationships, showing that `DeconvolutionAlgorithm` uses an instance of `BaseDeconvolutionAlgorithm` to perform deconvolution operations. Visualized with [10]

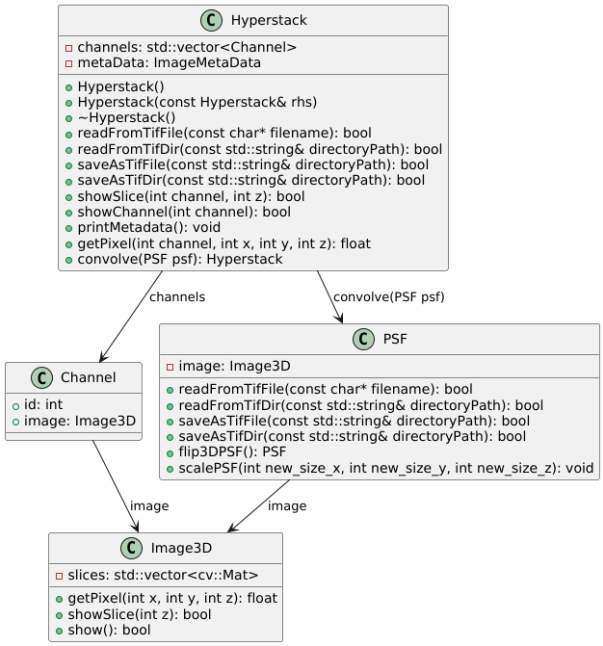


Figure 14: UML class diagram of image classes. The diagram shows four classes: `Hyperstack`, `Channel`, `Image3D`, and `PSF`. It illustrates how `Hyperstack` contains a list of `Channel` objects, and each `Channel` has an associated `Image3D`. The `PSF` class also contains an `Image3D`, and it's used by `Hyperstack` for convolution operations. Arrows represent the relationships, where `Hyperstack` interacts with both `Channel` and `PSF`. Visualized with [10]

```
1   Linux cm-framework 6.5.0-44-generic #44~22.04.1-
      Ubuntu SMP PREEMPT_DYNAMIC Tue Jun 18 ...
2
3
4   Architecture:                    x86_64
5   CPU operation mode:              32-bit, 64-bit
6   Address sizes:                   39 bits
      physical, 48 bits virtual
7   Byte Order:                      Little Endian
8   CPU(s):                          16
9   Online CPU list:                 0-15
10  Vendor ID:                       GenuineIntel
11  Model name:                      12th Gen Intel(
      R) Core(TM) i5-1240P
12  CPU family:                      6
13  Model:                           154
14  Thread(s) per core:              2
15  Core(s) per socket:              12
16  Socket(s):                       1
17  Stepping:                        3
18  Max CPU MHz:                     4400.0000
19  Min CPU MHz:                     400.0000
20  BogoMIPS:                        4224.00
21  Flags:                           fpu vme de pse
      tsc msr pae mce cx8 apic sep ...
22
```

Listing 1: System information

Library	Version	Open-Source
FFTW3	3.3.10	Yes
OpenCV	4.10.0	Yes
OpenMP	5.2.0	Yes
LibTIFF	4.5.1	Yes

Table 2: Versions and open-source status of the libraries used

Table 3: All measured values for deconvolution of Huygens. The table shows the number of iterations i, the runtime in seconds and the Wasserstein distance calculated from the result images and the input image (see Figure 12b) and the Standard Deviation sd of the values.

	i	1st run	2nd run	3rd run	4th run	5th run	sd
Runtime (sec)	10	0.96	0.96	0.99	0.98	0.94	0.0195
Wasserstein	10	5.0785	5.0785	5.0785	5.0785	5.0785	0.0
Runtime (sec)	15	1.41	1.45	1.48	1.39	1.50	0.0462
Wasserstein	15	5.0807	5.0807	5.0807	5.0807	5.0807	0.0
Runtime (sec)	20	1.92	1.83	1.86	1.89	1.91	0.0370
Wasserstein	20	5.0812	5.0812	5.0812	5.0812	5.0812	0.0
Runtime (sec)	25	2.38	2.86	2.49	2.48	2.30	0.2148
Wasserstein	25	5.0842	5.0842	5.0842	5.0842	5.0842	0.0
Runtime (sec)	50	4.09	5.31	4.12	4.16	4.44	0.5145
Wasserstein	50	5.0860	5.0860	5.0860	5.0860	5.0860	0.0
Runtime (sec)	75	6.84	7.95	6.76	6.50	6.29	0.6428
Wasserstein	75	5.0885	5.0885	5.0885	5.0885	5.0885	0.0
Runtime (sec)	100	8.52	8.97	9.09	8.95	8.47	0.2841
Wasserstein	100	5.0884	5.0884	5.0884	5.0884	5.0884	0.0

Table 4: All measured values for deconvolution of DeconvLab2. The table shows the number of iterations i, the runtime in seconds and the Wasserstein distance calculated from the result images and the input image (see Figure 12b) and the Standard Deviation sd of the values.

	i	1st run	2nd run	3rd run	4th run	5th run	sd
Runtime (sec)	10	20.8	18.2	17.5	17.8	17.7	1.3657
Wasserstein	10	14.7614	14.7614	14.7614	14.7614	14.7614	0.0
Runtime (sec)	15	29.9	26.2	26.1	26.0	26.2	1.6903
Wasserstein	15	17.4169	17.4169	17.4169	17.4169	17.4169	0.0
Runtime (sec)	20	34.9	34.8	34.3	34.7	34.3	0.2829
Wasserstein	20	19.1271	19.1271	19.1271	19.1271	19.1271	0.0
Runtime (sec)	25	43.4	42.8	43.1	43.0	42.9	0.2302
Wasserstein	25	20.3562	20.3562	20.3562	20.3562	20.3562	0.0
Runtime (sec)	50	88.2	85.6	85.4	85.4	85.5	1.2215
Wasserstein	50	23.7767	23.7767	23.7767	23.7767	23.7767	0.0
Runtime (sec)	75	130.2	126.7	127.4	128.3	126.5	1.5057
Wasserstein	75	25.5687	25.5687	25.5687	25.5687	25.5687	0.0
Runtime (sec)	100	173.1	181.5	171.9	255.9	169.7	36.8764
Wasserstein	100	26.7328	26.7328	26.7328	26.7328	26.7328	0.0

Table 5: All measured values for deconvolution of SQUID. The table shows the number of iterations i, the runtime in seconds and the Wasserstein distance calculated from the result images and the input image (see Figure 12b) and the Standard Deviation sd of the values.

	i	1st run	2nd run	3rd run	4th run	5th run	sd
Runtime (sec)	10	4.32	4.32	4.65	4.49	4.58	0.14
Wasserstein	10	4.8599	4.8599	4.8599	4.8599	4.8599	0.0
Runtime (sec)	15	5.77	6.31	5.82	6.09	6.24	0.24
Wasserstein	15	4.8598	4.8598	4.8598	4.8598	4.8598	0.0
Runtime (sec)	20	8.79	7.33	8.36	8.01	7.29	0.65
Wasserstein	20	4.8598	4.8598	4.8598	4.8598	4.8598	0.0
Runtime (sec)	25	8.44	9.27	8.63	8.71	9.12	0.37
Wasserstein	25	4.8597	4.8597	4.8597	4.8597	4.8597	0.0
Runtime (sec)	50	17.29	18.31	17.33	18.04	16.98	0.56
Wasserstein	50	4.8590	4.8590	4.8590	4.8590	4.8590	0.0
Runtime (sec)	75	23.13	27.14	25.54	27.74	23.03	2.19
Wasserstein	75	4.8585	4.8585	4.8585	4.8585	4.8585	0.0
Runtime (sec)	100	31.51	30.79	32.29	33.50	32.90	1.08
Wasserstein	100	4.8582	4.8582	4.8582	4.8582	4.8582	0.0