UNIVERSITY OF TWENTE

MASTER'S THESIS
BIOMEDICAL ENGINEERING

Wavefront Shaping At The Push Of A Button

Author:
Jeroen DOORNBOS

Supervisors:
Ir. Daniël COX
Ing. Tom KNOP
Prof. Dr. Ir. Ivo VELLEKOOP

External Member:
Prof Dr. Armağan KOÇER

Science & Technology Faculty
Biomedical Photonic Imaging

October 2, 2023

Abstract

Significance Scattering is one of the main limitations in achieving microscopic imaging deep inside tissue. Solving this would enable the research of biological phenomena on a larger scale. A technique to overcome the scattering of tissue is wavefront shaping. Wavefront shaping was conceived 16 years ago by Vellekoop and Mosk, and the field has developed significantly since then.

Aim This thesis works towards a practical application of these developed techniques. Wavefront shaping is a swiftly developing field in which user-friendliness, developer-friendliness, and the ability to share work through the scientific community are vital. The current implementation is not satisfactory in any of these requirements. The aim of this thesis is to create a developer-friendly system which can be used with the press of a button.

Approach A novel code-based platform, OpenWFS, was designed that allows for the development of wavefront shaping algorithms in Python. Additionally, this platform facilitates executing the code for wavefront shaping at the push of a button from a widely used open-source microscope controlling software called µManager. Using this software, a state-of-the-art wavefront shaping algorithm was expanded to simplify and generalise its use for the end user.

Results OpenWFS is able to integrate Python code into µManager microscope control software using a custom plugin that reads Python code and interfaces it with µManager. A large portion of the existing wavefront shaping code was restructured, clarified and adopted into this new platform. The µManager plugin was well received by the leading µManager developers and we are in the process of adopting it into the distribution of the software. The expansion upon the state-of-the-art wavefront shaping algorithm was done by considering that significantly contributing modes are spatially coherent, thus creating an algorithm that determines and refines its measurement protocol during measurement. It was determined that the wavefront shaping effectiveness can be successfully projected during the execution of the algorithm, allowing for further user-oriented development.

Conclusion OpenWFS successfully achieved the requirements of both users and developers, enabling the integration of Python code with the widely-used µManager microscope control software through a custom plugin. A substantial portion of the pre-existing wavefront shaping infrastructure was incorporated into OpenWFS, with the µManager plugin currently undergoing integration into the software's general distribution. The refinement of the state-of-the-art wavefront shaping algorithm has also been realised, demonstrating comparable or improved efficiency compared with the current golden standard, meaning a more user-friendly and robust implementation. The innovations presented serve as a foundation for further advancements in the field, facilitating more cooperative, accessible and versatile applications of wavefront shaping techniques.

Acknowledgements

The gratitude I feel towards everyone who aided me in this process is enormous. Science is done by the collective, and I have felt this very strongly throughout my project. In this project, I have worked especially close with my three supervisors whom I'd like to thank personally.

Beginning with my daily supervisors Daniël Cox and Tom Knop. I've very much enjoyed the cooperation and guidance you have given. You have found a gracious balance between supporting me and letting me grow, and your combination between digital and analogue smarts were perfect for this project. More than only on an academic level, you've made my stay at BPMI a joyous experience.

Ivo Vellekoop, who has not only supported me, but collaborated on OpenWFS and built the PyDevice almost entirely. I am very grateful for the opportunity being a part of the wavefront shaping team, and your skill to grasp the essence of and organise the many ideas we discussed were vital to the success of the project.

Armağan Koçer can also not be left out in this enumeration. Our talks have always left me more motivated and inspired, I was very happy that she accepted my invitation to be the external supervisor.

There are so many others I should thank; my family and friends, your support has made **all** the difference. I am honored to be on your side, and so glad you have been on mine.

Jeroen Doornbos, October 2023

Contents

Chapter 1

Introduction

'It is very easy to answer many of these fundamental biological questions; you just look at the thing!' - Richard Feynman

To see is to know. The incredible developments in the fields of biology and medicine were nigh impossible without the advancements made in technology that visualise the microscopic processes that underlie our biological machinery.
Scientists in the field of biomedical imaging work on the limitations of imaging technology to enable colleagues in biology. Dutch scientists have a long history in this field, e.g. Van Leeuwenhoek, Huyghens and Zernike, to name a few. The latter, Frits Zernike, was awarded the Nobel prize for his work on the phase contrast microscope, which is a solution for the limitation of the low contrast of cells, using the difference in refractive index of cells. This happens to intersect with my personal history, as my great-grandfather was his gardener.

On a larger scale, the refractive index mismatch of cells causes a different limitation: scattering. Scattering causes a loss in resolution as dispersed light cannot be distinguished from other light, causing a blurring of the image. The resolution is, fortunately, not irretrievably lost. In 2007, Vellekoop and Mosk showed that by using a Spatial Light Modulator (SLM) [1], light can be retrieved by sending in a tailored wavefront such that a focus is created despite scattering. This technique has since been coined 'Wavefront shaping'.

1.1 Wavefront shaping

In figure 1.1, we show the principle of wavefront shaping as presented in work from Vellekoop [1]. In 1.1a, a plane wave interacts with a strongly scattering material, resulting in a random speckle of diffused light. Although this seems like a random process, it is actually deterministic; with a static medium, the random speckle is reproducible. Due to the random scattering, some light reaches our desired focus location. As seen in figure 1.1b, by changing the wavefront of incoming light using a spatial light modulator (SLM), these random interactions can result in a constructively interfering focus. The challenge of wavefront shaping is finding the wavefront that does this.

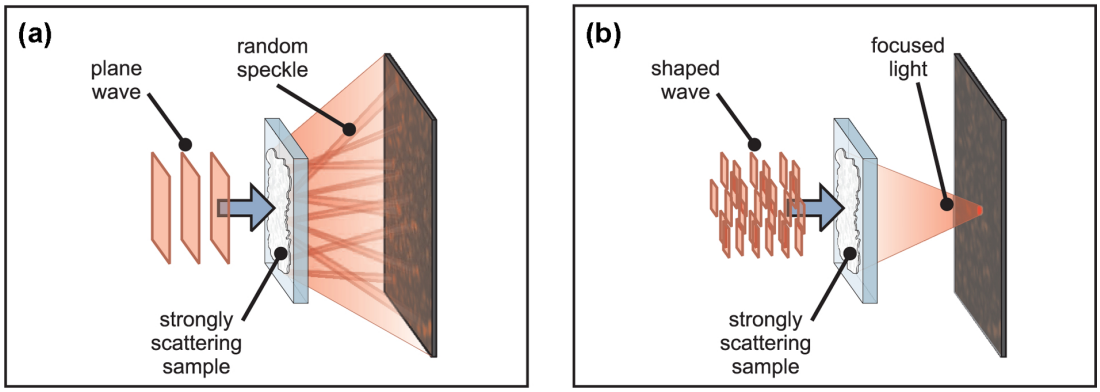


FIGURE 1.1: The principle of wavefront shaping from [1]. (a): The effect of a strongly scattering sample on a plane wave. Due to the scattering interactions, the light is spread out into a random speckle pattern. (b): The effect of a shaped wave, by shaping the wave in a particular manner, the scattering interactions result in a constructively interfering focus.

As generalised in [2], a wavefront can be optimised for any output mode, such as a focus, a plane wave or any other field pattern. Optimising a wavefront for a desired output is relatively simple once we know the transmission matrix of our scattering sample. In order to find this sample-specific transmission matrix, techniques and algorithms have been developed, which can be roughly separated into model-based, feedback-based and phase conjugation approaches.

The feedback-based approach works by measuring the elements of the transmission matrix individually. The most simple implementation of this is selecting a small region of the incoming light, changing the phase of this light, and measuring the response. Doing this for many regions allows you to build up the transmission matrix of the whole scattering object. This is called the '*stepwise sequential algorithm* (SSA)'. There are more techniques to do this, such as Hadamard- or Fourier-based methods [3], but both are still techniques to measure the transmission matrix of a scattering sample. Typically, this approach requires a significant number of measurements and therefore takes much time. Deciding how many measurements need to be done is a trade-off; measure too little, and the enhancement is low, measure too much, and the procedure takes long without much additional benefit.

The model-based approach treats the scattering sample not as a single transmission matrix, but characterises the optical behaviour of a sample by simulation [4]. Using wave propagation simulations, such as WaveSim [5], the propagation of light in a scattering sample can be estimated. Because scattering is a reversible phenomenon, one can model the propagation of light from the desired focus location and derive the required field of the incoming wave. Although it requires significant effort to build up such a simulation, using a model-based can be significantly faster than feedback-based methods.

The phase-conjugation approach involves measuring the field of the light coming out of the sample [6] [7]. The phase of this field is then sign-changed and used as the corrective illumination field. This is both fast and requires no prior knowledge, but a requirement is that the measured light all comes from the same spot, and we can only optimise the wavefront for that spot.

In this thesis, the groundbreaking work achieved in these efforts will be used towards making a platform and algorithms that are tailored for the use of the end user. The main

contribution is designing a robust open-source platform for wavefront shaping called 'OpenWFS' that enables both development and use. Additionally, the state-of-the-art algorithm developed by Mastiani and Vellekoop is developed further towards application. This was done using the setup specifically designed for deep-tissue microscopy: the two-photon wavefront shaping microscope.

1.2 Two-Photon microscopy

Two-photon microscopy is a laser-scanning fluorescence microscopy technique that uses non-linear optical effects to excite fluorophores at a specific focal spot selectively. This focal spot is then scanned through a sample, allowing the detection of the signal to be traced back to that one focal spot. Combining the scanned spots, an image is created.

Regular linear fluorescence requires the uptake of a photon with characteristic energy by the fluorophore. The excited fluorophore, after some internal relaxation, will then emit a Stokes-shifted photon. In multiple-photon microscopy, the same excited state is reached with multiple photons, which together have the characteristic excitation energy of the fluorophore [8]. This does, however, have a strong requirement that the photons arrive at the fluorophore at the 'same time'. This is the case when the arrival time is within 10^{-18} seconds. This is a statistical process, which depends on the laser power, as more incoming photons increase the chance of them arriving together. In conventional setups, this is achieved by using a pulsed laser source. The pulses of the laser are very short, packing the power of the laser into a very confined region. This prevents the use of a high-power continuous laser source, which would be problematic in many aspects.

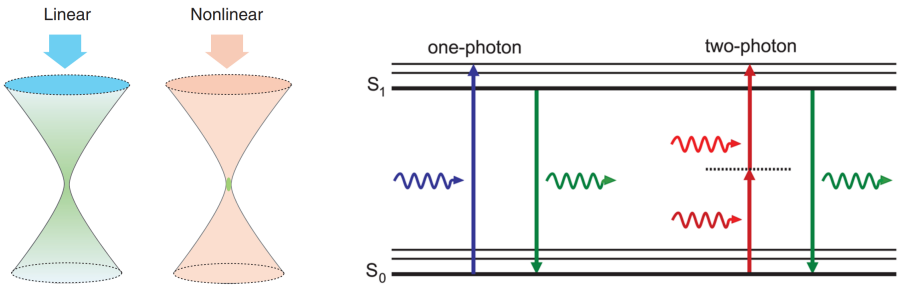


FIGURE 1.2: The difference between conventional and multiple photon excitation. Left: excitation by linear and non-linear methods [9]. Note that the green colour signifies the excitation area. Right: Jablonski diagram of one- and two-photon excitation and emission [8].

One of the key advantages of two-photon microscopy is its ability to image deep tissue structures [9]. This is in part due to the fact that longer wavelengths experience less scattering, allowing for more effective penetration into tissue [8]. In addition, due to the non-linear effect of two-photon excitation, only fluorophores in the focus are excited. This gives two-photon excitation much better spatial resolution compared to single-photon excitation.

Another advantage of two-photon microscopy is its ability to reduce photodamage to the sample being imaged. As can be seen in figure 1.2, traditional one-photon microscopy has a large area where photodamaging effects can occur. Due to two-photon excitation only

happening at the focus, photodamaging effects are only at the spot that is being measured. this allows for longer imaging times of biological samples.

Two-photon microscopy has been widely used in a variety of research areas [8], including neuroscience and cell biology. The ability to perform deep-tissue imaging is especially suitable for neural imaging, and this has been used to study the activity of neurons in the brains of mice [10] [11]. In cell biology, it has been used to study the dynamics of intracellular processes, such as vesicle trafficking [12] and cell division [13].

Expanding the use of two-photon microscopy with wavefront shaping has potential. As both techniques work towards deep-tissue imaging, the techniques can be used to enhance each other. Applying wavefront shaping for biology research, however, is not straightforward. The current state of the technology is not suited for end users, and that is a problem.

1.3 Problem statement

The current control software that is used for the two-photon microscope and wavefront shaping algorithm is complex to use, for both the end user and the algorithm developer. Especially for the end user, wavefront shaping is too complex to apply to their research without extensive aid from experts. Additionally, sharing work with scientists working on wavefront shaping outside the University of Twente is cumbersome, as they typically use different hardware.

As for the algorithms themselves, the feedback-based wavefront shaping approach is an effective method for wavefront shaping, but selecting which and how many measurements need to be done is complex. After a procedure, we can precisely tell which measurements were the most significant for the enhancement, but by then, much time has already been spent taking these measurements.

1.4 Project goals

The goal of the project is to develop wavefront shaping, from an experimental technology to a functional tool for biologists.

In the last 16 years, many developments have followed the initial results shown by Vellekoop and Mosk [1]. More efficient [3] and faster methods [4] [14] [7] for wavefront shaping have been developed.

This project can described by a number of system engineering goals and wavefront-shaping research questions. System engineering goals are:

1. Simplify the development of future wavefront shaping algorithms for the scientific community.
2. Simplify the use of currently existing wavefront shaping algorithms for the end user.

The design considerations will be discussed in Chapter 2. The product of this development is a code library written in Python, which we coined OpenWFS. It will be presented in Chapter 3. The last design requirement leads to our final chapter, Chapter 4. This is the research part of the project, where it is explored how the wavefront shaping algorithm can be expanded.

As mentioned in the problem statement, the use of the current feedback-based algorithm requires manual input for the number and type of measurements that are performed. In order to simplify its use, the following research questions emerge.

1. Can the number of input parameters of golden standard wavefront shaping algorithms be reduced?
2. Can we build a more efficient wavefront shaping algorithm?
3. How can we make the wavefront shaping algorithms robust for more sample types?
4. Can the potential of the wavefront shaping procedure be gauged before or during measurement?

Chapter 2

Design methodology

In this chapter, we analyse the requirements for the design of our new code for wavefront shaping. We do this by defining requirements and additionally useful features by performing a stakeholder analysis. A vital decision for this design is which microscopy control software is used, as this determines both the end-user experience and the ease of development. This decision was made by ranking and weighing the essential aspects of the available options.

Before commencing with the implementation, a developmental plan was conceived. This is shown in figure 2.1. Following that plan, Requirements were selected and rated with the current users, as shown in Section 2.1. Then, we investigated the potential microscope control software packages; these are described in Section 2.2. This chapter ends with the final selection of the platform, which is the final step in planning the development as described by figure 2.1.

In chapter 2.2 two viable choices were considered. From a planning perspective, this could be condensed to either improving the existing infrastructure or to replace the current one and build new functionality. Improving existing infrastructure would most likely mean that there is more time for individual features, and would result in a circular development process. Replacing the entire infrastructure on the other hand, is more likely to result in a linear development process.

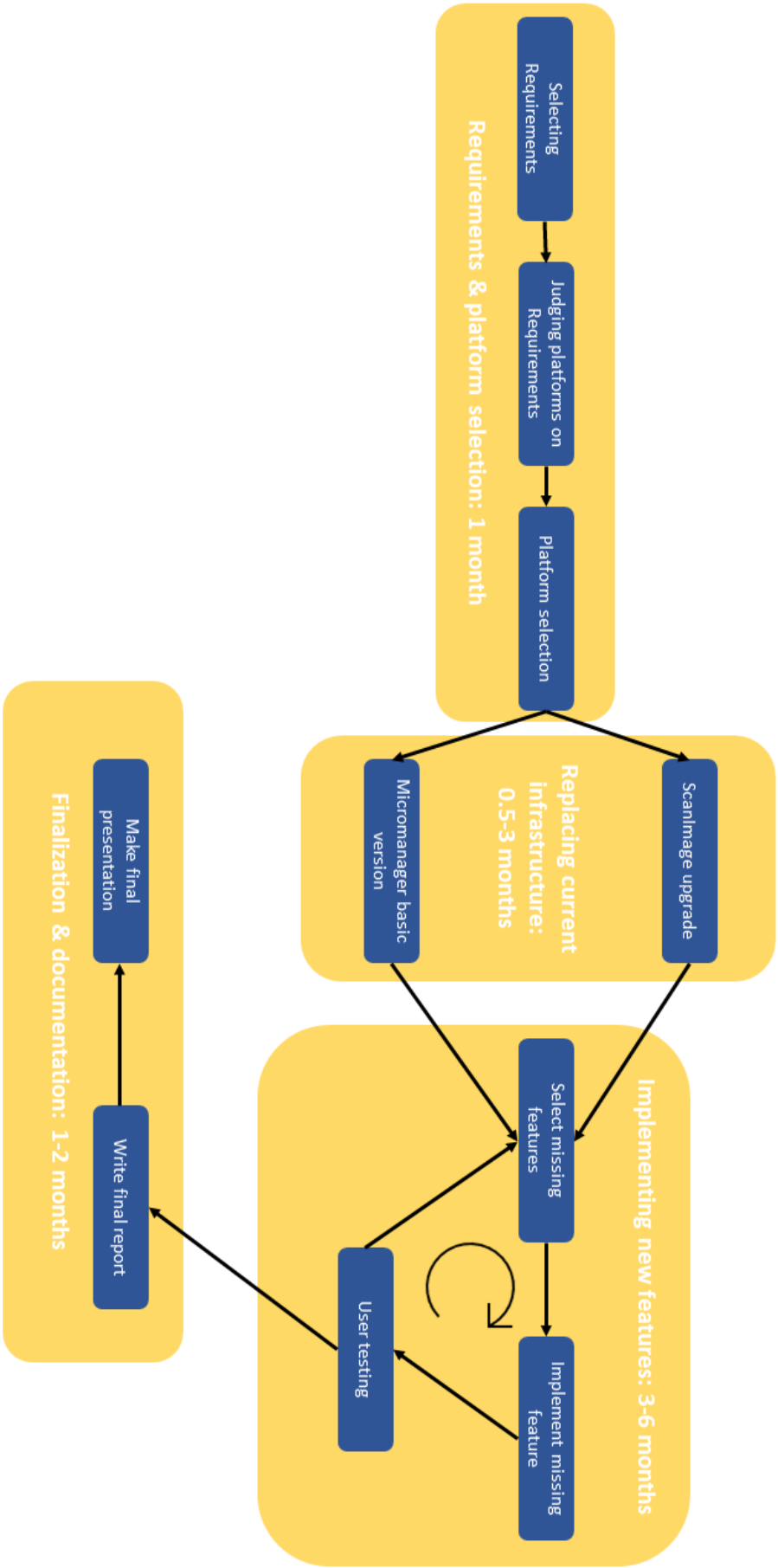


FIGURE 2.1: A outline of the dependencies & development in this project.

2.1 Requirements

As stated in Chapter 1, the goals for the wavefront shaping software are:

1. Simplify the development of future wavefront shaping algorithms for the scientific community.
2. Simplify the use of currently existing wavefront shaping algorithms for the end user.

In order to meet these goals, requirements were set up. For orientation, the current users of the system were asked to rate and add to a list of potential requirements. This is shown in Appendix B. However, the current users are all wavefront shaping developing physicists and, therefore, not representative of our full target audience. Even though we cannot directly select the platform based on the requirements of the current users, this allowed us to create distinct requirements that are important for the achievement of our goals:

1. The system needs to be easy to implement.
2. The system needs to be easy to use.
3. The system needs to allow easy wavefront shaping algorithm development.
4. The system needs to be open-source.
5. The programming language needs to be accessible and widely used.

We will use these requirements to make our final choice. Before that, however, we need to know what we can choose from.

2.2 Available software

In order to make an informed decision, we considered the characteristics of the available software packages. The software packages considered are microscope control software packages. Because a end-user requires an interface to interact with, these software packages all have an graphical user interface (GUI). Information herein was primarily gathered from users in the field, mostly in the form of correspondence with these experts and users.

2.2.1 Current implementation: ScanImage ®

ScanImage is software for laser scanning microscopes with a GUI. It is a MATLAB-based program that allows a developer to control a custom build laser-scanning microscope. It was developed on an NIH grant, which allowed the project to be open-source [15]. After the lapse of this grant in 2019, the project moved to a licensed and commercialized project supported by MBF Bioscience [16]. There are efforts to secure another grant, which would allow the project to return to an open-source state, but there is no guarantee that this will be achieved.

The software currently installed in the 2-photon setup is the open-source unsupported version of ScanImage. Its GUI is editable using the Matlab GUIDE toolbox. This toolbox will be discontinued in future Matlab versions, which would make future customization and development unsustainable.

The licensed ScanImage has two versions: a basic version and a premium version. In both cases, the license will be provided by MBF Bioscience. This licence allows for source access to the ScanImage codes. Especially with the help of ScanImage support, changing internal scanner properties should be possible.

Advantages ScanImage is currently implemented already, which allows for much time for development in improving the user experience. ScanImage is specifically built for laser-scanning microscopes, so it has many features that aid with its control.

Disadvantages The support of the open-source distribution of ScanImage is very limited, and the complex MATLAB code makes further development hard. As the tool for expanding its GUI will be unsupported by the MATLAB developers, building and maintaining a wavefront shaping GUI will not be easy.

2.2.2 SciScan

Another commonly used laser-scanning software package is SciScan [17]. This is an open-source software package based in LabView. Though developed by Scientifica, it can also be used for non-Scientifica-based applications. LabView is not necessarily a convenient programming language to work in for several reasons; therefore, many of the potential users strongly oppose its adoption. Moreover, the site of Scientifica no longer provides any direct links to SciScan but provides links to ScanImage instead [18].

2.2.3 µManager

µManager is a widely used and adapted open-source platform for microscope control. It has been installed on 10.000+ systems [19] and has built-in support for many hardware devices, including commercial ones. Its robust implementation yet extensive customizability

lets it surpass any commercially available software. Pycromanager is the Python package that interfaces with µManager.

However, µManager is missing support for laser-scanning devices, a requirement for 2-photon imaging. Therefore the only labs that have implemented µManager for their 2-photon setups have highly specific and custom-built packages interfacing and controlling the laser-scanning setup with µManager.

Because µManager is not built for laser-scanning microscopes, an addition must be added to the package to create a functional platform. This adds a device adapter that performs the scanning and behaves as if it was a camera, thus providing images to µManager. A open-source general-use software for this exists: OpenScan. However, it still being developed, and the software's author, Mark Tsuchida, has specified that if this option was going to be used in our implementation, at least to some degree we will have to do some low-level programming, enough to be considered contributors to the project. Other groups, like the Krummel Lab from the University of California were contacted, but also used custom software tailored for their specific hardware, thus not useful for our project.

Advantages µManager is software designed for easy use by developers and end-users. The API is very helpful, and support by the developers and community is great. The GUI is commonly used for many different microscope setups.

Disadvantages µManager has no support for laser-scanning software. Extending the software to support laser-scanning software would require extensive effort. Additionally, micromanager is written in C++, so we either need to bridge this software with another language or write all our wavefront shaping algorithms in C++.

2.3 Final selection

If we ignore the project-planning requirements, we see that µManager would provide a much better system. Considering that the support and longevity of µManager are projected to go up, whereas the support and longevity of ScanImage are expected to go down, choosing µManager is a more future-proof solution. Because the table in Appendix I fails to weigh these considerations, we present a more general matrix in Table 2.1.

Ease of implementation:
ScanImage is already implemented. Therefore little work is necessary to commence further development. On the other hand, µManager will require extensive effort to implement. Other groups have working versions for their specific hardware, but no general implementation for laser scanning microscopy are available. This means much low-level development has to be done. This does provide the opportunity to shape the platform exactly to our needs, but might significantly decrease time for expanding on the current functionality of the wavefront shaping algorithms.

Ease of use:
The ease of use is very dependent on the degree of development of the platform and on the users themselves. Nevertheless, having briefly used both systems, the authors consider neither superior in ease of use. µManager is more intuitive and visually more pleasing, but ScanImage is more extensively developed. Furthermore, the structure ScanImage uses

with the SI handle is suited for expansion and relatively easy to work with. All things considered, µManager gets 7 out of 10 points. ScanImage currently has more features than µManager but as it is less supported, it gets a 7 out 10.

Ease of development:
In order to judge the difference between the ease of developing a feature in both ScanImage and µManager, a certain feature was implemented in both packages: a GUI to control the gain. Because µManager does not have a working laser-scanning setup, as mentioned before, we implemented this feature in µManager using a conventional camera. Both packages have a laser-power plugin controlled through a NI data acquisition card. The resulting interfaces are shown in figure 2.2.

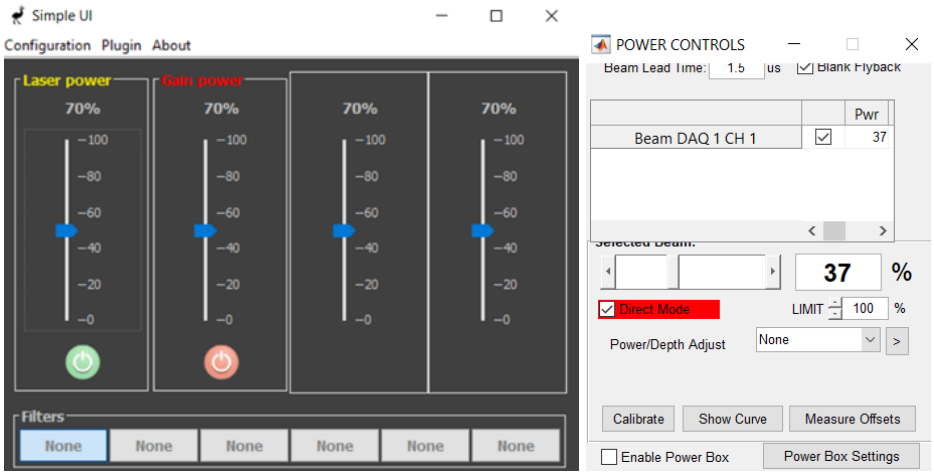


FIGURE 2.2: The implemented functional GUIs for both µManager (left) and ScanImage (right)

Both were connected with NI boards, and functionality was checked with oscilloscopes. The ease of implementation for µManager was better, as more online information was available, and the platform of µManager was deemed more convenient. That is not to say that the ease of implementation in ScanImage was complex, nevertheless. µManager & ScanImage both immediately put the values of the new GUI into the metadata. Because of the available information, µManager was deemed more capable for development than ScanImage. Note that this is not a fully independent judgement, as ScanImage was implemented first. The knowledge gained implementing a gain feature in ScanImage aided the development of the feature in µManager.

Developing features and functionalities is dependent on the available information and the robustness of the platform. µManager was specifically build to be developer friendly, and supports a plethora of microscope setups. The API is actively curated and community support is high. ScanImage also has an API, but the support of the open-source versions is non-existent, and community support is meagre.

Open source:
Both options are open source. The open source version of ScanImage, however, is no longer supported by the original creators. Because the open source version of ScanImage has to potential to become obsolete because of this reason, it does get 6 out of 10 points, as at some point, moving to the licensed option might be required.

Programming language:
Both MATLAB and Python are commonly used by scientific users. However, most users are more familiar with MatLab, but potential developers usually use Python. When asking the relevant stakeholders, and considering that the staff of the University of Twente is starting to replace its MATLAB programs with Python, we give Python 8 points and Matlab 7.

We ranked the properties on importance in accordance to users and assigned weights of 10, 15, 20, 25 and 30, respectively. Filling this into the WDM scheme results in Table 2.1.

TABLE 2.1: Weighted aspects of the platforms ScanImage and µManager.

Property	**Weight**	**ScanImage**	**µManager**
Ease of implementation	15	10	4
Ease of use	25	7	7
Ease of development	30	6	8
Open source	20	6	10
Programming language	10	7	8
Weighed average:		6.95	7.55

This results in a choice for µManager. The main differences are in the ease of implementation, ease of development and open-source categories. In short, µManager will be harder to implement but more worthwhile to implement. As mentioned, µManager is also the more future-proof choice, with constantly improving open-source developments on the platform. It is of key importance to get the structure of the integration with µManager to utilise most of the platform.

Chapter 3

OpenWFS: Open-source software for Python-based wavefront shaping

In this Chapter, we present the design of OpenWFS. We wrote this chapter in the form of a whitepaper that will be submitted. It has been written to concisely show and clarify the capabilities of the platform. After interaction with the scientific field, the field has reacted enthusiastically to our efforts, and the PyDevice µManager connection will most likely be integrated into the main version of µManager. In this thesis, it will also serve the purpose of clarifying and demonstrating the functionalities of OpenWFS. The co-authors of this paper are all people who worked in the development of the MATLAB code.

OpenWFS: A Python package for wavefront shaping and seamless integration in µManager microscopy software.

Jeroen Doornbos, Bahareh Mastiani, Gerwin Osnabrugge, Tom Knop, Daniël Cox, Harish Sasikumar, Michele Gintoli, Siebe Meijer, Giulia Sereni, Tzu-Lun Ohn, Ivo Vellekoop
University of Twente

Light scattering is one of the main limitations of imaging depth in microscopic imaging. Wavefront shaping is a technique that allows for a recovery of resolution despite this scattering. However, there is a need for a common platform to use and develop consistently, clearly and cooperatively. To that purpose, we present an open-source framework in Python called OpenWFS. It contains code controlling a wavefront shaping microscope using a MicroManager device that will run a Python script as a device for straightforward microscopy use. This paper will clarify and promote the use of the OpenWFS package.

Introduction

In optical microscopy, the main limitations of imaging depth are aberrations and the light scattering of samples [1]. Possible techniques developed for deep-tissue imaging despite or avoiding scattering are thoroughly investigated. Examples are the separation of ballistic and non-ballistic light [2], or chemical clearing techniques of the sample [3]. However, it was shown that this light can still reach the target destination despite scattering [4]. This has since been coined 'wavefront shaping'.

In wavefront shaping, As in adaptive optics (AO), a spatial light modulator (SLM) is used to spatially shape the phase of the light in the objective plane to achieve a better focus in the focus plane. In adaptive optics, the correction patterns are assumed to be smooth and continuous, while wavefront shaping has been shown to converge in strong aberrating and non-continuous cases [4].

Typical feedback-based wavefront shaping is performed by decomposing the light field into orthogonal modes [2], which can be optimized individually, setting that mode to the phase that enhances the image most, before combining into a single correction pattern.

The development of wavefront shaping significantly benefits from the contribution of many different research groups around the globe. Much progress has been made towards the methodology of wavefront shaping, such as a pixel-by-pixel, Hadamard pattern, or Fourier-based approaches [5], Fast techniques that enable wavefront shaping in diffusive samples [6] [7], and many potential applications have been developed and prototyped, including endoscopy, optical trapping [8] and deep-tissue imaging [9].

Because the techniques have reached a state of maturity over the decades of development, the focus will shift from a technique-developing effort to a technique-application effort such that a wide user base can make use of this emerging technique. This requires high-level control that is not typically built into experimental code. Therefore, we have integrated this platform with µManager. µmanager is ImageJ-based microscope control software that makes use of very powerful Java and C++ libraries. The recent additions in micromanager allow Python control of micromanager functions [10], but the reverse, micromanager control of Python functions, is still lacking. Here, we introduce a generic µManager device that integrates Python-based hardware control with µManager, allowing for a 'plug-and-play' style of user control of developmental wavefront shaping algorithms.

The development and use of wavefront shaping algorithms are done by vastly different fields. It can be said that there is a gap between optics-orientated physicists and microbiology-oriented biologists. The goal of OpenWFS is to close that gap, by allowing the users to interface with the algorithms through a familiar user interface, while not creating any additional overhead for the developers of algorithms. Our goals for this platform are therefore stated as such:

1. Easy-to-use Python-based platform for simulation and use of wavefront shaping and adaptive optics algorithms.
2. Modular code structure to maximise developer efficiency and enable easy cooperation between research groups.
3. A seamless zero-overhead integration with the µManager microscope control software.

Together, this works towards the goal of a 'push of a button' wavefront shaping system.

Wavefront shaping using Python

As the field of wavefront shaping has expanded, there is no longer one workflow or method for wavefront shaping. However, common requirements are the control of an SLM, detectors, and auxiliary hardware such as translation stages. Control of these hardware components typically requires detailed technical knowledge and low-level programming. Additionally, other setups may use different types of hardware.

Therefore, code for wavefront shaping tends to be complex and setup-specific. The goal of OpenWFS is to provide a library of re-usable components for hardware control and Wavefront shaping algorithms. This makes development easy, allowing developers to swap out and add components at will.

In figure 1 we break down a wavefront shaping procedure in simple steps. An algorithm determines a wavefront and projects it on the SLM. Then, a measurement is done in the setup, resulting in a measurement by the detector. This feedback is then used as an input to the algorithm. This loop may be traversed iteratively, or once, depending on the algorithm.

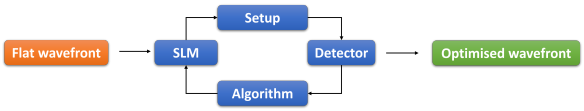


Figure 1: Diagram of a wavefront shaping procedure. The algorithm projects a pattern to the SLM, a measurement is done in the setup, resulting in a feedback measurement in the detector.

In OpenWFS, templates for the SLM, detector and algorithm were made. Furthermore, templates for common hardware components were added, such as the 1-dimensional stage and the XY stage. We will discuss them here individually, and then show how a simple wavefront shaping algorithm can be implemented in OpenWFS using these components. Where possible, the templates implement enforced units from the *astropy.units* package [11]. All these components have been made µManager compatible, which will be shown in the next chapter.

Spatial light modulator

The spatial light modulator (SLM) is the hardware responsible for shaping the wavefront. We have implemented support for a phase-only SLM, as shown in Appendix C, which is an interface containing methods for setting and updating the pattern shown on the SLM.

As one of the implementations of the general interface, we have implemented an SLM object using OpenGL, that controls the SLM connected to the display port of the PC. It has additional functionality to check and regulate spatial transformations, look-up tables & synchronisation. The spatial transformations are a texture mapping functionality, allowing the user to send a pattern to an SLM in a matrix, which is then mapped onto a specific geometry. This is useful, for example in [5], where a spherical geometry is used.

Detector

The detector can be any object that provides feedback for the wavefront shaping algorithm. It can be implemented as a camera, a laser-scanning microscope [12], photacoustic feedback [13], or guide-star-based feedback [14]. Algorithms for wavefront shaping should be generalisable for many types of detectors.

Controller

The controller is an implementation of an object that contains both a detector and an SLM. It solves communication and timing issues between the SLM and detector, as, at high speeds, synchronising measurements with the refresh rate of an SLM is not trivial. A single call to the 'measure' method in the controller object corresponds to updating the SLM, waiting for the image to stabilize, triggering the detector and reading the data from the detector. This way, the algorithm developer does not need to worry about these specifics.

This is particularly useful for synchronisation functionality that facilitates advanced timing. An SLM typically has an idle time between receiving and physically performing that instruction. For optimising SLM performance, this can be used by updating the SLM pattern before the previous measurement has been finished.

Additional components

With the basic necessities of a wavefront shaping setup covered, we typically require additional hardware control to make a useful microscope. We have added a template for a one-dimensional and XY stage, shown in Appendix C, both are µManager compatible. Any other device is also compatible with µManager, as shown in the next chapter.

Simulations

Currently, almost the entire setup has a mock software counterpart. We can simulate an experiment by creating a simulation object that contains the required functionality of both an SLM and a detector, allowing for the testing and development of wavefront shaping algorithms.

Algorithm

Using these interfaces, building an algorithm for wavefront shaping is straightforward. Below, we present code

that performs a wavefront shaping procedure called the stepwise sequential algorithm (SSA) [15].

As can be seen in the code, the SLM is divided into smaller elements, which are optimised individually using phase-stepping.

```
1   class StepwiseSequential:
2
3       def __init__(self, phase_steps, n_x, n_y, controller):
4           self._n_x = n_x
5           self._n_y = n_y
6           self._controller = controller
7           self._phase_steps = phase_steps
8
9       def execute(self):
10          self.controller.slm.phases = np.zeros(
11              (self.n_x, self.n_y), dtype="float32")
12          # reserve space to hold the measurements
13          self.controller.reserve((self.n_x, self.n_y,
14              self.phase_steps))
15
16
17          phases = np.arange(self.phase_steps) /
18              self.phase_steps * 2 * math.pi
19
20          for n in range(self.n_x * self.n_y):
21              for p in phases:
22                  self.controller.slm.phases.flat[n] = p
23                  self.controller.measure()
24
25              self.controller.slm.phases.flat[n] = 0
26
27          t = np.conj(self.controller.compute_transmission(
28              self.phase_steps))
29          return t
```

As seen in this code, the algorithm changes the incoming field by shifting the phase of a segment of the SLM. In lines 4 and 5, the segments are defined by n_x and n_y segments in the x and y direction. In line 10, an array is reserved to enter the measurements. In SSA, we measure the response of each segment (line 20) by projecting different phases (line 21) while keeping the other segments constant. The number of phases is defined in line 7, and the phase-stepping is done in lines 17-25. Because we are showing different phases on the segment, we expect the measurements to have the shape of a sine wave. A certain phase causes the maximum constructive interference and a π phase step from that, we expect maximum destructive interference. The amplitude of this sine wave is how much that segment contributes to the overall feedback.

In line 27, the transmission matrix elements of each segment are determined by a computation, which is essentially a discrete Fourier transform, as shown in Appendix C. The phase of the field can be used to set the SLM to the optimal wavefront.

In conclusion, wavefront shaping requires control over an SLM, a detector and an algorithm. A wavefront shaping environment additionally requires additional hardware, mock hardware and simulations. We present a framework in which these have been implemented in a general and flexible manner, which allows the user to develop and perform wavefront shaping experiments with full control in Python.

Controlling OpenWFS from µManager

OpenWFS allows for a consistent, testable and flexible framework to develop and experiment with wavefront shaping. However, as the techniques improve and become more capable, there will be a stronger focus on *applying* wavefront shaping rather than *developing* this technique. Therefore, we require control over wavefront shaping algorithms from a microscope environment that can be used by non-experts.

For this, we have chosen the widely used open-source microscopy control software µManager. We have integrated the OpenWFS functionality with µManager with the use of a µManager plugin we call 'Pydevice'. As shown in figure 2, this plugin allows the end user to select pre-existing Python code and automatically integrate it in the µManager environment. Performing a wavefront shaping experiment in µManager is straightforward and will store the parameters of the experiment automatically in the µManager metadata structures. This unlocks the technique of wavefront shaping for a whole new group of potential users, as a minimal amount of programming knowledge is necessary.

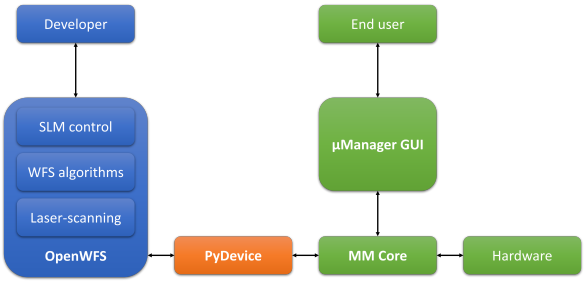


Figure 2: Blue: This paper introduces the Python-based wavefront shaping package OpenWFS, for the development of wavefront shaping algorithms. Green: The pre-existing µManager microscope control workflow. A GUI interfaces the control layer MMcore, which communicates with the individual hardware elements. Orange: The Pydevice enables µManager control of OpenWFS functionality.

PyDevice: wavefront shaping for everyone

The Pydevice plugin allows a user to import their own object chart into µManager. The user selects the file that contains the required objects and imports it in

µManager. For example, a test camera was created to create and fill the image buffer with random noise. The code for this is shown in Appendix C. Incorporating it in µManager is simple, a Python file containing the following is selected:

```
from test import Camera,RandomGenerator

r = RandomGenerator()
devices = {'cam': Camera(random_generator=r), 'rng': r}
```

Which is recognised by PyDevice with the keyword 'devices' and loaded into µManager as devices, shown in figure 3.

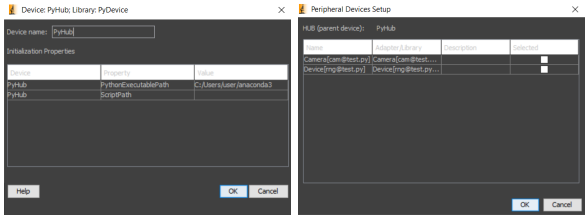


Figure 3: The configuration of a PyDevice is as straightforward as the configuration of any other µManager device.

The two classes can be loaded into µManager with the use of the PyDevice as shown in figure 3, by providing the script path or selecting a file with the file browser. The Python interpreter path can be specified, as well as a virtual environment path. As can be seen in Figure 4, the camera object has a property that is the RandomGenerator object, which is also added as an object in µManager. This allows communication between Python objects, which is necessary for more complex functionalities.

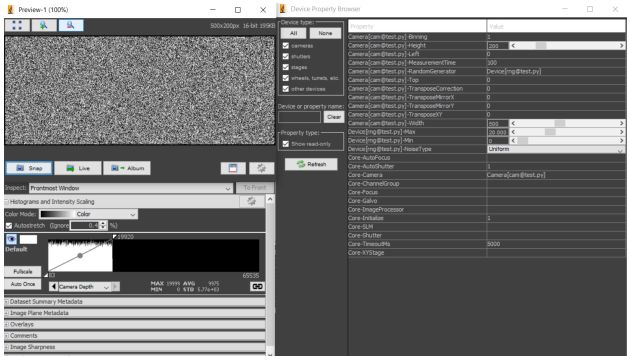


Figure 4: A test camera returning noise in the µManager environment.

How the bridge works

Typically, µManager devices have to be written in C++. This makes sense, as the devices are typically used to control hardware, and C++ is well suited for that task. For our purposes, however, we require control over the parameters and execution of Python objects. Therefore, Pydevice is a C++ plugin that runs a Python interpreter. In order to make the Python variables accessible to µManager, it translates and reads the variables from Python to C++.

This translation is done by PyDevice by scanning the properties of the Python objects. One of the essential translations is from the dynamic typing of Python towards the static typing of C++. In OpenWFS, object properties are given type hints to solve this. In Python, these type hints are not enforced, but Pydevice enforces these hints and converts the properties to the corresponding µManager property.

The Pydevice plugin automatically detects if a Python object implements the methods of a µManager Camera, Stage, or StageXY device, and exposes the Python object in µManager as such. The templates for these objects are seen in Appendix C. This allows the µManager interface to register a camera device in Python as a camera, which unlocks all plugins and interfaces that require such an object.

Developing new components

Pydevice allows the developer to prototype a device in Python with relative ease and import it to µManager with minimal modifications. We will show how this is done by showing a simple implementation of a Camera device, which was loaded into µManager in figure 3 and 4.

Camera device

Using the @property and @name.setter decorators, the properties become available in µManager. As explained, additional methods and properties are required for the Pydevice to recognise it as a specific device class, such as a camera. The templates in Appendix C contain all required properties and methods, so all code that follows the OpenWFS templates is compatible.

The properties *top, left, width, height* and *exposure_ms* are required by the µManager API for a camera. Additionally, the methods *trigger* and *read* are required. *get_image* is responsible for creating the buffer, and *wait* is responsible for filling it. This allows µManager to access this buffer. This allows the developer to make any Python class capable of returning a Numpy array behave like a camera in µManager. For clarity, we show the implementation of a demo camera in its entirety:

```python
class Camera:
    """
    Demo camera implementation that returns noise images.
    """

    def __init__(self, left=0, top=0, width=100, height=100,
        random_generator=None,
        measurement_time: Quantity[u.ms]=100 * u.ms):
        if random_generator is None:
            random_generator = RandomGenerator()
        self._resized = True
        self._image = None
        self._left = left
        self._top = top
        self._width = width
        self._height = height
        self._measurement_time = measurement_time.to(u.ms)
        self._random_generator = random_generator

    def trigger(self):
        if self._resized:
            self._image = np.zeros(self.data_shape,
                                   dtype=np.uint16)
            self._resized = False
        self.random_generator.generate_into(self._image)

    def read(self):
        return self._image

    @property
    def data_shape(self):
        return self._height, self._width

    @property
    def left(self) -> int:
        return self._top

    @left.setter
    def left(self, value: int):
        self._top = value
    @property
    def top(self) -> int:
        return self._top

    @top.setter
    def top(self, value: int):
        self._top = value

    @property
    def width(self) -> Annotated[int, {'min': 1,
                                       'max': 1200}]:
        return self._width

    @width.setter
    def width(self, value: int):
        self._width = value
        self._resized = True

    @property
    def height(self) -> Annotated[int, {'min': 1,
                                        'max': 960}]:
        return self._height

    @height.setter
    def height(self, value: int):
        self._height = value
        self._resized = True

    @property
    def measurement_time(self) -> Quantity[u.ms]:
        return self._measurement_time

    @measurement_time.setter
    def measurement_time(self, value):
        self._measurement_time = value.to(u.ms)

    @property
    def random_generator(self) -> object:
        return self._random_generator

    @random_generator.setter
    def random_generator(self, value):
        self._random_generator = value
```

As can be seen, this Camera implementation follows the template for the camera, shown in Appendix C. As seen in line 35, the required property *left* has the type hint *int*. PyDevice will convert this property in the corresponding µManager property. Optionally, a range can be defined, such as in line 50. As seen in line 70, the measurement time requires a unit. As the unit is internally set to µs, the value will always be in the unit µManager expects.

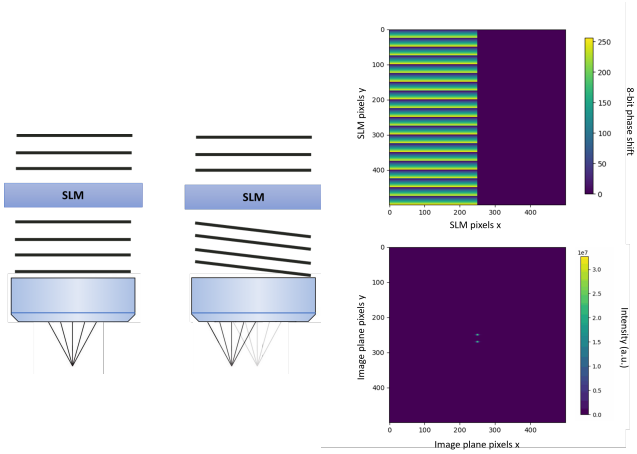
Simulations

With the device creation process clear, the next step is to simulate our experiments, ensuring that our algorithms are valid and efficient before any real-world application. We will now discuss these simulations, show their results and describe how this enables the development of wavefront shaping algorithms.

For the demonstration, we chose to simulate the experiment in the most simple manner possible: we compute the focus as the Fourier transform of the wavefront before the objective. All simulated aberrations are simulated as phase offsets in the pre-objective plane. This allows the user to 'set' a correct wavefront that can be solved by the wavefront shaping algorithms. This is an acceptable simulation, as the fact that an aberration in the image plane can be represented as phase shifts in the objective plane is the basis of all wavefront shaping. This working principle was then built into a device that both meets the requirements of an SLM and a camera, allowing a wavefront shaping algorithm to use the simulation device as both.

The simplest example of this is shown in Figure 5, where the simulation shows two foci, one shifted from the centre due to an angled wavefront. Note that due to the wrapping of a phase, 2π and 0 are identical from a wavefront shaping perspective. Because these phase values cyclically wrap around after reaching 2π, we can effectively portray a wavefront angle of 40π on the SLM by portraying 20 phase wraps.

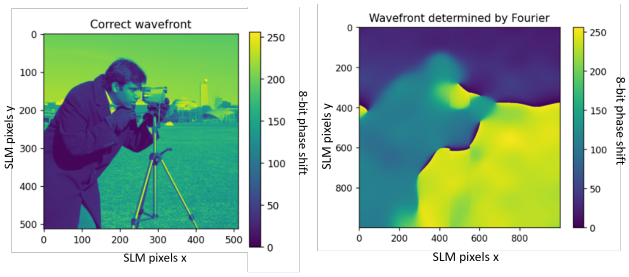
Additionally, similar to how we simulate the wavefront by altering the phases of the objective plane, we can simulate SLM illumination by altering the amplitudes of the objective plane by illuminating the SLM with a Gaussian distribution, which results in a Gaussian distribution in the image plane.



(a) (b)

Figure 5: Simple example of a wavefront shaping simulation. a): an illustration of how an angle in the SLM plane results in a location in the image plane. b): Results of a simulation where half of the SLM plane creates an angular wavefront, resulting in 2 foci in the image plane.

This is an idealized version of the physical situation during wavefront shaping. Therefore, any feedback-based algorithm should work here as well. To test the OpenWFS framework, we perform a Fourier-based wavefront experiment, as developed and tested in [5].

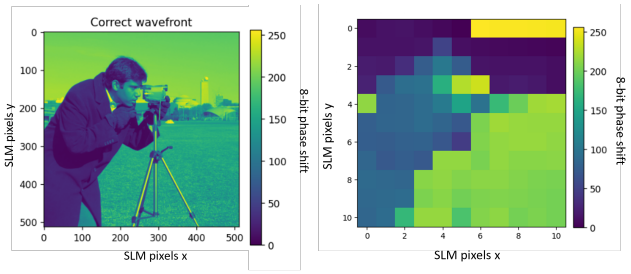


(a) (b)

Figure 6: a): The image set as an 'aberration' to the simulation objective plane. b) The resulting wavefront produced by a Fourier-based wavefront shaping algorithm with 121 modes.

Figure 6 demonstrates the effect of using a limited number of Fourier modes on the computed wavefront. Since only low-frequency modes were measured, only the low-frequency aspects of the correct wavefront were found during the simulation. In this case, low-frequency modes contained much of the potential enhancement, so this correction pattern reached 74% of the theoretical enhancement with only a k-space of 11 by 11 frequencies. Low-frequency corrections, corresponding to forward scattering, is the problem Fourier-based wavefront shaping was designed for [5].

Additionally, we can compare the Fourier-based algorithm simulation with the algorithm we have shown previously: SSA. The corrected wavefront is shown in figure 7. Its simulated enhancement reached 68% with the same number of modes as used in the Fourier-based algorithm.



(a) (b)

Figure 7: a): The image is set as an 'aberration' to the simulation objective plane. b) The resulting wavefront produced by a stepwise sequential algorithm with 121 modes.

Simulations in micromanager

These experiments were run in Python directly, but can also be integrated into µManager, as shown by Figure 8.

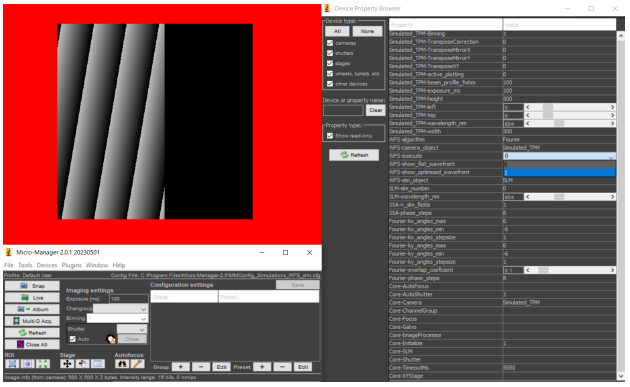
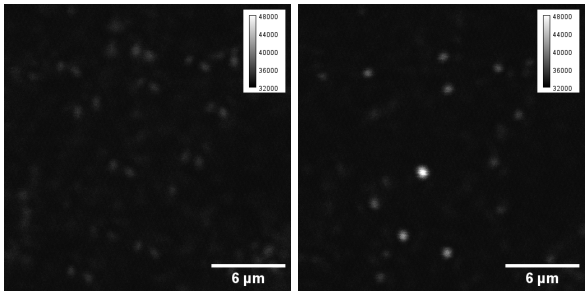


Figure 8: A simulation of wavefront shaping experiment in the µManager environment. The area marked red shows the patterns displayed on the SLM.

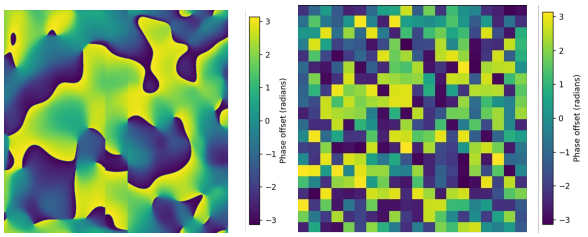
Experimental results

OpenWFS was tested on a forward-scattering PDMS with a single scattering interface of grid 120 containing fluorescent beads with a 500 nm diameter [12]. The experiments were performed at a depth of 91 µm below the scattering surface. The effect of this procedure is shown in figure 9.



(a) (b)

Figure 9: Images of fluorescent beads in a forward scattering sample a) before and b) after Fourier-based wavefront shaping with 169 modes



(a) (b)

Figure 10: Correction patterns found by a) Fourier-based and b) stepwise sequential algorithm. Note that the correction patterns gave rise to a signal enhancement of 2.9 and <1.0, respectively.

As shown in Figure 10, the correction patterns show similar features. Due to the low signal-to-noise of the SSA signal, however, it did not result in signal enhancement. The Fourier-based algorithm was able to produce significant enhancement, as seen in figure 9.

Future implementations

From the Fourier-based wavefront shaping perspective, intelligent techniques can be developed to scan the wavefront shaping modes more efficiently. Additionally, with the rise of model-based wavefront shaping, this platform can be used to connect the most recent developments with potential users.

Conclusions & Discussion

In this work, a standardised framework for wavefront shaping is presented. Due to the PyDevice plugin in µManager, any Python-based class can be integrated into the most widely used open-source microscopy framework available. Therefore, its use extends beyond wavefront shaping. We present a user- and developer-focussed open-source wavefront shaping code base which can be easily adopted and is highly modifiable, enabling further research and development. On top of this, the integration of µManager in this project enables the translation to common use in a robust, consistent and flexible manner. We have demonstrated the ease of development by implementing two established wavefront shaping algorithms and testing them in simulation and experiments.

In conclusion, we encourage the reader to join us in developing new algorithms and components for this framework. Code and documentation for this project is available at github.com/IvoVellekoop/openwfs.

Acknowledgements

The authors would like to thank Henry Pinkard for correspondence on the µManager environment, as well as Mark Tsuchida and Kyle Marchuk for their correspondence on laser-scanning software.

References

[1] Ishimaru A. Limitation on image resolution imposed by a random medium. Applied optics. 1978 2;17:348. doi:10.1364/AO.17.000348.

[2] Vellekoop IM. Feedback-based wavefront shaping. Optics Express, Vol 23, Issue 9, pp 12189-12206. 2015 5;23:12189–12206. doi:10.1364/OE.23.012189.

[3] Aoyagi Y, Kawakami R, Osanai H, Hibi T, Nemoto T. A rapid optical clearing protocol using 2,2-thiodiethanol for microscopic observation of fixed mouse brain. PLoS ONE. 2015;10. doi:10.1371/journal.pone.0116280.

[4] Vellekoop IM, Mosk AP. Focusing coherent light through opaque strongly scattering media. Opt Lett. 2007 Aug;32(16):2309–2311. doi:10.1364/OL.32.002309.

[5] Mastiani B, Osnabrugge G, Vellekoop IM. Wavefront shaping for forward scattering. Optics Express. 2022 10;30:37436. doi:10.1364/oe.470194.

[6] Liu Y, Ma C, Shen Y, Shi J, Wang LV. Focusing light inside dynamic scattering media with millisecond digital optical phase conjugation. Optica. 2017 Feb;4(2):280–288. doi:10.1364/OPTICA.4.000280.

[7] Tzang O, Niv E, Singh S, Labouesse S, Myatt G, Piestun R. Wavefront shaping in complex media with a 350 kHz modulator via a 1D-to-2D transform. Nature Photonics. 2019;doi:10.1038/s41566-019-0503-6.

[8] ižmár TC Mazilu M, Dholakia K. In situ wavefront correction and its application to micromanipulation. 2010;doi:10.1038/NPHOTON.2010.85.

[9] Streich L, Boffi JC, Wang L, Alhalaseh K, Barbieri M, Rehm R, et al. High-resolution structural and functional deep brain imaging using adaptive optics three-photon microscopy. Nature Methods 2021 18:10. 2021 9;18:1253–1258. doi:10.1038/s41592-021-01257-6.

[10] Pinkard H, Stuurman N, Ivanov IE, Anthony NM, Ouyang W, Li B, et al. Pycro-Manager: open-source software for customized and reproducible microscope control. Nature Methods. 2021;18:226–228.

[11] Astropy Collaboration, Price-Whelan AM, Lim PL, Earl N, Starkman N, Bradley L, et al. The Astropy Project:

Sustaining and Growing a Community-oriented Open-source Project and the Latest Major Release (v5.0) of the Core Package. . 2022 Aug;935(2):167.

[12] Thendiyammal A, Osnabrugge G, Knop T, Vellekoop IM. Model-based wavefront shaping microscopy. Opt Lett. 2020 Sep;45(18):5101–5104. doi:10.1364/OL.400985.

[13] Lai P, Wang L, Wang L. Photoacoustically guided wavefront shaping for enhanced optical focusing in scattering media. Nat Photon. 2015 01;9:126–132. doi:10.1038/nphoton.2014.322.

[14] Horstmeyer R, Ruan H, Yang C. Guidestar-assisted wavefront-shaping methods for focusing light into biological tissue. Nature Photonics. 2015 08;9:563–571. doi:10.1038/nphoton.2015.140.

[15] Wavefront Shaping for Biomedical Imaging. Advances in Microscopy and Microanalysis. Cambridge University Press; 2019. doi:10.1017/9781316403938.

Chapter 4

A new, pathfinding wavefront shaping algorithm

In this chapter, we introduce our novel algorithm for wavefront shaping. This new algorithm is intended to improve performance and increase usability for the end user. In section 4.1, we will elucidate how this was built on top of the existing Fourier-based algorithm. In section 4.2 the theory and implementation will be discussed, and in section 4.3, we will compare the performance of the conventional Fourier-based algorithm with the improved Fourier-based algorithm.

4.1 Approach

The challenge in feedback-based wavefront shaping is to optically characterise a sample efficiently. As explained in chapter 1; this can be done in different ways. In this chapter, we will work on a feedback-based wavefront shaping algorithm. The problem of efficient wavefront shaping can be reduced to finding a measurement technique that finds the largest amount of information on the corrective wavefront in the lowest amount of measurements. As shown in [3], different algorithms can have different rates of convergence. This is explained by the fact that some measurements give more information on the ideal wavefront than others. Our new approach builds on the elegant Fourier basis wavefront shaping algorithm developed by Mastiani et al. [3].

In chapter 4.2, we will further detail our new approach. Then, we will show the results of wavefront shaping experiments using the Fourier algorithm by Mastiani et al. and the new algorithm. The experiments were performed on samples with a single scattering layer: a PDMS sample filled with fluorescent beads with a diameter of 0.5 µm, with a scattering PDMS layer moulded from grid 220 glass diffuser as described in the sample preparation by Thendiyammal et al. [4].

4.2 Theory & Implementation

In feedback-based wavefront shaping, we measures the effect of a scattering sample on the light field. We do this by dividing the problem in smaller bits we call modes. In SSA we divide the SLM up into several blocks, and determine the optimal phase for each block. The optimal phase of these blocks can then be combined to create a corrective wavefront.

Fourier-based wavefront shaping works on the same principle as SSA, but uses a different type of mode. Instead of measuring the effect of smaller SLM segments, we measure the

effect of tilted plane waves, as shown in figure 4.1. This approach was developed to compensate for forward scattering, where the scattered light only requires a small tilt to come back to the focus. The phase patterns in figure 4.1 show phase-wrapping patterns, which allows us to tilt a wave by more than 2π phase difference.

In this dual-reference Fourier-based wavefront algorithm, we split the SLM into two sides, left and right, with a small overlap. We measure the effect of the tilted plane waves by phase-stepping, and combine the two sides by comparing the relative phase of overlapping segments. Just like in SSA, the measurements are processed by applying the Fourier transform on the results of the phase-stepping for each mode, which gives us the phase and amplitude of each mode.

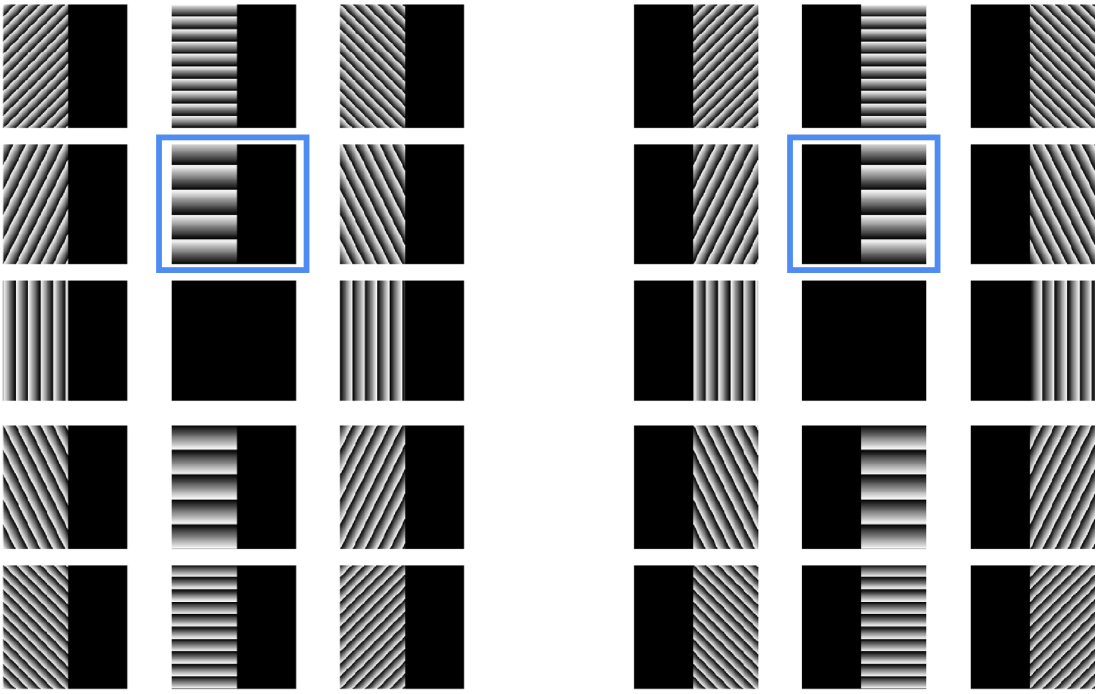


FIGURE 4.1: The displayed phase patterns for the left (left) and right(right) pupil in a Fourier dual reference experiment by Mastiani et al. [3]. Gray values represent values from -π to π. For illustration, note that the patterns signified in blue are represented by $(k_x, k_y) = (5, 0)$

In Fourier-based wavefront shaping, we need to determine which tilted plane waves are going to be measured, as there are (theoretically) infinite options. The matrix of measured tilted plane waves is called a k-space. There is no clear way to know, before the experiment is done, what the most effective k-space is. If we measure too little modes, we risk missing out on possible enhancement. If we measure too many modes, to procedure takes unnecessarily long without significant benefit. This problem is what we solve using the the new algorithm.

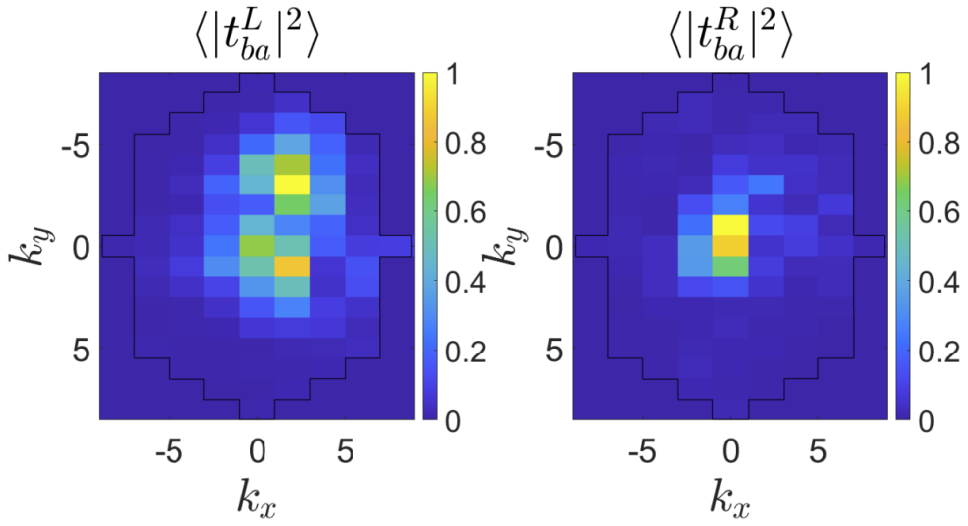


FIGURE 4.2: The normalised averaged amplitude of the feedback for each mode for the left (left) and right(right) pupil in a Fourier dual reference experiment by Mastiani et al. [3]. The black line envelopes the measured frequencies. As the measurements are done for the left and right pupils separately, we portray the Fourier-space of the found correction pattern for the right and left pupils separately.

Our new algorithm is based on the observation that strongly contributing modes are clustered in the k-space, as can be seen in figure 4.2. Additionally, we also observed that the different sides of the SLM do not necessarily have the same useful modes.

The algorithm we came up with is shown in figure 4.3. The algorithm begins by measuring nine modes near and on the origin in the frequency domain. Next, it will find the highest measured signal that borders unmeasured frequency domain elements, and measures these adjacent unmeasured elements. The ranking and measuring is repeated until a stopping criteria is met. Since this is a 2-step dual-reference algorithm, this algorithm is performed separately for the left and right side, and the results are combined after.

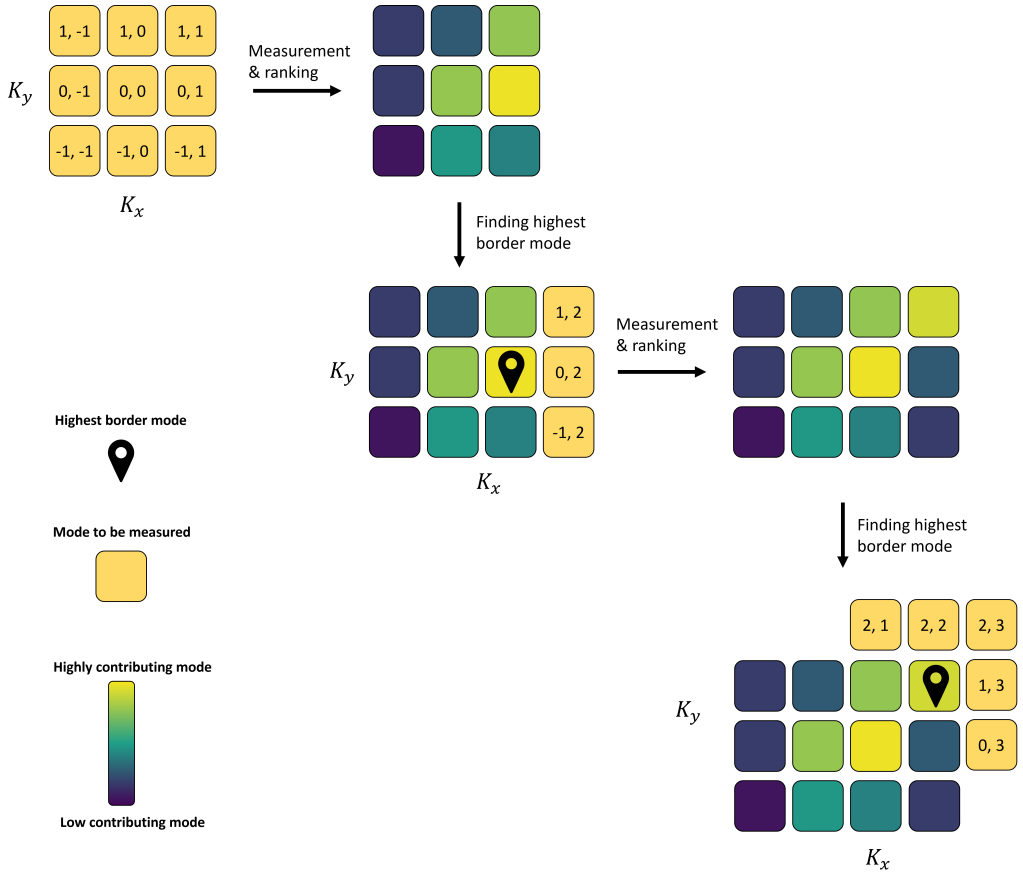


FIGURE 4.3: Graphical explanation of the pathfinding implementation

Because the Fourier modes are assumed to be orthogonal, it is possible to estimate intermediate enhancement during this procedure. If the effect on the overall feedback signal strength is low for a measured mode, the enhancement it can produce is also low. This allows for a rough projection of the potential enhancement as the experiment progresses.

4.3 Results

4.3.1 Algorithm comparison in simulation

To get a mechanistic understanding of the differences between the two algorithms, we will show the performance of the algorithms for different solutions. We will dub the conventional Fourier-based algorithm the basic Fourier approach, and our novel approach the pathfinding Fourier approach. Firstly, the extreme non-realistic cases. As the basic Fourier implementation is a growing square in k-space around $(k_x, k_y) = (0, 0)$, we expect optimal wavefront shaping for a random k-space with its main modes around $(0, 0)$. To achieve this, the optimal wavefront was set to an image, the public domain camera image from the SciPy.data set [20], as shown in figure 6 of chapter 3. This has the most of its highest effecting modes centred around $(k_x, k_y) = (0, 0)$, as seen in figure 4.4 (left). Figure 4.4 (right) shows the other extreme, where the 'correct' wavefront was set to a specific frequency convolved with a Gaussian. This results in a distinct peak in the k-space, which the pathfinding algorithm can find well.

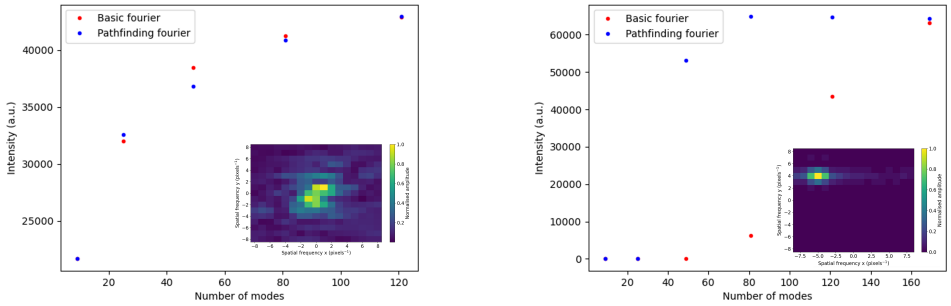


FIGURE 4.4: Simulation of wavefront shaping experiments using data generated from a SciPy sample image (left) and data generated to contain one specific spatial frequency (right). The amplitude left SLM side measured modes are plotted in the right bottom corners. the plots show the simulated intensities of correction patterns generated by the basic and pathfinding Fourier algorithms for 9, 25, 49, 81 and 121 modes.

As seen in figure 4.2, we can expect k-spaces we will encounter in practice to be between these extremes. In order to explore the true experimental case without the loss of compatibility due to experimental circumstances, we optimised a wavefront using the basic Fourier algorithm and set the optimised wavefront as the correct wavefront for the simulation.

In order to represent the effects of the pathfinding Fourier algorithm, we show the left SLM side k-space, or transformation row, as it is the major influence on the performance differences of the two algorithms in this case. In figure 4.5 (left), we see a large representation of the k-space containing $17^2 = 289$ modes. In figure 4.5 (right), the intensity of the simulated feedback of the generated wavefront is compared between basic and pathfinding Fourier algorithms for a number of modes, and figure 4.6 and 4.7 show the modes measured for the left SLM side for each of the data points in figure 4.5 right.

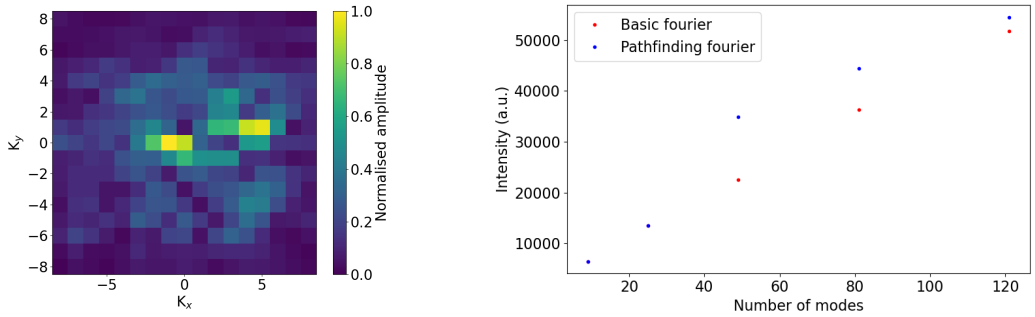


FIGURE 4.5: Simulation of wavefront shaping experiments using data from a real experiment. The amplitude left SLM side feedback for each mode (left), the simulated intensities of correction patterns generated by the basic and pathfinding Fourier algorithms for 9, 25, 49, 81 and 121 modes.

For the basic Fourier implementation, the increasing number of modes is simply increasing perfect squares centred in $(k_x, k_y) = (0, 0)$, as shown in figure 4.6.

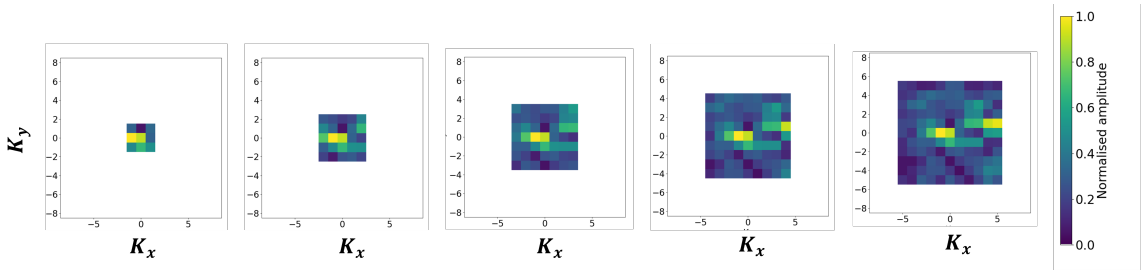


FIGURE 4.6: The normalized amplitude of the left SLM side measured feedback for each mode as computed by the basic Fourier algorithm for 9, 25, 49, 81 and 121 modes (left to right).

On the contrary, the pathfinding Fourier algorithm expands as detailled in figure 4.3. In order to compare the results with the basic Fourier implementation, the amount of modes was cut off after 9, 25, 49, 81 and 121.

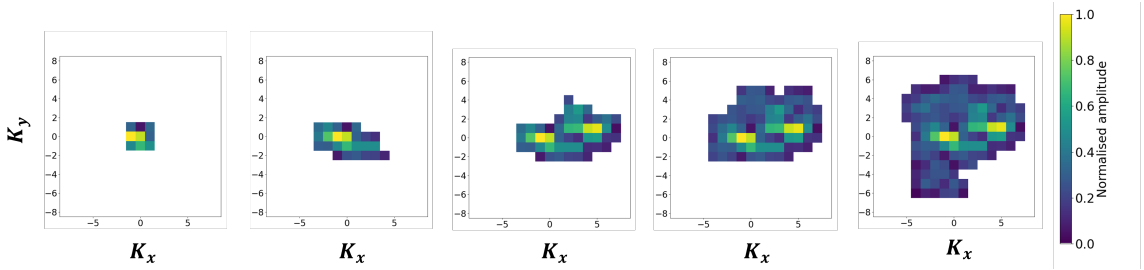


FIGURE 4.7: The normalized amplitude of the left SLM side measured feedback for each mode as computed by the pathfinding Fourier algorithm for 9, 25, 49, 81 and 121 modes (left to right).

When the comparison between the basic and pathfinding Fourier implementation is made, the most significant increase can be seen in the 49 modes case. When we purely look at the amplitudes of the modes in figure 4.6 and 4.7, we can indeed see that the pathfinding Fourier algorithm was able to select better modes.

4.3.2 Experimental algorithm comparison

In order to present the efficiency of our new algorithm, we commence with a normal Fourier-based wavefront shaping procedure at a depth of 80 µm with a 120 grit scattering sample. The amplitudes of the complex numbers representing the measured angles from this experiment are presented in figure 4.8.

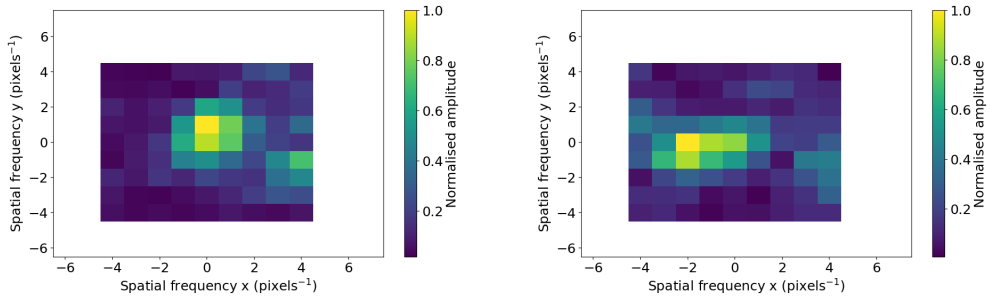


FIGURE 4.8: The normalised averaged amplitude of the feedback for each mode for the left (left) and right (right) pupil in measured data in a basic Fourier dual reference experiment for 49 modes per pupil

Next, the pathfinding Fourier implementation was tested with a maximum mode number of 30, resulting in figure 4.9.

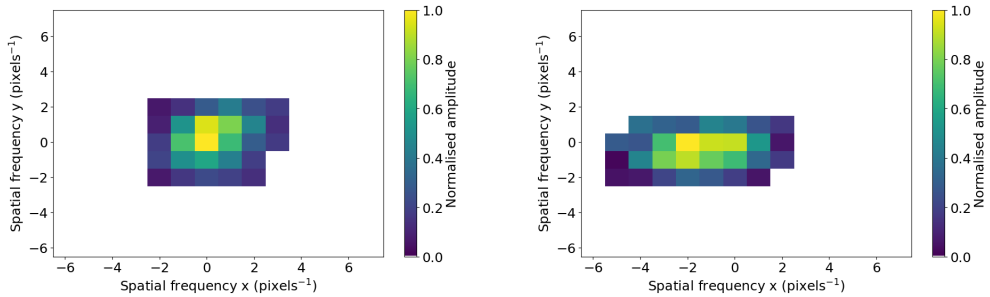


FIGURE 4.9: The normalised averaged amplitude of the feedback for each mode for the left (left) and right(right) pupil in measured data in a pathfinding Fourier dual reference experiment for 30 modes per pupil

As can be seen, the corrections are similar, and the algorithm successfully finds the most contributing modes, comparably to figure 4.8. As mentioned before, the intermediate enhancement of the algorithm can be measured by constructing the wavefront from the currently measured modes.

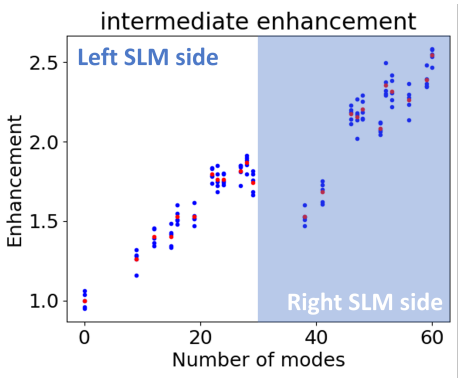


FIGURE 4.10: The intermediate enhancement of an experiment using the pathfinding Fourier algorithm, found by computing intermediate wavefronts from 0 to 60 modes and measured in order 5 times. Separate results are in blue. The average value per wavefront is signified in red. Enhancement was defined as the feedback signal divided by the average feedback signal for a flat wavefront

Because the pathfinding algorithm is a 2-step algorithm, one SLM side is optimised first. During the measurement of the intermediate enhancement, the right side is left uniform. For the intermediate enhancement measurements of the right side, the wavefronts are combined. This causes a drop of intermediate enhancement when the right SLM modes are added, as the overlapping SLM region is the average between the two optimisations. When we conduct the experiment for 50 modes, we find the results of figure 4.11.

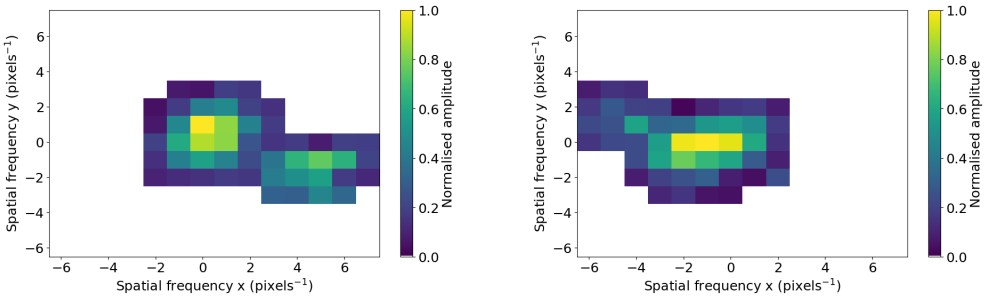


FIGURE 4.11: The normalised averaged amplitude of the feedback for each mode for the left (left) and right(right) pupil in a pathfinding Fourier dual reference experiment for 50 modes per pupil

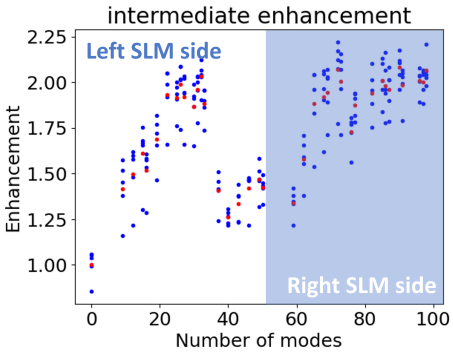


FIGURE 4.12: The intermediate enhancement of an experiment using the pathfinding Fourier algorithm, measured from 0 to 100 modes in order 5 times. Separate results are in blue. The average value per wavefront is signified in red.

In this case, the right SLM side shows the expected behaviour as the enhancement initially rises and levels off to a certain level of enhancement. The left SLM side however, shows a sudden yet unexplained drop in enhancement during the experiment around the 35^{th} measured node.

To conclude, we provided the results of one the optimisations that the new pathfinding wavefront shaping algorithm has performed in figure 4.13. Besides the very significant increase in signal strength, we can also note that the signal of some beads is smaller than before the measurement. This shows that the depth resolution was enhanced by the wavefront shaping procedure as well.

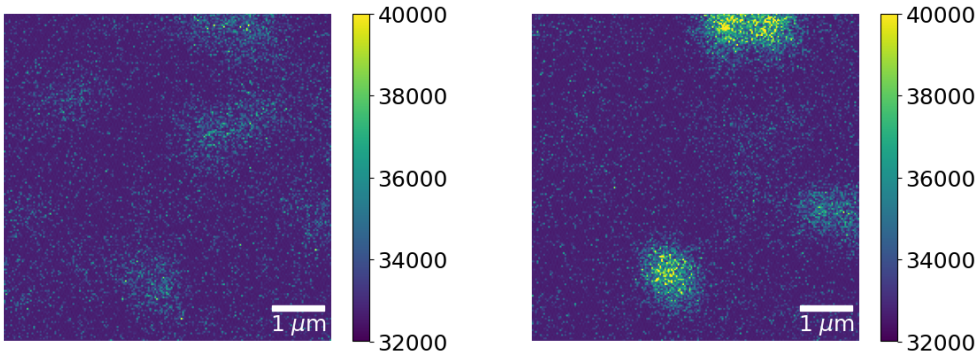


FIGURE 4.13: Images after a 30 modes pathfinding Fourier wavefront shaping procedure with a flat wavefront (left) and optimised (right) wavefront.

4.4 Discussion

In this chapter, we have introduced a new wavefront shaping algorithm that selects the optimal Fourier modes for wavefront shaping automatically. This algorithm relieves the end user of the burden of selecting modes through trial and error and makes the procedure more robust and efficient.

As shown by the extremes in figure 4.4, we expect the pathfinding algorithm to perform either similar or better than the current implementation for the same amount of modes. Perhaps more importantly, this is a step towards robustness and simplicity, as it reduces

the user-set parameters for the algorithm without hurting performance. Measuring intermediate enhancement like in figure 4.10 allows for additional functionalities, like using the plateauing enhancement as a stopping condition for the algorithm or rejecting modes that decrease in enhancement as seen in figure 4.12.

In section 1.4, these questions restated here were introduced to direct our development for an improved algorithm. We discuss these questions individually afterwards.

1. Can the number of input parameters of golden standard wavefront shaping algorithms be reduced?
2. Can we build a more efficient wavefront shaping algorithm?
3. How can we make the wavefront shaping algorithms robust for more sample types?
4. Can the potential of the wavefront shaping procedure be gauged before or during measurement?

4.4.1 Reduction of input parameters

In the previous algorithm, the input parameters determine which Fourier modes are measured. The pathfinding algorithm has reduced the number of parameters to one, namely the maximum amount of measured modes, instead of requiring the user to define a square or a circle in the k-space with respectively 4 or 2 parameters. Further improvement could also be made by projecting how much time a measurement will take.

4.4.2 Increasing efficiency

As mentioned before, it is expected that the pathfinding algorithm performs at or above the efficiency of the basic algorithm. Typically, a square or circle in the frequency domain centred in $(k_x, k_y) = (0, 0)$ is a reasonable estimation of the most contributing modes, but it is never precisely correct, and especially in cases where this estimation is not valid, the efficiency is greatly enhanced. In our algorithm, we have implemented a functionality that re-measures the most contributing modes with a higher number of phase steps. From the experimental data, it was inconclusive if this significantly improved efficiency, so further investigation in that regard may show merit.

4.4.3 Increase robustness

The pathfinding algorithm increases robustness compared to the original implementation, as it allows for more efficient solving of complex and askew forward scattering. Nonetheless, it cannot solve higher-frequency aberrations with high efficiency, like those seen in heavy defocus or spherical aberrations. In order to create a truly robust wavefront shaping algorithm, the algorithm should be able to choose between or able to combine multiple procedures, suited for different types of aberrations. As an example, a wavefront shaping procedure could consist of an initial defocus correction followed by a Fourier-based wavefront shaping correction.

4.4.4 Gauging potential

The final research question was directed towards gauging the potential merit of a wavefront shaping procedure. If the potential gain of a wavefront shaping procedure could be estimated before or during a measurement, the user could be provided with that information to influence the selection if and how the procedure should be performed. For example, if the amount of potential enhancement could be projected for a certain measurement time, a user could intuitively select the desired number of measured modes. This has been attempted in two manners. The first manner is seen in figure 4.10, where intermediate enhancement is measured for an increasing number of measured modes. Functionality was built into the pathfinding algorithm to measure the intermediate enhancement during the procedure. A significant disadvantage of this implementation is that the computing of a wavefront by mode-for-mode computing a corrective pattern and adding it the the correction front is a rather time-consuming task. These steps can be made significantly more efficient by remembering the intermediate wavefronts and only adding the newly measured modes, but this has not been implemented.

The second potential method for intermediate gauging is by estimating the signal-to-noise ratio of the measured modes. If the algorithm uses mostly orthogonal modes, the signal strength of the measured modes should relate directly to the measured enhancement. This method negates the disadvantage of additional measurements and the on-the-fly computing of the corrective wavefront but has the disadvantage of indirectly estimating enhancement. As seen in figure 4.12 and discussed earlier, a high measured signal strength for a mode does not necessarily translate into actual enhancement.

Additionally, an investigation was done on whether it would be valuable to 'probe' some frequencies to estimate the location and strength of Fourier modes before commencing the measurement. Although this shows potential for very specific situations, it was not continued. A potential combination of the pathfinding and 'probing' ideas would entail measuring a small number of random modes in a confined area and starting the pathfinding wavefront shaping procedure from the mode that has the highest amplitude.

In conclusion, the pathfinding algorithm is a promising technique and a significant step towards automated wavefront-shaping. This algorithm increases both the functionality and the user-friendliness of the two-photon wavefront shaping microscope.

Chapter 5

Conclusions and outlook

We have developed a Python-based open-source platform for wavefront shaping, called OpenWFS, which both developers and end users can use. We have used this platform to develop a novel version of the state-of-the-art Fourier-based wavefront shaping technique that is more robust, efficient and user-friendly.

OpenWFS is a user- and developer-focused open-source wavefront shaping code base which can be easily adopted and is highly modifiable, enabling further research and development. We have integrated OpenWFS with µManager, a general-purpose, widely used open-source microscope control software platform. This Python-to-µManager bridge, is such a robust connection that the developers of µManager are glad to integrate it into the platform. This means we have engineered a platform that meets our goal to simplify wavefront shaping development, and our goal to simplify the use of currently existing wavefront shaping algorithms for the end user.

Using OpenWFS, a novel pathfinding wavefront shaping algorithm was developed that automatically selects the optimal modes. This algorithm is more efficient and robust than current algorithms and relieves the end user from the task of selecting modes. The on-the-fly selection of modes additionally opens the development of intermediate enhancement gauging, which promises to make the algorithm even more automatic and robust. With that, we have answered the research questions towards input parameter reduction, robustness, efficiency, and intermediate enhancement gauging.

The work presented in this thesis also opens doors to future developments. The OpenWFS framework opens the technique of wavefront shaping to many potential users and developers. Efforts must be sustained to promote this framework such that it reaches that potential. The newly developed pathfinding Fourier-based wavefront shaping algorithm has been shown to be effective. However, it shows much promise for building extra functionality on top of the current state. Using signal-to-noise ratio as a metric to decide the merit of further measurements is promising.

Furthermore, we would strongly encourage the development of a multi-algorithm wavefront shaping procedure that tackles different types of aberrations efficiently and incorporates model-based wavefront shaping into that multi-algorithm procedure. This will bring us even further towards wavefront shaping at the push of a button.

Bibliography

[1] I. M. Vellekoop and A. P. Mosk. "Focusing coherent light through opaque strongly scattering media". In: *Opt. Lett.* 32.16 (2007), pp. 2309–2311. DOI: 10.1364/OL.32.002309.

[2] *Wavefront Shaping for Biomedical Imaging*. Advances in Microscopy and Microanalysis. Cambridge University Press, 2019. DOI: 10.1017/9781316403938.

[3] Bahareh Mastiani, Gerwin Osnabrugge, and Ivo M. Vellekoop. "Wavefront shaping for forward scattering". In: *Optics Express, Vol. 30, Issue 21, pp. 37436-37445* 30.21 (2022), pp. 37436–37445. ISSN: 1094-4087. DOI: 10.1364/OE.470194.

[4] Abhilash Thendiyammal et al. "Model-based wavefront shaping microscopy". In: *Optics Letters* 45 (18 2020), p. 5101. ISSN: 0146-9592. DOI: 10.1364/ol.400985.

[5] Gerwin Osnabrugge, Saroch Leedumrongwatthanakun, and Ivo M. Vellekoop. "A convergent Born series for solving the inhomogeneous Helmholtz equation in arbitrarily large media". In: *Journal of Computational Physics* 322 (2016), pp. 113–124. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2016.06.034.

[6] Zahid Yaqoob et al. "Optical phase conjugation for turbidity suppression in biological samples". In: *Nature photonics* 2 (Feb. 2008), pp. 110–115. DOI: 10.1038/nphoton.2007.297.

[7] Meng Cui and Changhuei Yang. "Implementation of a digital optical phase conjugation system and its application to study the robustness of turbidity suppression by phase conjugation". In: *Opt. Express* 18.4 (2010), pp. 3444–3455. DOI: 10.1364/OE.18.003444.

[8] Richard K P Benninger and David W Piston. "Two-Photon Excitation Microscopy for the Study of Living Cells and Tissues". In: *Curr Protoc Cell Biol.* (2013). DOI: 10.1002/0471143030.cb0411s59.

[9] Fritjof Helmchen and Winfried Denk. *Deep tissue two-photon microscopy*. 2005. DOI: 10.1038/nmeth818.

[10] Wolfgang Mittmann et al. "Two-Photon Calcium Imaging of Evoked Activity from L5 Somatosensory Neurons in vivo." In: *Nature neuroscience* 14 (July 2011), pp. 1089–93. DOI: 10.1038/nn.2879.

[11] D Huber et al. "Multiple dynamic representations in the motor cortex during sensorimotor learning". In: *Nature* 484 (Apr. 2012), pp. 473–8. DOI: 10.1038/nature11039.

[12] Hyo Won Lee et al. "Visualization of vesicular transport from the endoplasmic reticulum to lysosome using an amidine derived two-photon probe". In: *Chem. Commun.* 53 (45 2017), pp. 6097–6100. DOI: 10.1039/C7CC01518F.

[13] Ryotaro Kawasoe et al. "Two-photon microscopic observation of cell-production dynamics in the developing mammalian neocortex in utero". In: *Development, Growth & Differentiation* 62.2 (2020), pp. 118–128. DOI: https://doi.org/10.1111/dgd.12648. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/dgd.12648.

[14] Bahareh Mastiani, Tzu-Lun Ohn, and Ivo M. Vellekoop. "Scanning a focus through scattering media without using the optical memory effect". In: *Opt. Lett.* 44.21 (2019), pp. 5226–5229. DOI: 10.1364/OL.44.005226.

[15] National Institutes of Health. *RePORT ⟩ RePORTER*.

[16] *About — ScanImage latest documentation*.

[17] George Ward, Alex Murray, and Christian Wilms. "Open-Source Software for Controlling Two-Photon Laser Scanning Microscopes". In: *Microscopy Today* 25.5 (2017), pp. 12–17. ISSN: 1551-9295. DOI: 10.1017/s1551929517000785.

[18] *Two-Photon Imaging Equipment | Scientifica*.

[19] *OpenScan – Laboratory for Optical and Computational Instrumentation*.

[20] Gregory Lee. *skimage.data API*. URL: https://scikit-image.org/docs/stable/api/skimage.data.html#skimage.data.camera (visited on 09/16/2023).

Appendix A

Current setup specifications

Laser

The laser used in the setup is the Spectra Physics Mai Tai HP. This is a mode-locked Ti:Saph laser. The pulse width of the laser is less than 100 femtoseconds, and the laser can be tuned to 690 - 1040 nanometers. The repetition rate is 80 MHz. It is currently controlled by the MatTai 2.x GUI control program, which is controlled through a USB cable. For alignment purposes, the laser can be put in a continuous-wave mode. This is done by performing some manual steps in the MaiTai GUI.

Spatial Light Modulator (SLM)

The SLM is a device that is able to shape the wavefront of light. This is done by liquid crystal devices that have electro-responsive crystals that can be oriented and reoriented by applying a certain voltage. This reorienting changes the path length of the light, which corresponds to a phase shift relative to the other light. As one might deduce, it requires a bigger path length difference to shift the phase of 1040 nm light by π than it takes to shift the phase of 480 nm light by π. The orientation of the crystals can then phase shift the incoming light by 0 to 2π radians. As the laser is coherent, a 2π phase shift practically equals no phase shift at all. Therefore, as the SLM is able to shift the light further than 2π, all shifts can be seen as shifts between 0 to 2π radians. As the voltage-to-phase-shift relation is typically not fully linear, a wavelength-dependent lookup table associates the 0 to 255 grey value transmitted with an HDMI cable with the right voltage such that the grey values linearly correlate with the 0 to 2π phase shift.

Galvanometer mirrors

Galvos, or Galvanometer mirror positioning systems are designed to accurately steer a laser beam. In this application, 2 of them are used to control both x and y motions separately. The model used in the setup is the GVS111(/M). It is controlled by output voltages through a NI breakout box by the data acquisition card, which can directly be controlled by Matlab or Python using the NI toolbox.

Photo-multiplier tube (PMT)

The PMT, or photomultiplier tube, is a sensitive optical detector that transfers an incoming photon into a current. This is done by a dynode cascade, which amplifies the number of electrons in the cascade exponentially. This results in a very high sensitivity. The signal, however, still needs to be converted from current to voltage in order to digitise it. This voltage can then additionally be gained, allowing for detection through the NI box by the DAC.

NI breakout box & DAC

The data acquisition card used in the setup is a PCIe-6363, which is an X-series DAQ. The data acquisition card is connected to a national instruments device SCB-68A breakout box. This is a device to translate the voltage inputs and outputs of the galvos, PMT & other devices to a digital system. The DAC on the lab computer has room for 4 analogue outputs, but the breakout box supports 2. Therefore, there are 2 SCB-68A breakout boxes. Currently, only 2 of the analogue outs are taken by the galvos, and the other 2 are free. There are 8 analogue inputs per breakout box, which is not a limiting factor in the current setup.

Filters

Before the PMT, filters are placed to selectively filter out wavelengths. The multi-photon filter 680/SP is always in the beam path, which is a short-pass filter to filter out the excitation wavelength. On top of that, 5 further band-pass filters are available between 417 nm and 614 nm. Currently, there is a ThorLabs motorized filter wheel FW102C installed in the setup, but it is not connected to the computer. This filter wheel allows the selection between the 5 filters.

Software

The current setup uses ScanImage software to control the microscope & visualize its detection. It is an open-source software package that is specifically built for laser-scanning purposes. The ScanImage software can be controlled through its GUI, which has many different windows for different purposes. Through the Data Acquisition Toolbox, it is able to communicate through the DAC. As of 2019, ScanImage has been commercialized. ScanImage has moved away from its open-source model and transitioned to a licensing model. Due to this, the current implementation is unsupported and prone to be outdated. Additionally, some of its functionalities should be automated & clarified for the common user.

Appendix B

Expansion of platform scoring

B.1 User requirement feedback

In order to make a well-considered choice for the platform, the a set of requirements was defined. This set was then refined by the users, both by judging the set on importance on a scale from 0 to 10, but also by allowing the users to add their own requirements. Here are the average grades of the response of the users, rounded to the nearest tenth, there were 6 respondents, varying on levels of expertise.

General need to move away from ScanImage: 4.1/10

Hardware requirements

Laser scanning adaptable framework	9.7
Wavefront shaping adaptable	9.3
Galvo controls	9.3
DAQ control	9.5
SLM control	9.7
Laser power & shutter	7.5
Laser wavelength & settings adjustment	8.3
Stage control	7.3
Colour filter wheel selection	7.8
3D scanning (2D & Z control)	9.3

GUI requirements

Pixel dwell time adjustment	9.7
ROI size adjustment	8.5
Pixel size adjustment	9
Image visualisation	9.5
Zoom functionality	9.7
Bi- unidirectional scan	6.3
Stage control	8

User experience requirements

Ease of use (basic functionalities, common biologist)	8.8
Ease of use (complex functionalities, BPMI researcher)	8.5
User community size & platform support	7.3
Open-source licence	7
Modularity / ease of modification & expansion	8.5

Automation requirements

Automatic gain adaptation	8.2
Automatic gain reinitialization	7.8
Automatic conventional wfs algorithms	8.5
Metadata generation	9
Automated sample properties detection	6.3
Quantifyable 3D scanning (+ stage control + model based coordinate computation of focus)	6.5

B.2 Scores by platform

1-2: Very difficult, deal-breaking and/or very time-consuming 3-4: Difficult, sub-optimal and/or time-consuming 5-6: Requirement requires serious research, investment, or concession, but is undoubtedly reachable. 7-8: The requirement can be met with some effort, or the current state of the requirement is workable. 9-10: The current state of this requirement is nearly or fully met.

TABLE B.1: A weighed decision table using feedback from current users.

Hardware	Weight	µManager	Score	ScanImage	Score
Laser scanning adaptable	9.67	2	19.3	8	77.33
Wavefront shaping adaptable	9.33	8	74.7	6	56
Galvo controls	9.33	6	56	8	74.67
DAQ control	9.5	10	95	10	95
SLM control	9.67	6	58	8	77.33
Laser power & shutter	7.5	8	60	8	60
Laser wavelength & settings adjustment	8.33	8	66.7	8	66.67
Stage control	7.33	8	58.7	8	58.67
Colour filter wheel selection	7.83	10	78.3	8	62.67
3D scanning (2D & Z control)	9.33	7	65.3	4	37.33
Project Development					
Projected development time	20	3	60	7	140
Ease of implementing new feature	10	8	80	6	60
User Experience Requirements					
Ease of use (basic functionalities, common biologist)	8.83	–	0	–	0
Ease of use (complex functionalities, BPMI researcher)	8.5	–	0	–	0
User community size & platform support	7.33	8	58.7	3	22
Open-source licence	7	10	70	8	56
Modularity / ease of modification & expansion	8.5	8	68	6	51
Automation Requirements					
Automatic gain adaptation	8.17	0	0	0	0
Automatic gain reinitialization	7.83	0	0	0	0
Automatic conventional wfs algorithms	8.5	0	0	0	0
Metadata generation	9	9	81	7	63
Automated sample properties detection	6.33	0	0	0	0
Quantifiable 3D scanning	6.5	0	0	0	0
		µManager:	**1049.667**	**ScanImage:**	**1057.667**

Appendix C

Code templates for the OpenWFS framework

Detector

```python
class Detector(Protocol):
    """Minimal implementation of a detector object. Any detector must implement these
    members and attributes"""

    data_shape: tuple[int]
    """shape of the data returned by this detector (from numpy.shape). Read only."""

    measurement_time: float
    """In seconds. Read only. Used for synchronization, and should include the
    measurement time of any child detectors (see Processor)."""

    def trigger(self) -> None:
        """Triggers the detector to take a new measurement. Typically, does not wait
        until the measurement is finished. A detector may have a hardware
        trigger, in which case calls to trigger() may be ignored."""
        pass

    def read(self) -> np.ndarray:
        """Returns the measured data, in the order that the triggers were given.
        This function blocks until the data is available and raises a
        TimeoutError if it takes too long to obtain the data (typically because
        the detector was not triggered)."""
        pass
```

SLM

```python
class SLM(Protocol):
    phases: np.ndarray

    def update(self, wait_factor=1.0, wait=True):
        """Refresh the SLM to show the updates phase pattern.

        If the SLM is currently reserved (see `reserve`), this function waits until the
        reservation is almost (*) over before updating the SLM. The SLM waits for the
        pattern of the SLM to stabilize before returning.
```

```
        *) In case of SLMs with an idle time (latency), the image may be sent to the
        hardware already before the reservation is over, as long as the actual image on
        the SLM is guaranteed not to update until the reservation is over.

        :param wait_factor: time to wait for the image to stabilize.
        Default = 1.0 should wait for a pre-defined time (the `settle_time`)
        that guarantees stability for most practical cases. Use a higher value to allow
        for extra stabilization time, or a lower value if you want to trigger a measurement
        before the SLM is fully stable.

        :param wait: when set to False, do not wait for the image to stabilize but
        reserve the SLM for this period instead. This can be used to pipeline
        measurements (see `Feedback`). The client code needs to explicilty call `wait`
        to wait for stabilization of the image.
        """
        pass

    def wait(self):
        """Wait for the SLM to become available. If there are no current reservations,
        return immediately."""
        pass

    def reserve(self, time: Quantity[u.ms]):
        """Reserve the SLM for a specified amount of time.
        During this time, the SLM pattern cannot be changed."""
        pass
```

1D stage

```
class Stage(Protocol):
    step_size: Quantity[u.um]
    """Step size in µm"""

    position: Quantity[u.um]
    """Position in µm. Setting the position causes the stage to start moving to
    that position. Reading it returns the current position (note that the stage may still
    be moving!). Overwriting this attribute while the stage is moving causes it to
    start moving to the new position. Also see :func:`~wait`. Stages should use the
    step_size to convert positions in micrometers to positions in steps, using the
    equation `steps = round(position / step_size)`. This way, code that uses the
    stage can also choose to make single steps by moving to a position n * step_size.
    """

    def home(self) -> None:
        """Homes the stage. This function does not wait for homing to complete."""
        pass

    def wait(self) -> None:
        """Wait until the stage has finished moving.
        This should include any time the stage may 'vibrate' after
```

```
        stopping."""
        pass
```

XY stage

```
class XYStage(Protocol):
    x: Quantity[u.um]
    y: Quantity[u.um]
    step_size_x: Quantity[u.um]
    step_size_y: Quantity[u.um]

    def home(self) -> None:
        pass

    def wait(self) -> None:
        pass
```

Camera

```
class Camera(Protocol):
    data_shape: tuple[int]
    measurement_time: Quantity[u.ms]
    top: int
    left: int
    height: int
    width: int

    def trigger(self) -> None:
        pass

    def read(self) -> np.ndarray:
        pass
```

Controller

```
class Controller:
    def __init__(self, detector: Detector, slm):
        self.M = None
        """ Number of elements in each measurement. Read only. Equal to
            np.prod(source.data_shape). Updated when 'reserve' is called"""

        self.N = None
        """ Total number of measurements in the buffer. Read only.
            Set by 'reserve' function. """

        self.data_shape = detector.data_shape
        """ Shape of the original data, before flattening into column of length M."""
```

```
        self.slm = slm
        """Spatial light modulator object. An algorithm sets or changes slm.phases,
            and then calls measure() to schedule a measurement.
            Note that there is no need to call slm.update manually."""

        self._measurements_pending = 0
        self._measurements = None  # array storing all measurements.
        self._measurements_flat = None  # reshaped N x M view of the same array
        self._source = detector  # detector to take the feedback from.
        self._n = None  # current measurement number, must be < N

    def __del__(self):
        # clear remaining measurements. todo: implement aborting measurements?
        while self._measurements_pending > 0:
            self._await_data()

    def reserve(self, shape):
        """ Reserve space for storing measurements. Must be called before the first
            call to 'measure'. """
        self._n = 0
        self.N = np.prod(shape)
        self.M = np.prod(self._source.data_shape)
        self.data_shape = self._source.data_shape
        self._measurements = np.zeros((*shape, self.M), dtype="float32")
        self._measurements_flat = np.reshape(self._measurements, (self.N, self.M))

    def measure(self):
        """ Schedule a measurement. A measurement corresponds to updating the SLM,
            waiting for the image to stabilize, triggering the detector and reading
            the data from the detector. There is no guarantee as to when this measurement
            is performed (measurements may even be performed out of order or batched
            together in some implementations). However, the data is guaranteed
            to end up in the 'measurements' array in the correct order."""

        # Update the SLM.
        self.slm.update(wait=False)

        # we can use this time to process a previous measurement (if any)
        self._await_data()

        self.slm.wait()  # wait for the image on the SLM to stabilize
        self._source.trigger()  # trigger the camera
        self._measurements_pending += 1

        measurement_time = self._source.measurement_time
        if measurement_time is not None:
            # default fast wavefront shaping, continue processing (and even start next
            # frame) during measurement
            self.slm.reserve(measurement_time)
        else:
            # measurement time not known, wait till end of measurement
            self._await_data()
```

```
    def _await_data(self):
        """ Reads data from the detector (which includes processing it). If there are no
            outstanding measurements, this function does nothing. Note that other
            implementations of feedback may choose to batch-read data from
            the device and not even implement this function"""
        if self._measurements_pending > 0:
            data = np.array(self._source.read(), dtype="float32", copy=False)
            self._measurements_pending -= 1
            self._measurements_flat[self._n, :] = data.flat
            self._n += 1

    @property
    def measurements(self, partial=False):
        """ Called after performing a sequence of measurements to obtain the measurement
            data. Typically, you only call this function after performing all measurements
            that your reserved space for. If you want to peek before
            finishing all measurements, set 'partial'=True"""
        while self._measurements_pending > 0:
            self._await_data()
        if not partial and self._n != self.N:
            raise Exception(f"Measurement sequence not completed yet, only performed {
                            self._n} out of {self.N} "f"measurements.")
        return self._measurements

    def compute_transmission(self, phase_steps):
        """To do: calculate SnR"""
        t = np.tensordot(self.measurements, np.exp(-1j * np.arange(phase_steps) /
                        phase_steps * 2 * np.pi(len(np.shape(self.measurements)) - 2,[0]))
```

Gain implementation

```
class Gain:

    def __init__(self, port_ao="Dev4/ao0", port_ai="Dev4/ai0",
                 port_do="Dev4/port0/line0", reset=False, gain=0):
        self._port_ao = port_ao
        self._port_ai = port_ai
        self._port_do = port_do
        self._reset = reset
        self._gain = gain

    def set_gain(self, value):
        with ni.Task() as write_task:
            aochan = write_task.ao_channels.add_ao_voltage_chan(self._port_ao)
            aochan.ao_min = 0
            aochan.ao_max = 0.9

            write_task.write(value)

        return value
```

```
    def check_overload(self):
        with ni.Task() as task:
            task.ai_channels.add_ai_voltage_chan(self._port_ai)
            in_stream = task.in_stream

            data = in_stream.read(number_of_samples_per_channel=1)
            if data > 2.5:
                overload = True
            else:
                overload = False

            return overload

    def on_reset(self, value):
        if value:
            with ni.Task() as task:
                task.do_channels.add_do_chan(self._port_do,
                                             line_grouping=LineGrouping.CHAN_FOR_ALL_LINES)
                task.write([True])
                time.sleep(1)
                task.write([False])


    @property
    def reset(self) -> bool:
        return self._reset

    @reset.setter
    def reset(self, value: bool):
        self.on_reset(value)
        self._reset = value

    @property
    def gain(self) -> Annotated[float, {'min': 0, 'max': 0.9}]:
        return self._gain

    @gain.setter
    def gain(self, value: float):
        self.set_gain(value)
        self._gain = value
```

Dummy camera implementation

```
class RandomGenerator:
    """Demo device, used to test building device graphs.
    It generates random numbers for use in the Camera"""

    def __init__(self, min=0, max=1000, noise_type = NoiseType.UNIFORM):
        self._min = min
        self._max = max
```

```
        self._noise_type = noise_type

    def generate_into(self, buffer):
        buffer[:, :] = np.random.randint(self._min, self._max, buffer.shape, dtype=np.uint16)

    @property
    def min(self) -> Annotated[int, {'min': 0, 'max': 0xFFFF}]:
        return self._min

    @min.setter
    def min(self, value):
        self._min = value

    @property
    def max(self) -> Annotated[int, {'min': 0, 'max': 0xFFFF}]:
        return self._max

    @max.setter
    def max(self, value):
        self._max = value

    @property
    def noise_type(self) -> NoiseType:
        return self._noise_type

    @noise_type.setter
    def noise_type(self, value):
        if not value == NoiseType.UNIFORM:
            raise ValueError("Noise types other than uniform are not supported yet.")
        self._noise_type = value

class Camera:
    """Demo camera implementation that returns noise images. To test building device
    graphs, the random number generator is implemented as a separate object with
    its own properties."""

    def __init__(self, left=0, top=0, width=100, height=100, measurement_time:
                 Quantity[u.ms]=100 * u.ms, random_generator=None):
        if random_generator is None:
            random_generator = RandomGenerator()

        self._resized = True
        self._image = None
        self._left = left
        self._top = top
        self._width = width
        self._height = height
        self._measurement_time = measurement_time.to(u.ms)
        self._random_generator = random_generator

    def trigger(self):
        if self._resized:
```

```
            self._image = np.zeros(self.data_shape,
                                   dtype=np.uint16)
            self._resized = False
        self.random_generator.generate_into(self._image)

    def read(self):
        return self._image

    @property
    def data_shape(self):
        return self._height, self._width

    @property
    def left(self) -> int:
        return self._top

    @left.setter
    def left(self, value: int):
        self._top = value

    @property
    def top(self) -> int:
        return self._top

    @top.setter
    def top(self, value: int):
        self._top = value

    @property
    def width(self) -> Annotated[int, {'min': 1, 'max': 1200}]:
        return self._width

    @width.setter
    def width(self, value: int):
        self._width = value
        self._resized = True

    @property
    def height(self) -> Annotated[int, {'min': 1, 'max': 960}]:
        return self._height

    @height.setter
    def height(self, value: int):
        self._height = value
        self._resized = True

    @property
    def measurement_time(self) -> Quantity[u.ms]:
        return self._measurement_time

    @measurement_time.setter
    def measurement_time(self, value):
```

```
        self._measurement_time = value.to(u.ms)

    @property
    def random_generator(self) -> object:
        return self._random_generator

    @random_generator.setter
    def random_generator(self, value):
        self._random_generator = value
```