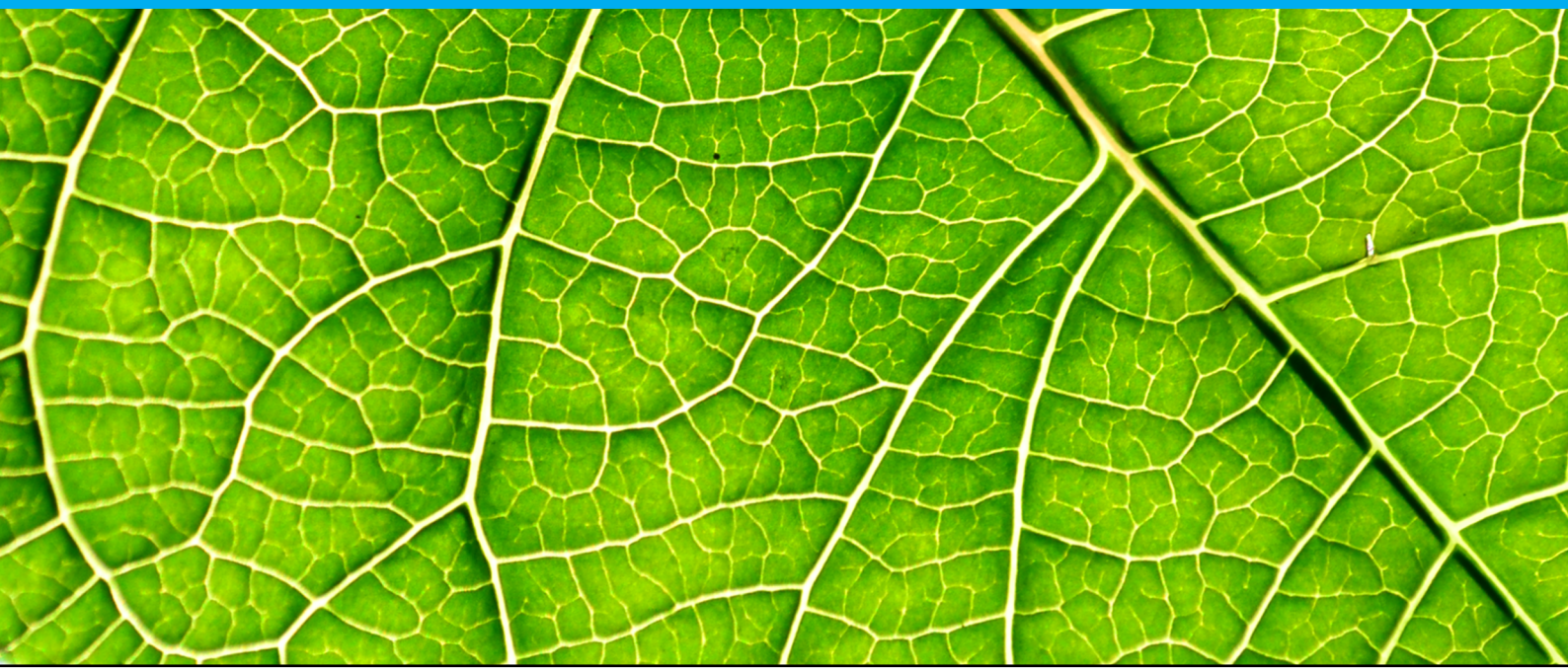
Online Computational Imaging Reconstruction

Guillermo Arto Sánchez

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Online Computational Imaging Reconstruction

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Guillermo Arto Sánchez

Monday, 3rd July 2017

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Online Computational Imaging Reconstruction

by

Guillermo Arto Sánchez

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: Monday, 3rd July 2017

Supervisor(s):

prof.dr.ir. Michel Verhaegen

Dean Wilding

Reader(s):

prof.dr. Gleb Vdovin

dr. Silvania Pereira

dr. Sjoerd Stallinga

Abstract

In optical imaging, image quality is not only determined by the system itself but also by the media in which light traverse. Differences in the refraction index of the media encountered by a light wavefront produces phase aberrations which distort the image received from the original object. Currently, there are two main approaches for solving this problem: *adaptive optics*, which rely on deformable mirrors and wavefront sensors for correcting the phase aberration before it reaches the imaging sensor; and *post-processing* techniques, which try to estimate the object after receiving distorted images.

Multi-Frame Blind Deconvolution (MFBD) methods are a family of algorithms that are capable of reconstructing the object by fusing the information carried by a set of differently aberrated images. These techniques are widely used in current optical systems, allowing a notable increase in image quality in most situations. However, there are cases in which they are not applicable; for example, looking at a dynamic object (e.g., a bird flying) or looking through static aberrations (e.g., in microscopy applications); but certain modifications in the optical system can be used for solving this problem.

Actual optical devices have only one aperture, thus creating one full-size image on the imaging sensor but, by segmenting the pupil, several images can be retrieved at the same time with different aberrations (i.e., light follows a distinct path for each aperture). However, using a multi-aperture system implies that there is less imaging sensor area available for each aperture, thus obtaining images with less resolution. Nonetheless, MFBD algorithms can usually be extended in order to support Super-Resolution (SR), a technique that allows the increase of the object resolution by retrieving extra information from the displacement between images.

This thesis is focused on the development of a functional prototype of a multi-aperture optical system that can do real-time object reconstruction. As a MFBD technique is needed, the novel Tangential Iterative Projections (TIP) algorithm (developed at Delft Center for Systems and Control) is selected. In order to achieve a fast and reliable reconstruction, the algorithm is: modified for increasing its robustness against noise, expanded in order to support SR and implemented efficiently in both CPU and GPU. Finally, the system is tested in a real environment, showing promising results.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

I would first like to thank my thesis supervisors, prof.dr.ir. Michel Verhaegen and Dean Wilding, for giving me the opportunity to work with them in such interesting field; giving me freedom to develop myself in the areas I felt more curiosity and steering me in the right direction whenever I felt lost.

Specially, I would like to thank Dean Wilding for all the patience, understanding, honesty and knowledge shared during this whole year. In addition, I am very grateful to him and dr. Paolo Pozzi for opening the door of their office to me, definitely helping me to focus in my thesis.

A very special gratitude goes out to all DCSC members for their kind words and help whenever I need it. It is a pleasure to work in such a warm environment.

Leaving my family and friends behind for starting by myself a new life abroad it is a challenging task. However, coming to Delft has been an amazing experience, feeling like at home basically thanks to the good friends I have met here. Because of that, I would like to express my deepest gratitude for their constant support and company.

Con esta tesis finalizo 8 años de mi etapa académica, en la que no sólo he aumentado mis conocimientos y competencias profesionales, sino que también he crecido como persona. Esto no hubiera sido posible sin el incondicional apoyo de mi familia y amigos; gracias a ellos soy quien soy hoy en día, teniendo mi más profundo agradecimiento.

Finalmente, quiero dedicarle estas palabras a la más importante de mis mentoras, mi madre:

Gracias, por enseñarme la humildad como virtud.
Gracias, por enseñarme la fortaleza en la adversidad.
Gracias, por enseñarme la felicidad como máxima.
Gracias, mamá, por enseñarme a vivir.

Delft, University of Technology
Monday, 3rd July 2017

Guillermo Arto Sánchez

Para los que no están hoy aquí.
Para los que no llegué a conocer.
Para los que se fueron hace mucho, hace no tanto y hace nada.

Chapter 1

Introduction

The common tendency in optical systems, such as telescopes and microscopes, is to increase the image quality. In astronomy, looking at the sky from a ground telescope produces spread and averaged versions of the observed stars due to the wavefront aberrations produced by the atmospheric turbulences. Furthermore, in the microscopy field, imaging a biological sample also produces aberrated images due to the varying refraction index encountered in the different layers of tissue.

Luckily, this problems can be solved using two different approaches, either in an independent manner or cooperatively. Adaptive Optics (AO) is concerned with the rectification of the incoming aberrated wavefront before it reaches the imaging sensor, modifying the light path such that the image is back on focus. The other method relies on computer algorithms in order to be able to separate the aberration from the original object using as input a set of aberrated images, this is known as Blind Image Deconvolution (BID).

Although AO systems are widely used in high-performance imaging systems, they are also very expensive and, depending on the application, very complex to implement. However, BID methods can be implemented in any system with the only requirement of having a standard computer, making this approach cheaper and easily applicable in most situations.

This thesis is focused on BID methods and its applicability in different cases.

1-1 Motivation

In the Control for Scientific Imaging and Instrumentation (CSI) Group at Delft Center for Systems and Control (DCSC), a novel BID method has been developed, the Tangential Iterative Projections (TIP) algorithm [10]. TIP shows promising results comparing it to other methods due to is fast initial convergence and robustness against noise.

The motivation behind this thesis is to use the novel TIP algorithm along with a non conventional optical design in order to create an optical system with new characteristics that can not be obtained in a standard optical system.

1-2 Objective and approaches

As stated before, the overall objective is to build a fully operational prototype of an non conventional optical system that makes use of the TIP algorithm. As this is very broad, the goal is divided in several objectives with their own approaches:

- Analysis and modification of the TIP algorithm: The idea is to understand how the algorithm works and performs by analyzing its behavior in different conditions and varying tuning parameters. Afterwards, distinct modifications can be proposed in order to increase its robustness and capabilities.
- Design and assembly of the optical system: Knowing the properties of the TIP algorithm, it is possible to think about peculiar modifications of an optical device in order to take advantage of them. The final assembly should allow the easy use of the system in different situations by the user.
- Algorithm implementation: In order to produce an operational prototype that works in a fast manner, the algorithm needs to be implemented efficiently in the computer. The approach is to make an optimized code implementation that uses all the computer resources in order to achieve a computational time of the algorithm of less than one second in the majority of situations.
- Interface development: As the prototype is expected to be easy to use, an interface needs to be developed so the user can operate the prototype in a seamlessly way.
- Testing and experimentation: For ensuring that the algorithm is working as expected, different tests need to be carried out. Firstly it is recommended to simulate the system before building it, in order to make sure that theoretically it could work. Once that the previous step is satisfactory, the prototype can be built and tested in different conditions for checking that the desired characteristics are attained.

1-3 Contributions

After finishing the work of this thesis, achieving the objectives previously explained, several contributions can be outlined:

- Several modifications of the TIP algorithm have been made; being the Super-Resolution (SR) support the most important one.
- In depth analysis on how the main tuning parameters of the algorithm modify the convergence properties and object reconstruction.
- Design of a modular lens holder for a multi-aperture system.
- Development of a Python library, `mklfft`, that allows multi-core FFT computations using the `Intel MKL` library, achieving the fastest FFT implementation in Python for a CPU.

- Development of a Python library, `mkltip`, that uses C-compiled functions and the `mklfft` library for decreasing the computational time of the TIP algorithm run in a CPU.
- Development of a Python library, `gputip`, that allows running the TIP algorithm in a NVIDIA GPU.
- Testing of the multi-aperture system in numerical simulation and in a real environment, achieving the desired properties and a computational time of less than a second in most situations.
- A scientific article is being written with Dean Wilding about the multi-aperture system working along the modified TIP algorithm used in different situations.

1-4 Report structure

This report is divided in several chapters and appendices:

- **Chapter 2**: This chapter for introducing the main factors that decrease the image resolution and quality; describing how to counteract the image aberration, with special emphasis in the use of BID methods.
- **Chapter 3**: How the original TIP algorithm works is analysed in depth.
- **Chapter 4**: Certain modifications of the TIP algorithm are proposed in order to increase its robustness and capabilities (SR support).
- **Chapter 5**: Taking into account the properties of the TIP algorithm a non conventional (multi-aperture) optical system is designed and assembled.
- **Chapter 6**: In order to partially ensure the expected behaviour, convergence properties and performance of the optical system working with the TIP algorithm, numerical simulations are run.
- **Chapter 7**: In this chapter a CPU optimized code of the TIP algorithm is achieved developing the libraries `mklfft` and `mkltip`.
- **Chapter 8**: In this chapter a GPU optimized code of the TIP algorithm is achieved developing the `gputip` library.
- **Chapter 9**: Pre-existent and newly developed libraries are compared among them for the FFT and TIP computations in order to determine their performance for different parameters.
- **Chapter 10**: After assembly the optical system and integrating the optimized code, the performance of the system is tested in different real situation in order to verify the advantages and limits of this methodology.
- **Chapter 11**: In this final chapter the overall conclusions can be found along with future improvements.

- **Appendix A**: As in Chapter 6 there are enough figures for comparing the different parameters, the extra figures created are moved to this appendix.
- **Appendix B**: An introduction to GPU programming useful for Chapter 8.
- **Appendix C**: As in Chapter 9 there are enough graphs for showing the comparison, the rest are moved to this appendix.
- **Appendix D**: In Chapter 10 it is mentioned the use of certain centering algorithms, which are explained in this appendix.

Finally, the different chapters can be grouped as shown in Figure 1-1.

Problem introduction and solving approach	Chapter 1 - Introduction Chapter 2 - Image Degradation and Restoration
TIP analysis and modification	Chapter 3 - The TIP algorithm Chapter 4 - Modifying TIP
Algorithm implementation	Chapter 7 - CPU Implementation Chapter 8 - GPU Implementation Chapter 9 - Computational Performance Analysis
Optical system design and analysis	Chapter 5 - Optical System Design Chapter 6 - Numerical Simulations Chapter 10 - Optical Device Experimentation
Final words	Chapter 11 - Conclusions

Figure 1-1: Grouping of the chapters found in this report.

Chapter 2

Image Degradation and Restoration

Before starting to explain the main body of this master thesis, first it is necessary to introduce the basics in the optics field, how images get aberrated and how it is possible to decrease this aberration. This chapter is divided in the following sections:

- Section 2-1 introduces the diffraction limit, the optical and media aberrations and the mathematical model of an optical system.
- Section 2-2 explains two different approaches for deblurring the images. Adaptive Optics (AO) and Blind Image Deconvolution (BID).
- Section 2-3 introduces several BID techniques ranging from Phase Diversity (PD) and Multi-Frame Blind Deconvolution (MFBD) to Super-Resolution (SR).
- Section 2-4 shows a different optical system compose of a partitioned aperture.

2-1 Image resolution

When taking an image through an optical device, it is always desired to achieve the highest image resolution. However, there are several physical factors that affect negatively the final image.

2-1-1 Diffraction limited systems

The diffraction is a phenomena encountered in light due to is wave-like nature, due to the interference pattern created by the Huygens-Fresnel principle in an aperture. This is a main physical-limiting factor in optical systems that require high magnifications (e.g., telescopes and microscopes) because the interference pattern (i.e., airy disk for a circular aperture) produced on the image sensor impedes the ability to resolve objects that are close of each other.

Figure 2-1 illustrates this phenomena for one and two point-like light sources.

(a) Airy disk.

(b) Superposition of two airy patterns.

Figure 2-1: Maximum optical resolve limited by the diffraction. The left image shows the pattern produced by a point-like light source. The right image shows that this pattern does not allow to differentiate between two points that are too close to each other.

2-1-2 Wavefront aberration

When a point emits or reflects light, it creates a wavefront (only defined for a point-source) that travels through the media. Initially, the shape of this wavefront is a sphere surrounding the point-source but it is usually considered flat if such point is on the optical axis of the imaging system and at a long distance. When this object is recorded by an imaging sensor, the image obtained is of lower quality due to the aberrations induced in the wavefront by the media. The two main sources of aberrations are the optical system itself and the media.

Optical aberrations

Optical aberrations produce image distortion within the system, the main causes are usually imperfect and misaligned components. There are several kinds of aberrations: spherical aberration, astigmatism, coma, pincushion and barrel distortion among others. This kind of aberrations can be corrected to a certain degree by realigning the optical system, developing higher fabrications standards and using corrective optics.

Media aberrations

These aberrations are not produced by the actual optical system but by the media in which the light travels (see Figure 2-2). A very important sub-set of media aberrations for telescopes are the atmospheric turbulences, which are created due to the random and inhomogeneous refractive-index of the air in the atmosphere; disturbing any light beam which propagates through it and finally producing a distortion in the wavefront's shape and intensity. This means that if a collimated beam crosses the atmosphere and arrives to a focus lens, the final quality will be decreased. Another sub-set is the wavefront aberrations in microscopy, produced mainly by the different tissues (which have different refractive index) of the biological sample. This kind of aberration is usually static, due to the nature of the sample, in comparison to the highly dynamic aberrations found in atmospheric turbulences [11].

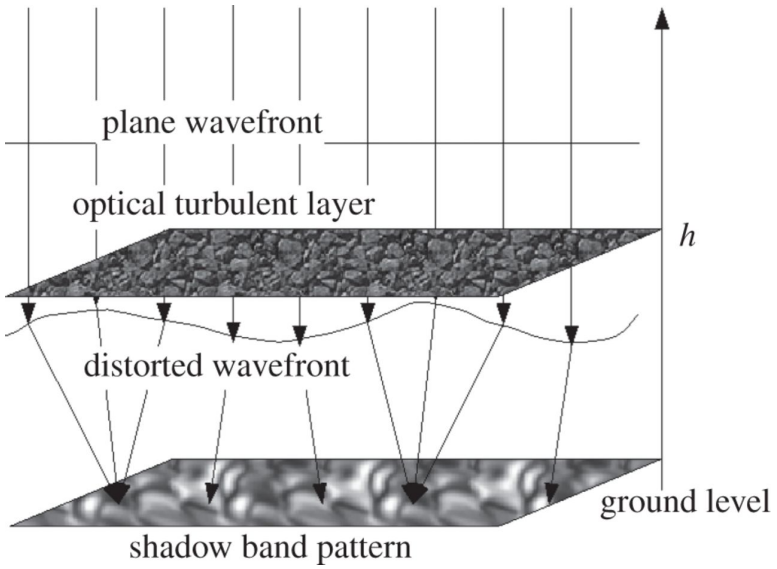


Figure 2-2: Wavefront aberration produced by the atmospheric turbulences. Image credits [1].

Measure aberrations

An incoming wavefront ϕ [rad] can be represented by the sum of a certain basis functions f_i (also known as modal representation):

$$\phi = \sum_i \alpha_i f_i \tag{2-1}$$

In the optics field, this is usually done using Zernike polynomials, represented by Z_n^m (with m and n being different indexes). These polynomials are usually chosen as a basis functions f_i because of their mathematical properties (orthogonality) and easiness to represent certain aberrations. For example, the polynomials Z_1^{-1}, Z_1^1 and Z_2^0 represent a tip, tilt and defocus aberration on the image. Figure 2-3 illustrates the graphical representation of some of the first modes.

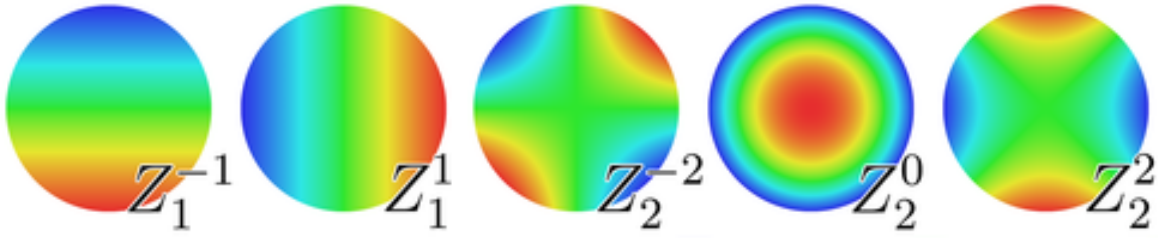


Figure 2-3: Representation of the first five Zernikes without piston. Image credits: Wikipedia.

2-1-3 Early studies by D. L. Fried

D. L. Fried studied in depth how the atmospheric turbulences deteriorate the image recorded in telescopes. Among his several contributions, there are two that are specially interesting in

this field.

Fried parameter

D.L. Fried derived a theory for relating the statistics of wave distortion to optical resolution [12] in two different cases: short-exposure and long-exposure. Taking into account that most of the atmospheric distortion can be treated as a random tilt; in a very-short-exposure this tilt does not reduce the sharpness of an image but displace it. However, in a long-exposure, the randomness of the tilt disturbance spread (average) the image, therefore the final image sharpness is decreased.

The *Fried parameter* r_0 [m] was defined. Having the property that the Root Mean Square (RMS) phase distortion over a circular pupil of diameter r_0 is of one [rad] [2]. Meaning that with the actual aberrations in the optical path and using a telescope of diameter $D > r_0$, the image quality obtained would be similar to the one using a telescope of diameter r_0 with no aberrations/turbulences.

Probability of lucky shot

If an image is taken through a turbulent atmosphere with a sufficiently short-exposure time, there is a possibility that this image will have sharp features (quality near to the diffraction limit of the optical system) because the instantaneous aberration is almost null at that moment [13]. At each instant, a randomly aberrated wavefront arrives to the imaging system, making the average resolution follow $\frac{\lambda}{r_0}$. However, due to the randomness of this disturbance, there is a finite possibility that a short-exposure image shows a resolution close to the diffraction limit $\frac{\lambda}{D}$. In the end, the probability of getting a high quality image is

$$\text{Prob} \approx 5.6 \cdot \exp\left[-0.1557 \cdot \left(\frac{D}{r_0}\right)^2\right]. \tag{2-2}$$

2-1-4 Image formation

Before thinking about how to increase the image quality or reduce the aberrations, it is necessary to introduce a mathematical model in which different solutions can be tested.

Linear system

A basic point is how the light emitted by an object is recorded in a digital system, this is the image formation model. Since modern detectors (i.e., image sensors) are digital, the analysis of the image formation is done in the discrete domain but before, it is necessary to introduce the terminology used in this thesis: $o(x, y)$ is the object intensity at a point (x, y), $h(x, y)$ is the Point Spread Function (PSF) of the optical system and $i(x, y)$ is the intensity perceived by the sensor in a particular point (i.e., pixel of the camera). Assuming that the system is Linear Shift Invariant (LSI) (i.e., every ray of light goes through the same PSF), the image intensity is obtained by means of convolution:

$$i(x, y) = (h * o)(x, y) + w(x, y) \tag{2-3}$$

with $*$ as the convolution operator. It is also noticeable that a noise term $w(x, y)$ has been added in order to model the inherent noise in the image formation process. In Figure 2-4 can be seen the previous process.

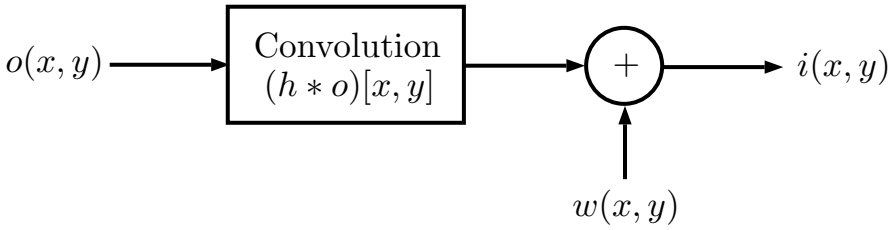


Figure 2-4: Image formation of LSI system.

As the convolution of large discrete time signal can be time consuming, one possibility to decrease this time is to use the Fourier domain via the Fourier transform $\mathcal{F}[\cdot]$ because the convolution reduces to a simple multiplication (see Eq. (2-4)).

$$I(f_x, f_y) = H(f_x, f_y) \cdot O(f_x, f_y) + W(f_x, f_y) \tag{2-4}$$

with $I(\cdot, \cdot)$, $H(\cdot, \cdot)$ and $O(\cdot, \cdot)$ being the original signals in Fourier domain. In addition, this Fourier signals can be quickly obtained using the Fast Fourier Transform (FFT) algorithm.

Expected noise

It is shown that during the convolution of the object with the PSF (see Eq. (2-3)), there is also noise involved. In digital cameras, the principal sources of noise (ADC, temperature) create a Gaussian distribution for each pixel independently. However, there is also another important source of noise in certain conditions, the shot noise.

The shot noise arises due to the discrete nature of light and electric current. When a light beam, or a stream of a discrete number (N) of photons, arrives at a image sensor, it follows a random Poisson distribution with a Signal-to-Noise Ratio (SNR):

$$\text{SNR} = \sqrt{N} \tag{2-5}$$

Therefore when the number of photons is sufficiently large, there is almost no noise contribution. However, in the case of imaging dim objects or taking short exposure images, N can become small enough, thus creating a dominant shot-noise on the image.

2-2 The deblurring problem

Once the reasons of blurred images are understood it is possible to derive methods in order to obtain a better image. There are two main ways to address this issue: using an active corrective optic system or to digitally treat the low quality image. The first method is more related to the use of AO while the second is focused on post-processing the image with algorithms (e.g., BID).

2-2-1 Adaptive optics

There are several articles were the topic of AO is treated. In the book of *Verhaegen et a.l* [2] there is a good introduction which is summarized in the following paragraphs.

The main idea of AO is to cancel the effect of air's turbulence before it arrives to the imaging sensor measuring the wavefront's aberration and using an active optical device to counter such aberration. This is achieved by the use of the *phase conjugation* principle: Reflecting the incoming aberrated wavefront in a mirror with the same shape but half of the amplitude, produces a new reflected flat wavefront. Therefore, the image obtained in the sensor is like the original object. Figure 2-5 illustrates a telescope that, using a Wave-Front Sensor (WFS)

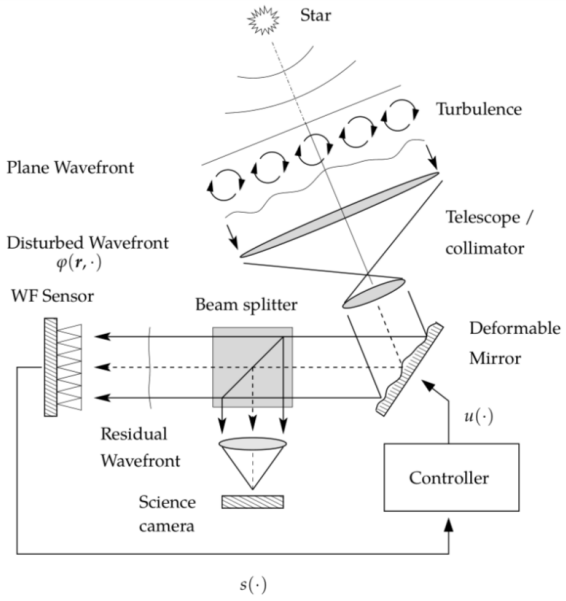


Figure 2-5: Scheme of a telescope with AO. Showing a WFS and a DM. Image credits [2].

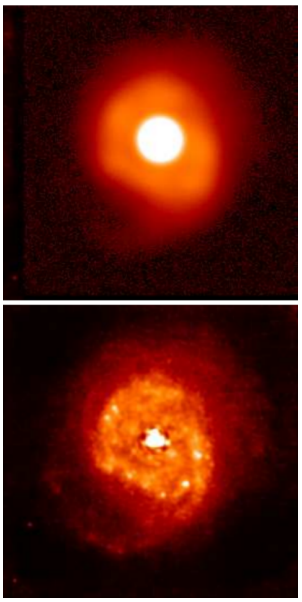


Figure 2-6: Comparison of the same object recorded with the AO system off(top) and on(bottom). The increase in resolution is noticeable. Image credits [2].

for measuring the phase aberration and a controller with a Deformable Mirror (DM), can obtain higher quality images under turbulences. The WFS provides information about the remain phase distortion and feed it to the control loop, this controller is in charge of driving the DM in a way that the WFS receives a flat phase. In the end, the image recorded is less corrupted by the effects of atmospheric perturbations leading to a higher quality image (see Figure 2-6)

Apart from its initial use in astronomical imaging, AO can be used in other fields such as microscopy and industry. As an AO can be used to correct atmospheric disturbances, it can also counteract aberrations in the optical system and to shape the wavefront for having certain desired characteristics.

2-2-2 Blind image deconvolution

BID is a image processing technique used for the deblurring problem for a single observation, which uses an image in order to obtain the PSF and the real object. In this case both PSF and object are unknown, so it is necessary to estimate both of them. This is usually done using an Alternating Minimization (AM) algorithm but this does not always solve the problem because trivial solutions may be obtained. Nevertheless, it is possible to use deconvolution to estimate the object if the blur is known but, in most applications this can not be hold true because the atmospheric turbulences and optics aberrations can not be known accurately. Thus, in most cases the only way to obtain a sharp image is by BID.

Inverse problems (such as BID) are usually difficult to solve and, typically, ill-posed. A problem is well-posed [14] when: a solution exists, it is unique and it is stable under perturbations. While the first conditions mean that the inverse exists, the last one requires the continuity of the inverse function. In addition, this third condition is the most difficult one to solve because there are no mathematical tools to convert an unstable problem to a stable one; however, it is usually useful to solve an approximated problem with regularizers.

The mere fact of deconvolving the image with a known PSF (which usually acts as a low-pass filter) is difficult because it is an inverse problem. If an estimate of $O(f_x, f_y)$ is required, it can be obtained using the inverse PSF in Eq. (2-4):

$$\hat{O}(f_x, f_y) = H^{-1}(f_x, f_y) \cdot I(f_x, f_y) \tag{2-6}$$

$$= O(f_x, f_y) + H^{-1}(f_x, f_y) \cdot W(f_x, f_y) \tag{2-7}$$

Now, the inverse PSF acts as a high-pass filter, meaning that the noise is going to be increased at high spatial frequencies, leading to a low quality object estimate. However there are techniques that can improve the result, such as: Wiener filtering, Least Squares (LS) estimation and Richardson-Lucy algorithm.

On the other hand, the problem of BID is even more difficult to solve because the PSF is not known. This problem is highly nonlinear and an infinite number of solutions can be found due to the fact that there is only one variable available ($I(f_x, f_y)$) and two variables are estimated ($H(f_x, f_y)$ and $O(f_x, f_y)$).

Bilinear problems

BID is a bilinear problem, as it is going to be showed. Assuming that the object and PSF are unknowns, the image formation can be defined as the following mapping

$$\mathcal{H} : \mathbf{H} \times \mathbf{O} \rightarrow \mathbf{I}$$

Then, the noiseless case of image formation is

$$\begin{aligned} i &= \mathcal{H}(h, o) \\ &= h \cdot o \end{aligned}$$

If one of the variables is kept constant, then there is a linear relation between the operator $\mathcal{H}$ and the other variable. If o is constant then

$$\begin{aligned}\mathcal{H}(h_1 + h_2, o) &= (h_1 + h_2) \cdot o \\ &= h_1 \cdot o + h_2 \cdot o \\ &= \mathcal{H}(h_1, o) + \mathcal{H}(h_2, o)\end{aligned}$$

as the previous argument holds true as well when h is held constant; the BID is proved to be a bilinear problem in which keeping one variable fixed transform the problem to a linear ill-posed one. Finally, solving both linear coupled problems are equal to solving the BID problem.

2-3 Blind deconvolution methods

In this section, the ideas and properties behind BID algorithms are going to be introduced because in this thesis this kind of algorithm is used for solving the deblurring problem.

2-3-1 Phase retrieval and phase diversity

"Wavefront sensing by phrase retrieval implies extraction of the Fourier transform of a complex signal based on observation of the modulus of the signal" [15]. This means that in order to estimate the phase aberrations, only the image intensity on the focal plane is needed. This method can be used along an AO system because the phase of the wavefront is required for the control algorithm or in a BID scheme. What is more, the addition of a PD can be used to obtain a joint estimate of object and phase aberration [15] [16] [17]. The effects produced by both optical and media aberrations can be modelled by a distorted wavefront (or phase) $\theta(x, y)$.

Phase retrieval concept

As in the article [15]; having a monochromatic point object, the imaged PSF $h(x)$ is related to the coherent system function $p(x)$ by:

$$h(x) = |p(x)|^2 \tag{2-8}$$

and $P(f)$ given by the Fourier transform of $p(x)$:

$$P(f) = \int_{-\infty}^{\infty} p(x) e^{-i2\pi f_x} dx \tag{2-9}$$

with (f_x, f_y) as the spatial frequency terms. This leads to the following expression:

$$P(f) = A(f) e^{i\theta(f)} \tag{2-10}$$

with $A(f)$ being the pupil function. Now, given the previous equations, we can state that the phase retrieval concept is, based on the measurements of $h(x)$, estimate $\theta(f)$. This can

be achieved using a LS algorithm for the minimization of a mean-square error cost function $J(x)$ which uses the actual measurement $h(x)$ and its estimate $\hat{h}(x)$:

$$J(x) = \|h(x) - \hat{h}(x)\|_2^2 \tag{2-11}$$

Nevertheless, there are more algorithms available, like the Gerchberg-Saxton algorithm and others that use a Maximum Likelihood Estimator (MLE).

Phase diversity

An object $o(x)$ emitting incoherent light will be measured by an optical system as:

$$i_1(x) = h_1(x) * o(x) + w_1(x) \tag{2-12}$$

with image measured $i(x)$, spatial convolution operator $*$ and noise $w(x)$. If the optical system allows the addition of a PD $\phi(f)$ (e.g., defocus) into the wavefront and assuming that $\theta(f)$ is static between measurements; a second image $i_2(x)$ is obtained which holds the information of the actual phase $\theta(f)$ and the known phase diversity $\phi(f)$, allowing the joint estimate of $o(x)$ and $\theta(f)$ [15]. If there are several images available, it is possible to fuse them [17] using a MLE frame. Then, the estimation of the object $\hat{O}(f)$ is calculated using the previous information as follows:

$$\hat{O}(f) = \frac{\sum_k I_k(f) H_k^*(f)}{\sum_k H_k(f) H_k^*(f) + \beta(f)} \tag{2-13}$$

with a noise regularization parameter $\beta(f)$, used to avoid noise amplification at certain frequencies.

Some previous approaches for fast algorithms used a regularized variant of the Gauss-Newton optimization method using a likelihood criterion (see Vogel et al. [18]). As for the joint estimation, Van Noort et al. [19] derived a Joint Phase Diverse Speckle image restoration used mainly for astronomy purposes. Finally, a recent development in analytical PSF done by Ramos et al. [20], allows the use of Zernike polynomials directly in the wavefront estimation, introducing a faster PD reconstruction algorithm.

2-3-2 Blind Deconvolution Methods

The BID methods can be classified by the time they were developed and if they use deterministic or statistical techniques (see book [14]). Most methods treat the images as signals and use iterative algorithms for finding the minimum of a cost function (or the increase of a likelihood). Other methods uses a state-space representation with a Kalman filter (see Zhang et al. [21]) or an ARMA model for subspace identification (see Yu et al. [22]). There are also some methods that allows a close loop solution of the problem, these are also known as direct methods because they are not iterative. In the article of Yitzhaky et al. [23], a comparison of different direct methods can be found.

Iterative blind deconvolution

Ayers et al. [3] developed an algorithm for solving the deblurring problem, known as Iterative Blind Deconvolution (IBD) that has been very used and modified since then. It assumes some a priori knowledge of the functions ($h(x,y)$ and $o(x,y)$)to find, like that they are nonnegative (a real PSF and Object do not show negative intensity measurements), and try to deconvolve them using in the Fourier domain. The general scheme of this algorithm is illustrated on Figure 2-7. It is also important to note that this scheme is analogous to other algorithms (e.g., Gerchber-Saxton). This algorithm can be seen as an AM scheme in which in each step,

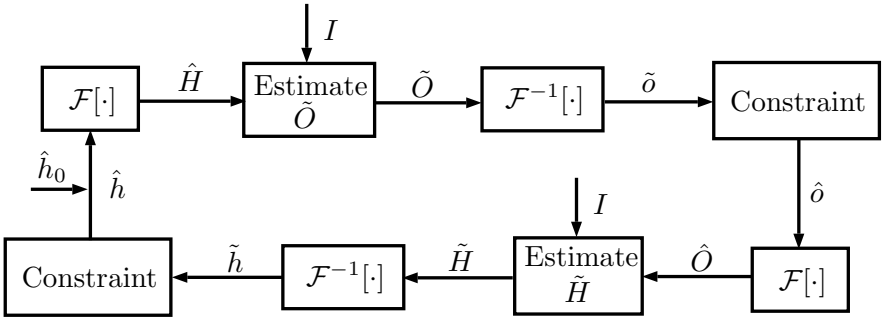


Figure 2-7: Iterative Blind Deconvolution algorithm.

one of the two variables is set constant in order to further estimate the other and so on. In this case, the estimation is done in the Fourier domain while the image constraints are imposed in time domain (the nonnegativity assumption is done setting to zero those values that are negative).

Maximum likelihood estimation

Once the model of image formation has been selected (the convolution of the PSF with the Object plus noise) one possible way to find (or estimate) the parameters is using the widely known MLE, which is a standard approach for parameter estimation and inference [24].

The MLE has very interesting optimal properties, such as: convergency (for sufficiently large data samples the estimation converges asymptotically to the true value), efficiency (lowest variance of the estimated parameter) and parametrization invariance (same solution found independently of the parametrization model). In addition, this method is a prerequisite for: chi-square test, Bayesian methods, inference with incomplete data and stochastic modelling among others.

Having a data vector (or array) from the image $y = (y_1, \ldots, y_m)$, the goal is to find the population (PSF and Object) parameters that most likely have generated such data. From a statistically standpoint, the population corresponds to a probability distribution which is associated to a value of the model's parameter. With $f(y|w)$ being the Probability Density Function (PDF) that indicates the probability of obtaining y for a parameter w. If the different y_i's are statistically independents, the the PDF for y can be written as

$$f(y = ((y_1, \ldots, y_m)|w) = f_1(y_1|w) \times \cdots \times f_m(y_m|w) = \prod_{i=1}^{m} f_i(y_i|w) \tag{2-14}$$

For a given set of parameter values, the PDF illustrates which data is more probable but the current problem is the inverse, to find the most probably parameters that have produced such date. Therefore, the likelihood function is defined for such purpose:

$$\mathcal{L}(w|y) = f(y|w) \tag{2-15}$$

The idea of the MLE is finding the set w that "most likely" produces the observed data y, therefore it is necessary to maximize the likelihood function. Usually, the log-likelihood function $\ln \mathcal{L}(w|y)$ is used for computational easiness. When a maxima has been found it must satisfy that the derivative is null (so it is a maxima o minima):

$$\frac{\partial \ln \mathcal{L}(w|y)}{\partial w_i} = 0 \tag{2-16}$$

and the second derivative is negative (so it is a maxima):

$$\frac{\partial^2 \ln \mathcal{L}(w|y)}{\partial w_i^2} < 0 \tag{2-17}$$

Richardson-Lucy algorithm

The Richardson-Lucy (RL) algorithm [25] is very popular one in the fields of astronomy and medical imaging. Derived from the Bayes theorem, it comes from a MLE framework and was also found by use of the Expectation Maximization (EM) algorithm.

The reason of the popularity of this algorithm is due to the high quality image reconstruction even under high levels of noise. In addition, the results can be improved incorporating certain a priori knowledge by assuming a functional PSF [4]. As we are dealing with a BID problem, this algorithm is can only used when the PSF is known. The Blind-Deconvolution scheme proposed using this algorithm is showed in Figure 2-8. Finally, the final iterative algorithm

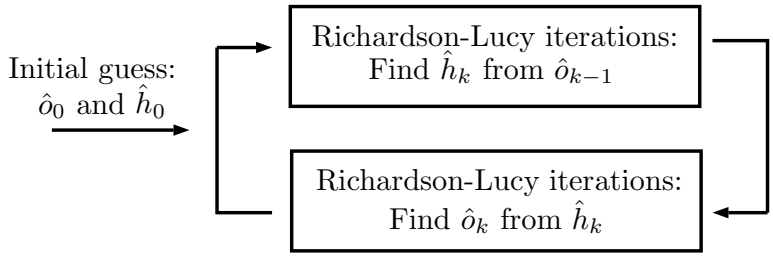


Figure 2-8: Richardson-Lucy scheme for solving the BID problem.

assuming isoplanatic conditions is

$$h_{k+1}^n = \left\{ \left[\frac{i(x)}{h_k^n(x) * o^{n-1}(x)} \right] * o^{n-1}(-x) \right\} h_k^n(x) \tag{2-18}$$

$$o_{k+1}^n = \left\{ \left[\frac{i(x)}{o_k^n(x) * h^n(x)} \right] * h^n(-x) \right\} o_k^n(x) \tag{2-19}$$

While the convergence of the RL is well known for a non-blind deconvolution problem, the convergence of this double scheme for BID is not guaranteed [26]. The addition of some prior PSF constraints may help to avoid the trivial solution, but it was found that this destroys the underlying property of monotonicity, therefore not ensuring convergence.

Maximum a posteriory estimation

Trying to remove the camera shake on single images, it has reformulated the BID problems by means of a Maximum A Posteriori (MAP), knowing that images have a heavy tailed distribution on their gradients. This approach makes a joint estimation of both PDF and o, also known as the $\text{MAP}_{h,o}$. However, the direct application of the MAP method was reported to fail; the explanation of this behaviour and a possible solution is treated in [27].

Being $\mathcal{P}(h, o|i)$ the posterior probability of p and o when i happens (or it is observed). In addition, the prior densities of the unknowns are $\mathcal{P}(h)$ and $\mathcal{P}(o)$. Using MAP for a joint estimation, the maximization of $\mathcal{P}(h, o|i)$ is done by the pair (h, o), leading to the posteriori probability

$$\mathcal{P}(h, o|i) \propto \mathcal{P}(i|h, o)\mathcal{P}(h)\mathcal{P}(o) \tag{2-20}$$

For simplicity, a uniform prior on h is assumed while the prior of o, which favors natural images, is based on sparse gradient distributions with the following measurement metric (also known as regularizer)

$$\log \mathcal{P}(o) = -\sum_{k}(|g_{u,k}(o)|^{\alpha} + |g_{v,k}(o)|^{\alpha}) + C \tag{2-21}$$

$g_{u,k}(o)$ and $g_{v,k}(o)$ are the horizontal and vertical gradients at pixel k, C and α are a normalizing constant and a sparsity parameter respectively. In addition, the likelihood of $\mathcal{P}(i|h, o)$ can be fitted with

$$\log \mathcal{P}(i|h, o) = -\lambda \|h * o - i\|^2 \tag{2-22}$$

Minimizing the MAP allows to obtain the optimum value for $(\hat{h}, \hat{o})$ (assuming that the PSF have an uniform distribution)

$$(\hat{h}, \hat{o}) = \arg\min_{h,o} \left(\|h * o - i\|^2 + \lambda_o \sum_{k}(|g_{u,k}(o)|^{\alpha} + |g_{v,k}(o)|^{\alpha}) \right) \tag{2-23}$$

where λ_o is the regularization factor for the image.

Deconvolution using image priors

One possibility of overcoming the problems of BID is introducing some image priors as explained in Levin et al. [5]. The idea is using some priors that can ease the computation of the optimization algorithm by restricting the set of all possible images to a more natural ones. This can be posed in the Fourier domain, giving as a result:

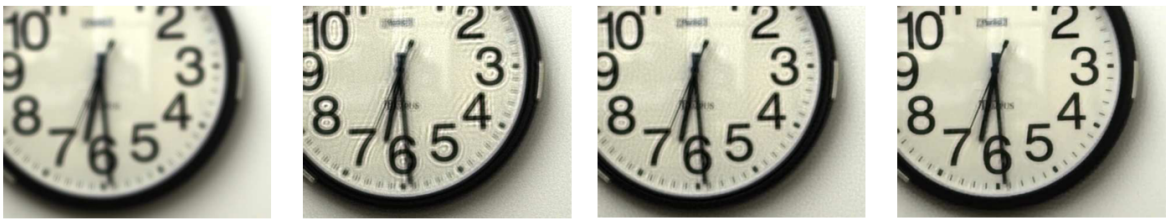
$$O(f_x, f_y) = \frac{H(f_x, f_y) \cdot I(f_x, f_y)}{|H(f_x, f_y)|^2 + w \sum_k |G_k(f_x, f_y)|^2} \tag{2-24}$$

It is remarkable that when no Gaussian prior is used ($w = 0$), the solution will be equal to dividing $I(f_x, f_y)$ by $H(f_x, f_y)$. It is also observable that the effect of this filter is larger in higher frequencies, where the noise can affect the result by a large degree.

Although Gaussian priors can be useful, the distribution of derivative filters in natural images is sparse instead of Gaussian. This approach obtains a better result but the optimization

problem is no longer convex, therefore a iterative algorithm is necessary. The Iterative Reweighted Least Squares (IRLS) is used to solve the system, achieving a better result.

Figure 2-9 shows a comparison between the results obtained by different methods. It is remarkable how the RL suffers from ringing and the the sparsity prior obtains the most natural solution.



(a) Captured image (b) Richardson-Lucy (c) Gaussian prior (d) Sparsity prior

Figure 2-9: Comparison between different IBD algorithms and priors. Image credits [5].

Although the use of sparse prior is the most suitable for BID, it is usually very computational extensive. In order to increase the performance, Khrishnan et al. [28] derived a faster algorithm which made use of lookup tables in order to speed certain calculations.

Convex programming

The BID is a highly nonlinear and nonconvex problem, meaning that the solutions obtained could be local minimas. However, recent work on relaxation conditions by Ahmed et al. [29], showed that it was possible to retrieve two unknown signals, w and x, observing the product of their convolution when the problem was relaxed as a Semi-Definite Program (SDP) with a nuclear norm.

2-3-3 Multiframe Blind Deconvolution

The BID is an ill-posed problem for which most methods suffer from convergence and stability issues. Another more stable way to obtain the recorded object is to take several aberrated images of the same object and try to fit it to each PSF, this is known as MFBD. There are in general two different multichannel models: Single-Input Multiple-Output (SIMO) (also known as *multiframe model*) and Multiple-Input Multiple-Output (MIMO). A typical optical system with only one imaging system is a SIMO model in which each channel is for each picture taken with different aberrations (see Figure 2-10). The imaging formation model is an extension of Eq. (2-4) with a total number of frames K (or images taken):

$$i_k(x,y) = (h_k * o)(x,y) + w_k(x,y)\ , \qquad k = 1,\dots,K \tag{2-25}$$

MFBD by incremental EM

One possible way to solve the MFBD is by using incremental EM as in Harmeling et al. [30]. In this method, there is no need to define image priors, which increase the simplicity, because there are several frames available, making the optimization system better determined.

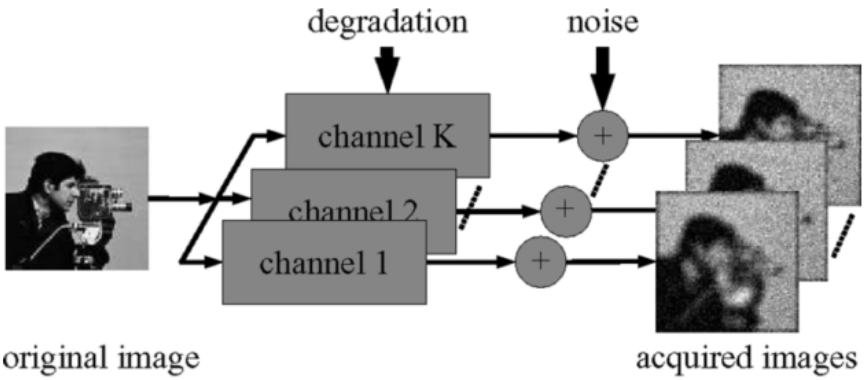


Figure 2-10: SIMO model scheme. Image credits [6].

From a statistical point of view, the probabilistic image model is:

$$\mathcal{P}(i_1, \ldots, i_K|o) = \prod_{k=1}^{K} \int \mathcal{P}(i_k, h_k|o) dp_k \tag{2-26}$$

which is the MLE. In addition, Jensen's inequality allows to bound this equation from below.

$$\sum_{k=1}^{K} \log \int \mathcal{P}(i_k, h_k|o) dh_k \geq \sum_{k=1}^{K} \int q_k(h_k) \log \frac{\mathcal{P}(i_k, h_k|o)}{q_k(h_k)} dh_k \tag{2-27}$$

Assuming that the convolution is corrupted by Gaussian noise, the optimization problem yields a non-negative LS, solved using the generalized EM updates:

- E-step: find $h_k \geq 0$ that minimizes $\|i_k - h_k * o_{k-1}\|_2^2$
- M-step: find $o_k \geq 0$ such that $\|i_k - h_k * o_k\|_2^2 \leq \|i_k - h_k * o_{k-1}\|_2^2$

The EM method has been used for astronomical imaging giving good results (see Schulz et al. [31] and Schulz [32]).

MFBD by tangential iterative projections

One of the most recent articles about MFBD is by Wilding et al. [10], where an improvement on the actual techniques is achieved by the Tangential Iterative Projections (TIP) , showing the optimum denoising. This thesis is based on that algorithm and it is going to be explained in depth in Chapter 3.

In the original article, this algorithm is compared with other MFBD algorithms, the most notable advantage is the robustness to noise (due to the use of a LS method), the solution smoothness (due to the physical constraints) and implementation speed (faster than a MLE algorithm). The LS method with TIP produces an optimum denoising making a better estimate of the object in most low light applications.

2-3-4 Super-Resolution

The term digital SR is a technique that computationally extend the image resolution beyond the pixel spacing (but can not go show details beyond the optical limit). For example, Pauca et al. [33] developed a multilenslet camera with different filters and aberrations in order to obtain a SR image. In imaging applications where media aberration play a determinant role in the performance of the optical system, applying digital SR methods can increase the final image quality (see Gerwe et al. [34]).

Blind superresolution

Usually, the problem of MFBD and SR are solved independently. However, Sroubek et al. [7] developed a new unified approach for both problems. The main idea is to increase the image formation model with a decimation operator D in order to model the function of the image sensor; this means assuming that the object is a High Resolution (HR) image which resolution is decreased by the optical system, therefore a Low Resolution (LR) image is obtained. Keeping that in mind, it is possible to compute an algorithm to recover a HR image from a set of LR ones. The new image formation model is shown on Equation (2-28).

$$i_k(x,y) = D \cdot (h * o)(x,y) + w_k(x,y) \tag{2-28}$$

Before developing an algorithm, it is necessary to calculate a cost function. After a thorough mathematical derivation, the complete cost function was obtained:

$$J(H,O) = \sum_{k=1}^{K} \|DH_kO - I_k\|_2^2 + \alpha Q(O) + \beta R(H) \tag{2-29}$$

with the regularization term $Q(O)$ of the object (tries to obtain a smoother version of the object) and the regularization term $R(H)$ of the PSF (a feasible and smooth PSF is desired). In order to minimize this function, the common AM (a variation of the steepest descent) is used with the following steps:

- Step 1: Fix p then $O^m = \sum_{k=1}^{K} H_k^T D^T I_k$.
- Step 2: Fix o then $H^{m+1} = O^T D^T I$

It is also possible to improve the convergence by adding the standard constraints for the HR image and the PSF. In order to compare the performance of this algorithm with other methods available, several LR images are taken and a HR estimation is reconstructed; this can be observe on Figure 2-11.

Figure 2-11: After taking eight LR images of a car, several HR images are reconstructed: left, bilinear interpolation; middle, BSR estimate of the previous algorithm; right, new image with optical zoom. Image credits [7].

2-4 Multi-aperture systems

Doing high-resolution imaging through a turbulent atmosphere can be done thanks to speckle imaging techniques. This is achieved taking and post-processing several short-exposure images in which the atmospheric turbulence is frozen (and not averaged). One way to collect these images is in a stream fashion, meaning that only one imaging sensor takes pictures; however, there is also te possibility of taking several pictures at once using a certain number of apertures imaging the same object. This last possibility is known as multi-aperture or pupil segmentation imaging. Figure 2-12 illustrates different configuration possibles for multi-aperture imaging systems. Loktev et al. [8] developed an adaptable multiaperture imaging

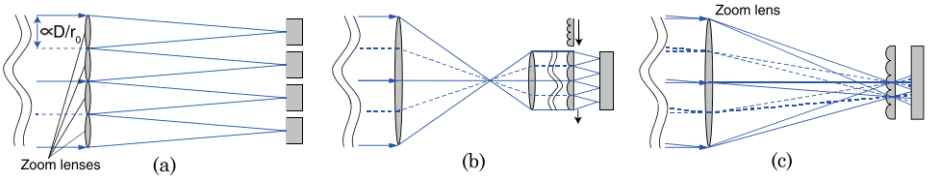


Figure 2-12: *(a)* Direct pupil sampling, *(b)* pupil image sampling and *(c)* plenoptic imaging schemes. Image credits [8].

system for horizontal turbulence path in which the resolution was improved by setting the size of the subaperture to the r_0 and by applying MFBD techniques to the images recorded simultaneously by each subaperture. The set-up used is of the same type as the one shown in Figure 2-12b. After the image acquisition, the algorithm used to estimate the object is similar to AM with the standard nonnegativity constraint along some extra steps: rejection of the less sharp subimages according to the metric $S = \sum I^2/(\sum I)^2$ (I is the image intensity); subimages aligned by means of cross correlation and suppression of edge-ringing artifacts by means of apodization. Figure 2-13 illustrates the result obtained applying the multiaperture method while imaging an object at 550 m.

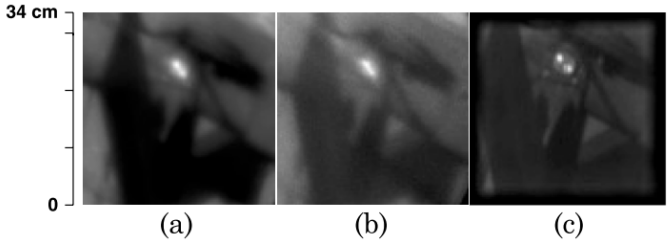


Figure 2-13: *(a)* Average, *(b)* sharpest, *(c)* estimated. Image credits [8].

2-5 Summary

In this chapter an introduction to the deblurring problem and has been made, explaining how images are aberrated and the two main approaches to solve this problem: AO or post-processing techniques. As this thesis is focused on the post-processing techniques, some of them have been introduced. In addition, SR has been explained due to the applications in MFBD algorithms and the possibility of using it along a multi-aperture system.

Chapter 3

The TIP Algorithm

In this chapter, the Tangential Iterative Projections (TIP) algorithm, developed by Wilding et al. [10], is going to be explained in depth in order to gain a good insight in how it works from the inside.

Moreover, Figure 3-1 shows the images that are going to be used throughout the whole report for testing the reconstruction performance of the algorithm. It is observable that one image is binary with very sharp edges while the other it is the well-known *Lenna* picture, used in gray scale, seen in uncountable computer vision articles.

(a) USAF Target. **(b)** Lenna.

Figure 3-1: Different images used for comparison purposes through this thesis.

3-1 How it works

The TIP algorithm makes use of an Alternating Minimization (AM) scheme by means of the Least Squares (LS) method in the Fourier domain. Therefore the imaging formation model used is the one in Eq. (2-25) but in the frequency domain:

$$I_k(f_x, f_y) = H_k(f_x, f_y) \cdot O(f_x, f_y) + W_k(f_x, f_y) \,, \qquad k = 1, \ldots, K \tag{3-1}$$

As it can be observed, the idea is to have several images I_k with different Optical Transfer Function (OTF) H_k of the same object O. This is used in order to decrease the difficulty of the optimization (recall that this is a very ill-posed problem). The idea of using different images is that they carry different information of the object because each PSF modify them in a different way, therefore, although each image is aberrated, the overall information carried in the whole set maybe enough the retrieve a good estimate of the object.

Having that in mind, the next step is to fuse that information so both PSFs and object can be retrieved. This can be achieved using a LS approach

$$\left\{\hat{H}_k,\ \hat{O}\right\} = \arg\min_{H_k,O} \sum_k \|I_k - H_k \cdot O\| \tag{3-2}$$

in which a better estimate of $\hat{H}_k$ and $\hat{O}$ is obtained alternatively. The AM scheme works by first fixing one of the two variables (either H_k or O) in order to estimate the other and then the same is done but interchanging the variables.

The TIP algorithm starts with the estimation of the object $\hat{O}^{n+1}$ first using the solution of the LS:

$$\hat{O}^{n+1} = \frac{\sum_{k=1}^{K} \hat{H}_k^{*n} I_k}{\sum_{k=1}^{K} |\hat{H}_k^n|^2}. \tag{3-3}$$

With the new object estimation, the next step is to obtain an estimate of each OTF. However, instead of using the LS solution for finding each $\hat{H}_k$, this algorithms uses a *tangential projection* (TIP belongs to the Projection Onto Convex Sets (POCS) family) which is

$$\tilde{H}_k^{n+1} = \frac{I_k}{\hat{O}^{n+1}} \tag{3-4}$$

into the affine space. However, as the posed optimization problem is unconstrained, the solution obtained so far could be physically unfeasible (e.g., a negative intensity value in the PSFs). In order to solve this, a projection $\mathcal{P}_{\mathcal{H}}$ into the feasible solution set is used for constraining $\tilde{H}_k^{n+1}$. Then $\hat{H}_k^{n+1}$ is calculated as follows:

$$\hat{H}_k^{n+1} = \mathcal{P}_{\mathcal{H}}\left[\tilde{H}_k^{n+1}\right] \tag{3-5}$$

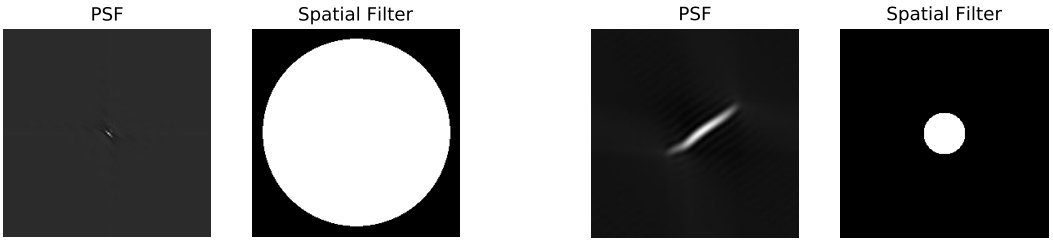
Now, Eq. (3-3), Eq. (3-4) and Eq. (3-5) are repeated for a certain number of iterations or until a certain optimization criteria is met. After this, a last estimation of the object is done with Eq. (3-3). But, before it is returned, the estimation is not constrained so it needs to be projected in the feasible set of solutions with the operator $\mathcal{P}_{\mathcal{O}}$

$$\hat{O}^{N+1} = \mathcal{P}_{\mathcal{O}}\left[\hat{O}^{N}\right] \tag{3-6}$$

3-2 Constraints

In the previous section several projections are used for constraining the solutions found via optimization. There are several kinds of constraints that could be applied as long as they are convex (so the convergence of the optimization is still guaranteed). The main projections used for the PSF and object are:

- $\mathcal{P}_{\mathbf{f}} : H(\mathbf{f} \notin \mathbb{S}) = 0$. Assuming that the PSF is smaller than the object, this also reduces its high frequency content (it is bandwidth limited), therefore those frequencies that are not in the set $\mathbb{S}$, their value is set 0. See Figure 3-2 for a comparison between different frequency sets. This constraint is useful when the estimated PSFs are too small, producing a low quality object estimation; decreasing the frequency bandwidth produces an increase in size of the estimated PSFs.



(a) High bandwidth of PSF. **(b)** Low bandwidth of PSF.

Figure 3-2: Comparison of the two PSFs obtained by TIP when the spatial filter of the PSF cut-offs at higher or lower frequencies. Observe how the PSF grows larger when only low frequencies are allowed.

- $\mathcal{P}_{\geq\alpha} : h(h < \alpha) = 0,\ \alpha \geq 0$. As the estimation of the PSF maybe noisy, this constraint only lets the values higher than α to be significant and also it removes the negative values. In addition, this constraint is very important in order to keep the convergence properties of the algorithm (because it creates high-frequency values that helps retrieving more information from the images); it should be noted that a too small value of α makes the algorithm to diverge.
- $\mathcal{P}_{\geq 0} : o(o < 0) = 0$. This ensures that there is no negative values in the estimated object because it is physically unfeasible by definition. Figure 3-3 illustrates the convergence and divergence of the algorithm for two different α values.

(a) Object estimated with $\alpha = 0.20$. **(b)** Object estimated with $\alpha = 0.00$.

Figure 3-3: Comparison of different α values. When it is too low, the algorithm diverges.

There are others constraints needed due to numerical inaccuracies of the FFT and also for

keeping the variables normalized during the iterative process.

- $\mathcal{P}_{\text{Re}} : x = \text{Re}\left[\mathcal{F}^{-1}[X]\right]$. When the PSF/object is calculated from its frequency domain counterpart, only the real part of the inverse Fourier transform is taken. This is done in order to ensure that there are no complex values when only reals are physically possible.
- $\mathcal{P}_{\text{Norm}} : x = \frac{x}{\max(x)}$. This is a form of normalization in the spatial domain which constraints the maximum value of the image to 1.

3-3 Summary and results

In this chapter, the TIP algorithm has been analyzed in depth. It works in both frequency and spatial domain; in the former, the estimation of the object and the OTFs is performed doing a LS and tangential projection respectively, following an iterative AM scheme; the spatial domain is basically used for imposing the constraints, in order to keep the solutions physically realistic.

Regarding the constraints, the non-nengativity ,$\mathcal{P}_{\geq 0}$, and the thresholding for the PSF, $\mathcal{P}_{\geq \alpha}$, are basic for keeping the solution feasible. In addition, the latter is key for ensuring the convergence of the algorithm as it eliminates low noisy values of the PSFs and creates high-frequencies (thresholding produces sharp edges that in the Fourier domain represents high-frequency terms) that helps estimating the object in the following iteration by extracting more information from the input set.

One actual limitation of this method is that it does not support Super-Resolution (SR) but it can be easily extended, this is done in the following chapter.

Figure 3-4 illustrates how the algorithm works when 4 aberrated images of the same object are fed to it. It is observable that from a set of distorted images, the algorithm manages to fuse all information available in order to reconstruct the underlying object.

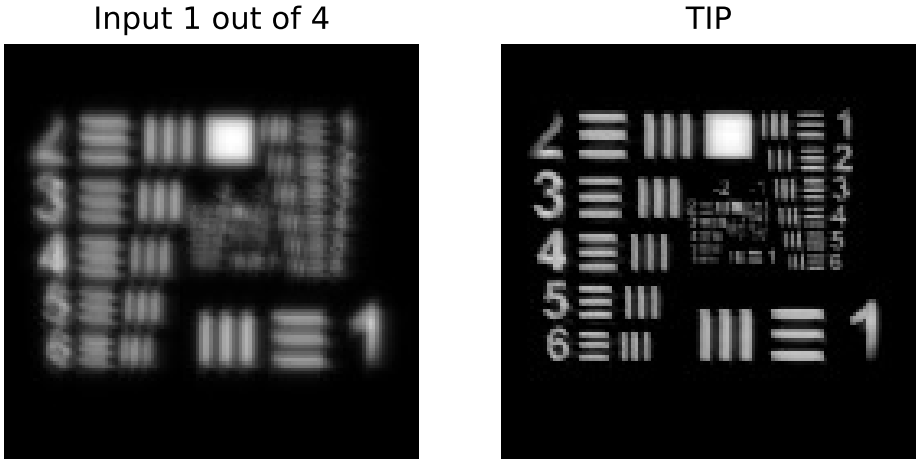


Figure 3-4: The left image is one of the input images fed to the algorithm and the right one is the reconstruction obtained after a few iterations.

After the previous explanations, the pseudocode of the TIP algorithm (see [10]) is shown in Algorithm 1:

Algorithm 1 Reconstruct the object using the orignal TIP

procedure TIP(i_k, N) ▷ i_k is a stack of K blurred images.
  Initialization:
  $H_k^0 \leftarrow \mathcal{F}(\delta)$ ▷ The PSFs are initialized as delta functions.
  $I_k \leftarrow \mathcal{F}(i_k)$
  Main Loop:
  for $n \leftarrow 0, N$ **do**
    $\hat{O}^{n+1} \leftarrow \frac{\sum_{k=1}^{K} \hat{H}_k^{*n} I_k}{\sum_{k=1}^{K} |\hat{H}_k^n|^2}$
    for $n \leftarrow 1, K$ **do**
      $\tilde{H}_k^{n+1} \leftarrow \frac{I_k}{\hat{O}^{n+1}}$
      $\hat{H}_k^{n+1} \leftarrow \mathcal{P}_{\mathcal{H}} \left[\tilde{H}_k^{n+1}\right]$
    end for
  end for
  Final Reconstruction:
  $\hat{h}_k \leftarrow \mathcal{P}_H \left[\mathcal{F}^{-1}(H_k^N)\right]$
  $\hat{O} \leftarrow \frac{\sum_{k=1}^{K} \hat{H}_k^{*N} I_k}{\sum_{k=1}^{K} |\hat{H}_k^N|^2}$
  $\hat{o} \leftarrow \mathcal{P}_O \left[\mathcal{F}^{-1}(\hat{O})\right]$
  return $\hat{h}_k, \hat{o}$ ▷ $\hat{h}_k$ is a stack of K PSFs.
end procedure

Chapter 4

Modifying TIP

In this chapter, the TIP algorithm is modified in order to achieve Super-Resolution (SR) and increase the convergence speed and robustness in certain conditions.

4-1 Extension to SR

4-1-1 How SR works?

It may be hard to visualize how it is possible to retrieve information from higher frequencies (or smaller pixels) when only information from a Low Resolution (LR) source is available. The main idea is to use the information provided by the sub-pixel displacement of the object in different images [9]. In addition, it is assumed that several images of the same object produced by different aberrations are available because, if not, there is no extra information that the SR scheme can retrieve.

The light coming from small details of an image (e.g., edges) is averaged in the pixels of the camera, therefore, those details are lost. However, if another image is taken with a small displacement with respect to the previous one, the incoming light is still averaged but it reaches different pixels. Then, if enough of this slightly displaced images are taken, it would be possible to estimate details smaller than the actual pixel size. This can be easily seen in Figure 4-1.

The question arises on how to obtain such images. Luckily, when we are looking through the atmosphere, turbulences produce different kind of aberrations, two of them are *tip* and *tilt* which precisely produce the effect desired: vertical and horizontal displacements of the incoming image. However, if the optical system is a microscope imaging a static sample (which produces static aberrations) this technique can not be used without modifying the system itself. In this case, segmenting the pupil in several apertures allows the retrieval of several images from different perspectives, this means that each image is produced by a different aberrations when the light goes through the sample.

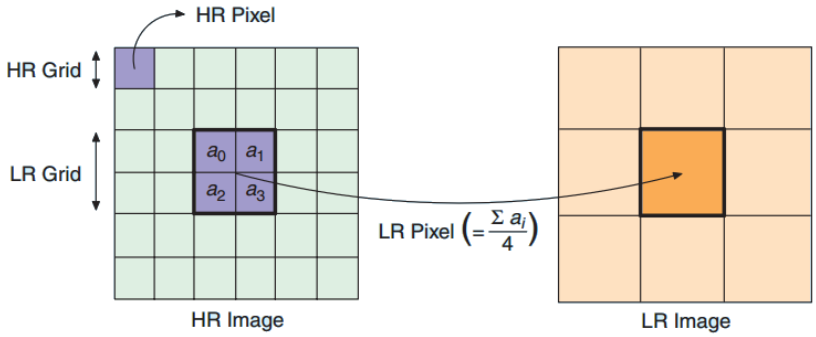


Figure 4-1: Averaging of small feature in LR pixels. Image credits [9]

Limits of SR There are different SR techniques, each one of them with different limiting factors. In this case, as the SR is achieved fusing the sup-pixel displacement information carried by a set of low-resolution images, the maximum image quality (or information) is limited by the maximum information that arrives to the imaging sensor; as the aperture of the optical system acts as a low-pass filter (thus, creating the airy pattern in a circular aperture), the maximum resolve capability is still limited by the diffraction limit. Therefore, in the case in which the input images set carries all the information possible, the best reconstruction with a very high resolution would not be better than a diffraction limited image.

4-1-2 Including SR in TIP

An interesting addition to the TIP algorithm would be to obtain a higher resolution object estimate at the same time it is deblurred. As explained before, this is also known as SR. In order to do it, it is necessary to expand the mathematical model of our system using the downsampling operator D as shown in Eq. (2-28):

$$i_k(x,y) = D \cdot (h * o)(x,y) + w_k(x,y) \tag{4-1}$$

so the optimization unconstrained problem in Fourier domain would be:

$$\left\{\hat{H}_k,\ \hat{O}\right\} = \arg\min_{H_k,O} \sum_k \|I_k - D \cdot H_k \cdot O\| \tag{4-2}$$

Then, updating Eq. (3-3) and Eq. (3-4), the following expressions are obtained:

$$\hat{O}^{n+1} = \frac{\sum_{k=1}^{K} \hat{H}_k^{*n}}{\sum_{k=1}^{K} |\hat{H}_k^n|^2} \frac{I_k}{D} \tag{4-3}$$

and

$$\tilde{H}_k^{n+1} = \frac{1}{\hat{O}^{n+1}} \frac{I_k}{D}. \tag{4-4}$$

In this case, the term $\frac{I_k}{D}$ can be seen as the upsampling operator U; defining a High Resolution (HR) version of I as $I_k^{\mathrm{HR}} = \frac{I_k}{D} \equiv U \cdot I_k$ then it possible to keep using Eq. (3-3) and Eq. (3-4)

without any modification except on the input images, that only needs to be upscaled before hand.

It is also worth mentioning how the input images of the TIP algorithm should be scaled because there are several algorithms [35] [36] (e.g., bicubic interpolation, nearest neighbor and linear interpolation among others). The idea is to keep the sub-pixel displacement information of the averaged features in each pixel, so the best way of not disrupting it when resizing the image is by using *linear interpolation*.

NUI One of the interesting properties of TIP is that it allows deblurring and increasing the object resolution in the same process. However, this is a high computational task and in other conditions there are other techniques that could be used. A couple of those techniques are Papoulis-Gerchberg (PG) [37] and Non-Uniform Interpolation (NUI) [38]. NUI makes use of several LR images that are not blurred (making it ideal for using it with very low/inexistent aberrations) but that have certain sub-pixel displacement (see Figure 4-2). It works by first

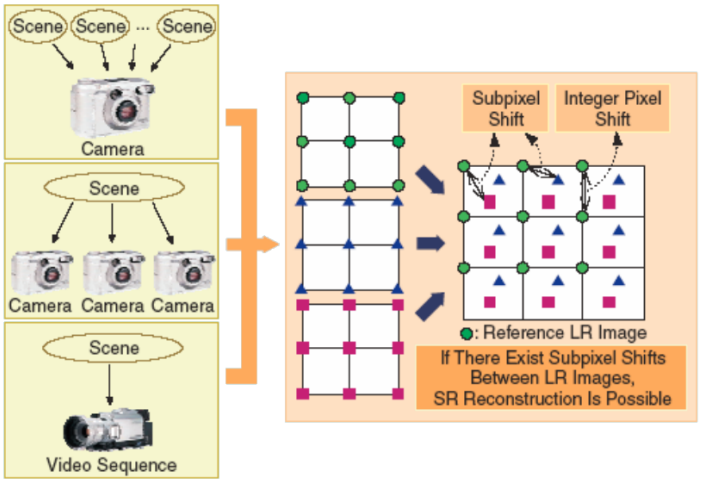


Figure 4-2: Graphical explanation of how the LR images are obtained and merged afterwards using NUI. Image credits [9].

calculating the shift between images and later building a *non-uniform* grid where the value of the pixels (of the desired output size) is calculated using interpolation algorithms. Figure 4-3 illustrates a comparison between the *bicubic interpolation* method (which does not fuse the information of several images) and the NUI (which clearly shows more details).

4-2 Further modifications

Although the algorithm has been proved to converge, some modifications to the code were added in order to increase its convergence properties in difficult cases (e.g., low contrast images with high noise). The modification implemented are the following:

Estimating the OTF from a LS solution There are cases where the high noise levels in the object estimation and input images (see Figure 4-4 for an example) produces a rather poor

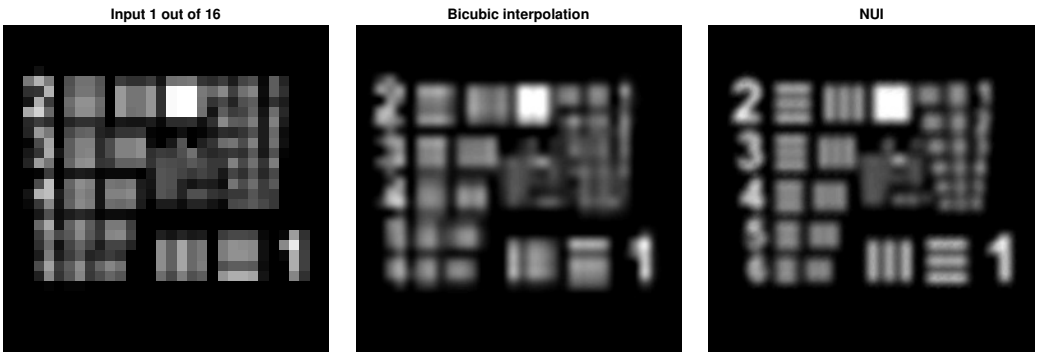


Figure 4-3: Comparison of the results obtained using *bicubic interpolation* and NUI. There are 16 input images of 32×32 pixels with random sub-pixel displacements, while both interpolated ones are of 256×256 (8x scaling factor).

and noisy estimation of the OTFs during the iterative process. However, a possible way to

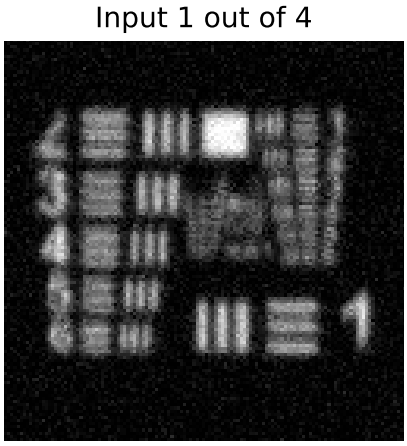


Figure 4-4: Aberrated input image with high levels of noise.

diminish this behavior is to estimate the OTFs using a LS approach due to its optimal noise removal properties. The optimization problem can be described as

$$\hat{H}_k = \arg\min_{H_k} \|I_k - H_k \cdot O\|, \tag{4-5}$$

with the solution being

$$\hat{H}_k = (\hat{O}^* \cdot \hat{O})^{-1} \cdot \hat{O}^* \cdot I_k = \frac{\hat{O}^* \cdot I_k}{|\hat{O}|^2}. \tag{4-6}$$

Therefore, Eq. (3-4) would be substituted by Eq. (4-6) in the iterative part of the algorithm. Figure 4-5 illustrates a comparison of the object and PSF estimated when the LS solution is used.

(a) PSF estimated using the LS solution. **(b)** PSF estimated using the original TIP.

Figure 4-5: Comparison of the results obtained using the original projection or the LS solution when estimating the PSFs. Observe how the LS solution helps to remove noise in the PSF.

Constraint the object in each iteration One peculiarity of the TIP algorithm is that it relies on the projections of the PSFs during the iterative process in order to get a feasible object. It is possible to constraint (or project) the object in each iteration for ensuring a faster convergence in certain conditions (see Figure 4-6). However, this procedure might also create instabilities (or divergence) in other conditions.

Figure 4-6: Comparison for when the object is constrained in each iteration (or not).

Subtraction of the minimum value When the object is constrained, it usually implies using a non-negativity constraint such as $\mathcal{P}_{\geq 0}$. However, although this has been shown to work in binary images, it can create darker areas (or a loss of contrast) in gray images. This is due to the fact that the estimation in the Fourier domain does not really produce normalized images in the spatial domain, therefore it is possible that the minimum value of the estimated images is not 0 but a negative number. A way to avoid this is to introduce a new projection, $\mathcal{P}_{\min} : x = x - min(x)$, that shifts the intensity of the image so it has a minimum value equal to 0.

Stochastic variation One property of the projection $\mathcal{P}_{\geq \alpha}$ used on the PSFs is that the removal of low values produces sharp edges which are related to an increase of high frequencies. Thanks to this effect, each time that the object is estimated using Eq. (3-3), it is more likely to obtain more high frequency information (where the fine details lie) of the object. However,

after a few iterations of the algorithm, this effect diminish due to the convergence of the PSFs to the *real* values, therefore less high frequency excitations is obtained which, in the end, makes the optimization algorithm to stall (it can be seen as when a local minima is reached).

One way to solve this is to keep exciting (to certain extent) the frequencies of the constrained PSFs in each iteration so it can escape local minimas and keep converging to a better minima. This can be done adding a random value to each frequency, the way implemented in this case is:

$$\begin{aligned}\hat{H}_k &= \text{stochastic}(\hat{H}_k, \gamma) \\ &= \hat{H}_k + \gamma \cdot \Big(\text{rand\_uniform}[-1,1] \cdot \text{Re}[\hat{H}_k] + j \cdot \text{rand\_uniform}[-1,1] \cdot \text{Im}[\hat{H}_k]\Big)\end{aligned} \quad (4\text{-}7)$$

Eq. (4-7) makes use of a random uniform distribution to create a matrix normalized to $[-1, 1]$. This matrix has a gain parameter γ and then it is multiplied and added to the actual values of the OTFs (both real and imaginary part). This allows vary each complex frequency by a maximum percentage of its actual value. Figure 4-7 illustrates how the SR version of TIP can obtain better results than the original TIP. In addition, this result can be enhanced with a careful selection of the γ value, increasing the high frequency reconstruction and thus, lowering the ringing in the image.

Updated TIP algorithm Due to the previous modifications and additions to the original TIP algorithm, the pseudocode has been updated in Algorithm 2.

4-3 Summary

In this chapter the original TIP algorithm has been modified in order to increase its robustness and capabilities.

One of the goals of this thesis is obtaining SR images, for achieving this, the image formation model was extended, leading to a modified algorithm. The idea is, basically, upscale linearly (for maintaining the sub-pixel displacements) the input images in order to match the desired output resolution.

Finally, further modifications are introduced into the algorithm for ensuring a faster and more robust convergence. Constraining the object during the iterative process and estimating the PSFs with the LS solutions increases the robustness against noise. In addition, due to the convergence to local minimas, a solution that requires the use of random numbers was explained.

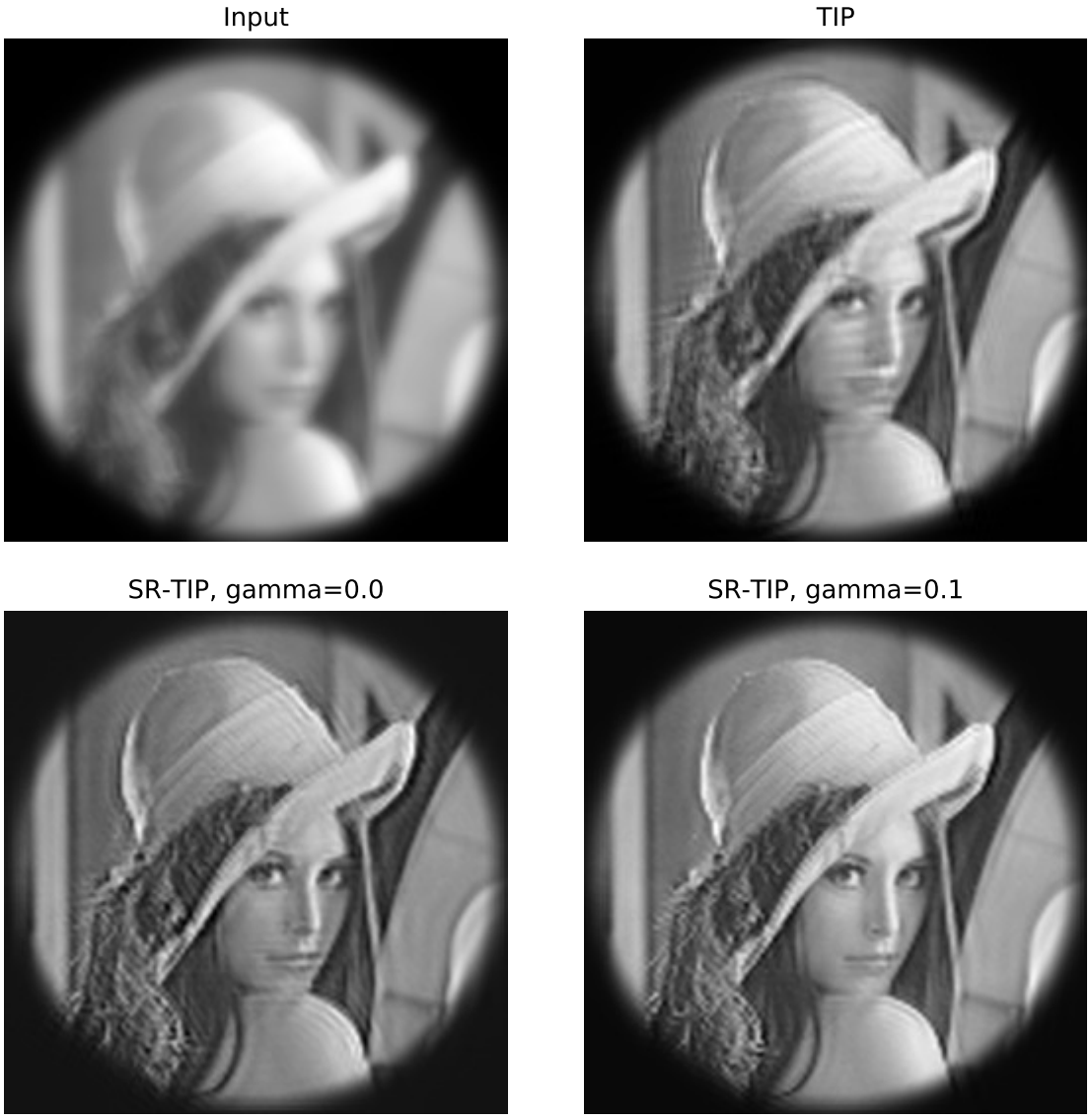


Figure 4-7: Comparison of the original TIP algorithm with the SR version for different values of γ. The sizes of the images are 128×128 for *Input* and *TIP* while for both $SR - TIP$ is 256×256.

Algorithm 2 Reconstruct the object using the modified TIP

procedure MOD-TIP($i_k, N, scale$) ▷ i_k is a stack of K blurred images.
  Initialization:
  $H_k^0 \leftarrow \mathcal{F}(\delta)$ ▷ The PSFs are initialized as delta functions.
  $i_k^{HR} \leftarrow$ **rescale**$(i_k, scale)$ ▷ Rescale the input images to the desired output size.
  $I_k \leftarrow \mathcal{F}(i_k^{HR})$
  Main Loop:
  for $n \leftarrow 0, N$ **do**
    $\hat{O}^{n+1} \leftarrow \dfrac{\sum_{k=1}^{K} \hat{H}_k^{*n} I_k}{\sum_{k=1}^{K} |\hat{H}_k^n|^2}$
    if *required* **then** ▷ Constraint object in each iteration.
      $\hat{O}^{n+1} \leftarrow \mathcal{P}_{\mathcal{O}} \left[\hat{O}^{n+1}\right]$
    end if
    for $n \leftarrow 1, K$ **do**
      if *required* **then** ▷ Select standard projection or LS solution.
        $\tilde{H}_k^{n+1} \leftarrow \dfrac{I_k}{\hat{O}^{n+1}}$
      else
        $\tilde{H}_k^{n+1} \leftarrow \dfrac{\hat{O}_{n+1}^{*} \cdot I_k}{|\hat{O}^{n+1}|^2}$
      end if
      $\hat{H}_k^{n+1} \leftarrow \mathcal{P}_{\mathcal{H}} \left[\tilde{H}_k^{n+1}\right]$
      $\hat{H}_k^{n+1} \leftarrow$ **stochastic** $\left(\hat{H}_k^{n+1}\right)$ ▷ Add random variations to all frequencies.
    end for
  end for
  Final Reconstruction:
  $\hat{h}_k \leftarrow \mathcal{P}_H \left[\mathcal{F}^{-1}(H_k^N)\right]$
  $\hat{O} \leftarrow \dfrac{\sum_{k=1}^{K} \hat{H}_k^{*N} I_k}{\sum_{k=1}^{K} |\hat{H}_k^N|^2}$
  $\hat{o} \leftarrow \mathcal{P}_O \left[\mathcal{F}^{-1}(\hat{O})\right]$
  return $\hat{h}_k, \hat{o}$ ▷ $\hat{h}_k$ is a stack of K PSFs.
end procedure

Chapter 5

Optical System Design

Once the TIP algorithm has been expanded with new features, the next step is to take advantage of its properties in order to develop a different kind of optical system. These properties are:

1. Good object estimation with few input images.
2. Possible to do Super-Resolution (SR).
3. Fast initial convergence.

Regarding the first property, it comes to mind the main use of this kind of Multi-Frame Blind Deconvolution (MFBD) algorithm; taking several images of the same object in a row. However, this means that for the entire set of images, the object must remain the same; this is essentially true for astronomical purposes (assuming perfect object tracking) but, this is not the case when observing dynamical objects in a telescope (e.g., a bird flying, a car in a highway, etc) nor in a microscope (e.g., moving samples). A way of solving this would be taking several images of the object at the same time with different aberrations (so the different images bring different information that the TIP algorithm can extract). This can be done partitioning the pupil of an optical system so it can hold several apertures and, therefore, projecting several images on the imaging sensor. The resulting device would be a multi-aperture optical system. In addition, this system would be very useful in microscopy because instead of illuminating the sample several times for getting different images that could be used in a MFBD algorithm, only one image is necessary, thus decreasing some degradation effects (e.g., photobleaching in fluerescence microscopy).

This pupil segmentation has the drawback of decreasing the imaging sensor area used for the image, therefore the final reconstructed image would be of smaller size (less resolution) in comparison with the image taken using the full-aperture. However, the second listed property of the algorithm can be used (SR). The TIP algorithm is capable of extracting high frequency information, making possible to retrieve a full sensor size estimated object using the multi-aperture system.

In addition, in order to develop a system capable of producing results fast, the third property of the algorithm is useful. Having an algorithm with a fast initial convergence and an optimized CPU/GPU code, allows to reconstruct the object almost in real-time. This topic is discussed in Chapters 7, 8 and 9.

5-1 Optical system schematic

The design of the multi-aperture optical system was carried out by Dean. Figure 5-1 illustrates a diagram of the final system adapted to be used as a microscope.

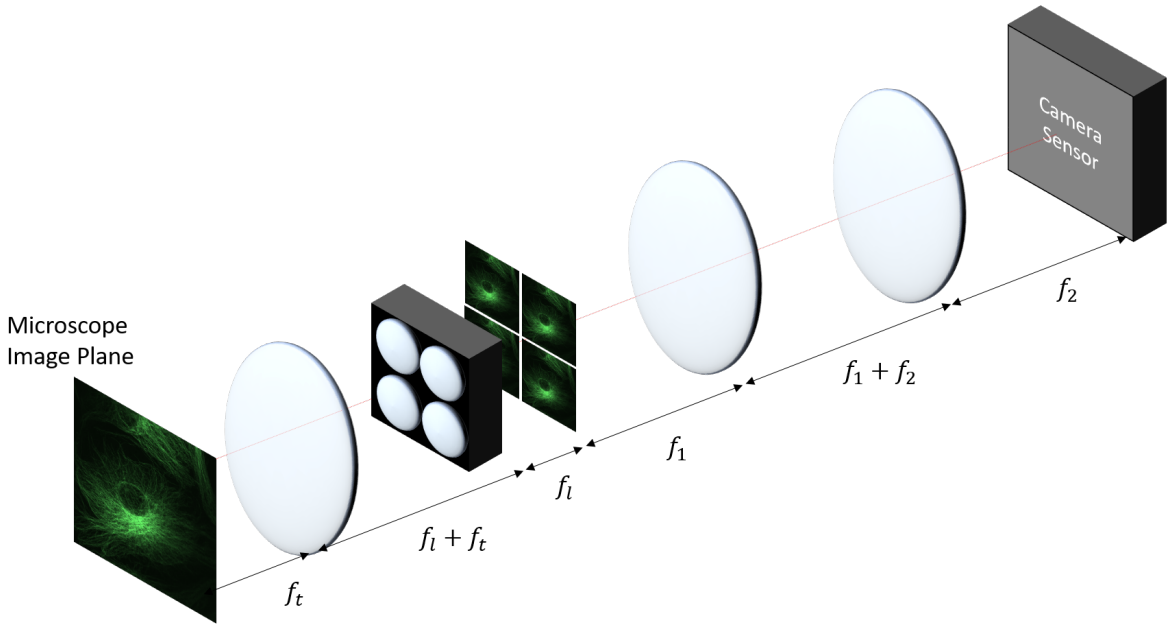


Figure 5-1: Diagram of the multi-aperture optical system acting as a microscope. Image credits: Dean.

The optical system works by, first, taking a full-aperture image of the object at a focal length f_t; afterwards, the image is divided in 4 (because in this design there are 4 lenses) at a distance of $f_t + f_l$ from the first lens. This produces already 4 separated images with different aberrations (as their light followed distinct paths from the object). Finally, the last two lenses adapt (with a magnification factor f_2/f_1) the size of the 4 images onto the camera sensor.

5-2 3D design

In the previous section, it was noted the necessity of holding 4 lenses together. In order to do that, a special lens holder was designed and 3D printed.

The main idea is to have a modular design that allows the use of different lenses configuration without having to disassemble the rest of the optical system. Therefore, two pieces are necessary: one piece used for holding the lenses in the required configuration and that can be

easily detached from the other piece; a secondary piece compatible with the optical equipment so it can be easily inserted in the optical path.

Figure 5-2 illustrates an sketch of the designed lens holder. First, the 4 lenses are inserted in a tap-like plastic piece so they remain in place. Second, the tap-like piece has the right shape to fit inside the other plastic piece.

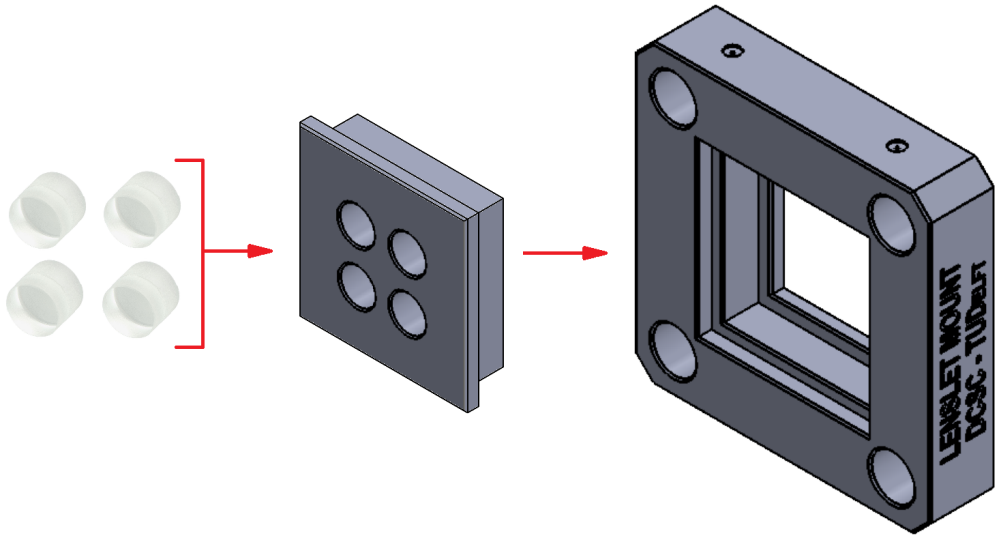


Figure 5-2: Schematic of how the lens holder works. The 4 lenses are inserted in the small plastic piece which, afterwards, is attached to the larger one.

5-3 Assembly

Once the design is done and all pieces are available, it is possible to start the assembly procedure.

Figure 5-3 shows an image of the lens holder completely assembled.

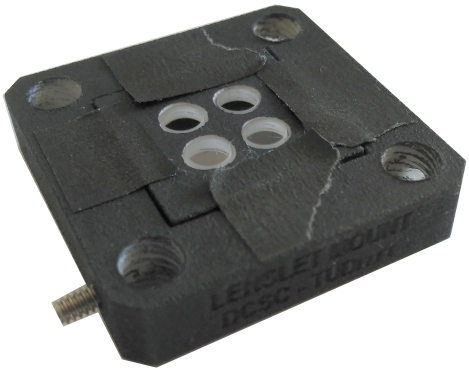


Figure 5-3: Picture of the 3D printed holder with the 4 lenses inserted.

The complete optical system is illustrated in Figure 5-4. In this case it is used as a telescope. The first two lenses gather and magnify a supposed collimated wavefront from a far object;

afterwards, it is divided in 4 images in the segmented pupil; finally, the image is adapted to the sensor size by the last two lenses.

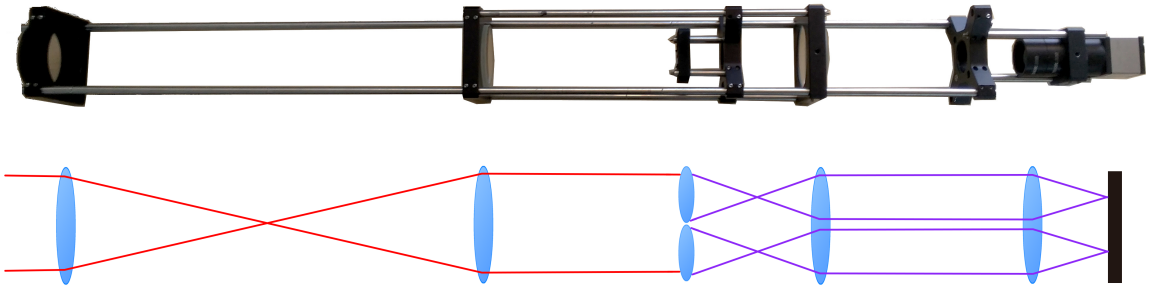


Figure 5-4: Multi-aperture optical system used as a telescope. The top image is a photography of the assembled system and the bottom one shows the optical diagram.

Figure 5-5 shows the complete optical system mounted on a tripod and looking at Oude Delft for testing the behavior of the algorithm.

Figure 5-5: Multi-aperture telescope in use.

Figure 5-6 shows the raw input image obtained from the imaging sensor. It is observable how the lenses produce 4 small images of the same object but with different aberrations (same areas in the image might look sharper or more distorted); also, they are slightly spatially moved due to the position of the lens.

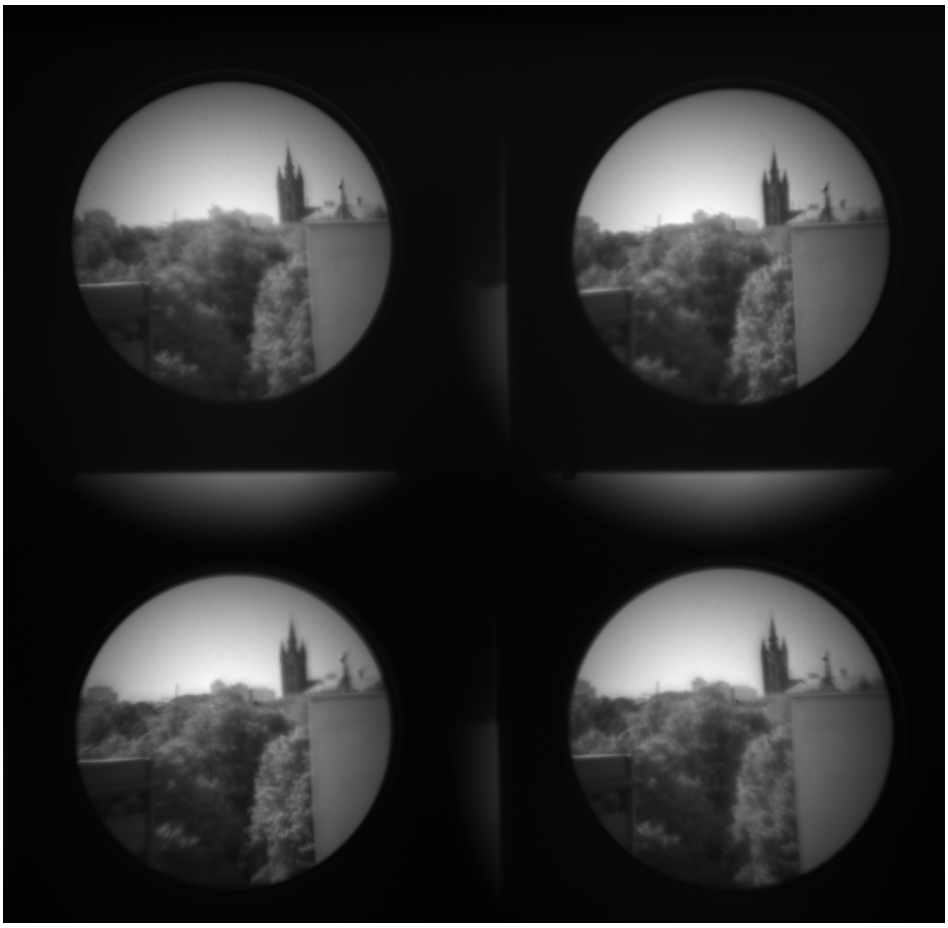


Figure 5-6: Image received on the camera from the partitioned aperture.

5-4 Summary

In this chapter, the different properties of the TIP algorithm and how a multi-aperture system could benefit from them has been discussed.

Such optical device has been designed along with specific pieces necessaries (lens holder) and assembled. Finally, its behavior has been tested.

Chapter 6

Numerical Simulations

This chapter is used for analyzing the expected behavior of the multi-aperture optical system (designed in Chapter 5) by means of numerical simulation. The idea is to simulate different kind of aberrated input images and observe the estimation provided by the TIP algorithm.

Firstly, the convergence properties are analyzed by running several numerical simulations with varying parameters. However, before this can be done, it is necessary to explain how the input images are generated and which metric is used.

6-1 Image generation

First of all, a phase aberration needs to be generated. This is achieved using random number generator plus Gaussian smooth as shown in Algorithm 3.

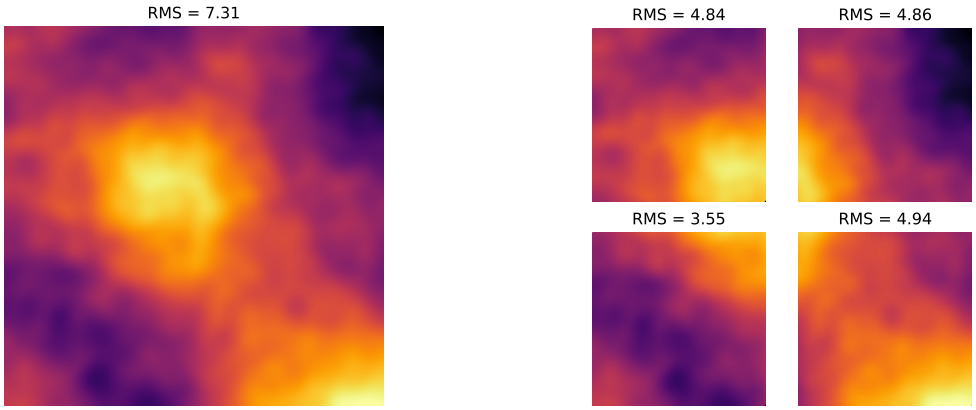
Algorithm 3 Generation of a wavefront aberration.

1: **procedure** GENERATEABERRATION(*npix, varLF, varHF, gainHF*)
2: $lowfreq \leftarrow$ **gaussianFilter**(**randn**$(npix, npix), varLF)$
3: $highfreq \leftarrow gainHF *$ **gaussianFilter**(**randn**$(npix, npix), varHF)$
4: $aberration \leftarrow lowfreq + highfreq$
5: $aberration \leftarrow$ **normalize**$(aberration, [-\pi, \pi))$
6: **end procedure**

In line 2 a smooth aberration map is created and, in order to obtain a more realistic PSF as observed in atmospheric turbulences, small high-frequency phase changes are added (see line 3); finally the aberration map is normalized and returned.

Figure 6-1 illustrates the phase aberration map obtained with the previous code. Also, the RMS is compared with the full-aperture case and the multi-aperture one. As it was expected, dividing the aperture makes possible to obtain images from regions that have smaller phase

aberrations (smaller RMS); although the final images obtained are also smaller in size, so less resolution achieved.



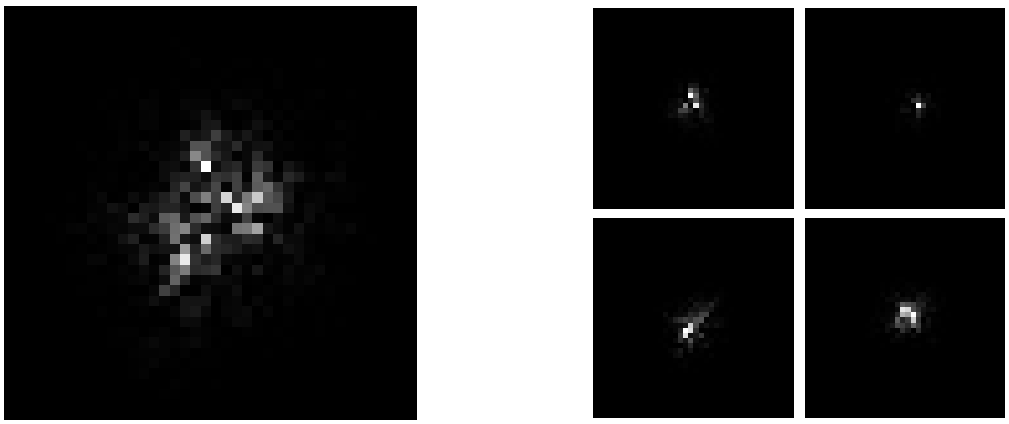
(a) Full-aperture. **(b)** Multi-aperture.

Figure 6-1: RMS calculation of the same phase aberration for a full-aperture and multi-aperture optical system.

Once the phase aberration map θ is available, the next step is to calculate the PSF produced by that aberration and the physical aperture A of the system. This can be done with the following equation:

$$PSF = |\mathcal{F}[A \cdot \exp\{\theta\}]|^2 \tag{6-1}$$

Figure 6-2 illustrates the PSFs obtained for both kind of systems. It is observable that for the multi-aperture system, the fact of having a smaller RMS phase aberration, produced PSFs which are less spread out (which in the end produced a less aberrated image).



(a) Full-aperture. **(b)** Multi-aperture.

Figure 6-2: PSF comparison between the optical systems which use a full-aperture or a multi-aperture.

Having the PSFs ready, calculating the aberrated object is a straight forward computation: convolve the object with the PSF and then add some Gaussian noise. Figure 6-3 illustrates the

images obtained. The full-aperture image is remarkably more blurred than any of the other multi-aperture images, proving the previous point that a smaller RMS phase aberration map produces a less aberrated object. In addition, it is remarkable that the multi-aperture images shows more fine details than the full-aperture one, even with a half the resolution (4 times less the amount of pixels); using the designed multi-aperture system when high aberrations are encountered can produce higher quality images.

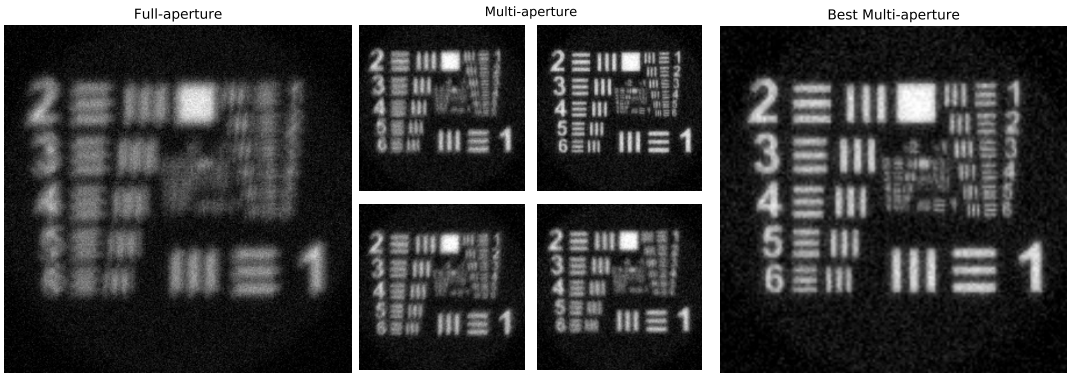


Figure 6-3: Image comparison between the optical systems which use a full-aperture or a multi-aperture.

6-2 Performance metric

Before the comparison of the algorithm performing in different circumstances can be done, it is necessary to select a metric which reflects as best as possible the performance (error) between the original image and the reconstruction obtained with the algorithm.

One very straight forward error metric would be the the Root Mean Square (RMS) of the error of the images $(o - \hat{o})$ in the spatial domain (see Eq. (6-2)):

$$\text{RMSE} = \sqrt{\frac{1}{m}\frac{1}{n}\sum_{m=1}^{M}\sum_{n=1}^{N}\left(o(m,n) - \hat{o}(m,n)\right)^2} \tag{6-2}$$

However, TIP does not necessarily reconstruct the object in the exact spatial location (it could be offset a few pixels) of the original object. This could lead to big RMSE values in the cases were the reconstruction is good but slightly shifted. One way of solving this would be by using information independent of the image shift, this is the magnitude of the Fourier spectrum (not the phase because is where the offset is stored). Then, the equation that arises is:

$$\text{RMSE} = \sqrt{\frac{1}{m}\frac{1}{n}\sum_{m=1}^{M}\sum_{n=1}^{N}\left(|\mathcal{F}[o(m,n)]| - |\mathcal{F}[\hat{o}(m,n)]|\right)^2} \tag{6-3}$$

Eq. (6-3) solves the problem of the offset between images but produces a new one; the different frequencies do no contribute equally in the error. The Fourier spectrum of an image usually has low frequency values that are orders of magnitude larger than the high frequency ones, this pose a problem when comparing the fine details of an image (produced by the

high frequencies). A solution to this problem is to compare the error of the magnitudes in percentages.

$$\text{RMSE} = \sqrt{\frac{1}{m}\frac{1}{n}\sum_{m=1}^{M}\sum_{n=1}^{N}\left(\frac{|\mathcal{F}[\hat{o}(m,n)]|}{|\mathcal{F}[o(m,n)]|}-1\right)^2} \tag{6-4}$$

Then Eq. (6-4) is chosen as performance metric for the following analysis.

6-3 Object estimation

Once the image generation function works, it is possible to use the TIP algorithm for different cases and parameters. In order to have a fair comparison, the algorithm is run with different values of the tuning parameters with the same set of input images (except when the number of images is being analysed).

6-3-1 PSF lower bound analysis

A very important tunning parameter of the TIP algorithm is the PSF lower bound (or α as explained in [10] and in Chapter 3). This a constraint that sets to zero all values of the PSF below a certain threshold, ensuring robustness against noise and creating high frequencies in the OTFs that then help to do a better estimation of the object in the following iteration.

Figure 6-4 illustrates the final estimation obtained. It is observable that high values ensure the convergence of the algorithm. However, when this value is lowered, a sharper estimated is obtained (because the PSF is more realistic) until a too low value makes the algorithm diverge.

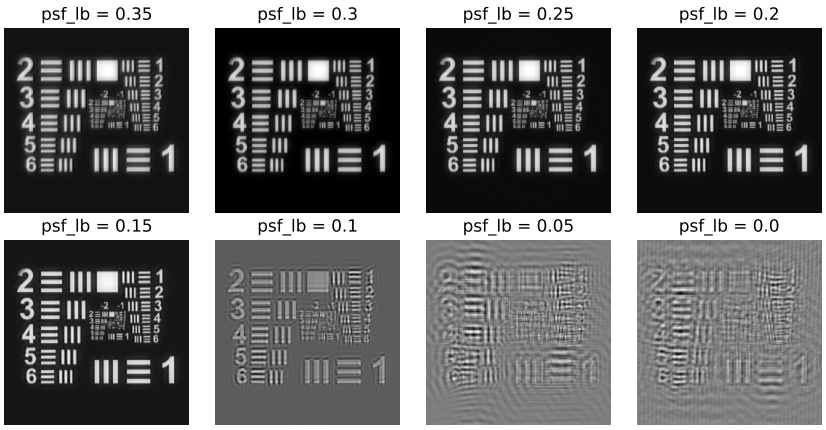


Figure 6-4: Comparison of the different estimations obtained for a varying value of the PSF lower bound for the USAF image.

This same trend can be observed in Figure 6-5. In addition, high values makes the algorithm converge faster (at a certain point the error stalls) but to higher error values.

This comparison is also made for the Lenna image, as the results are similar the reader may refer to Appendix A.

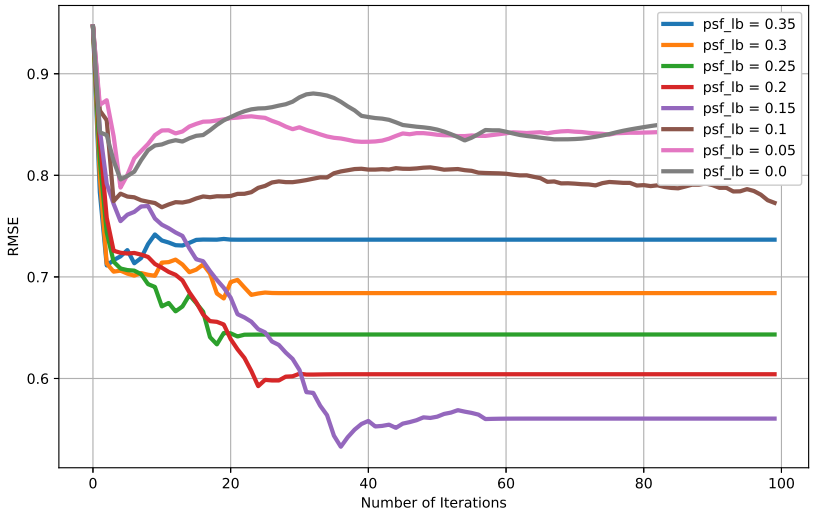


Figure 6-5: Comparison of the error evolution obtained for a varying value of the PSF lower bound for the USAF image.

6-3-2 Number of images analysis

Another parameter analyzed is the number of input images provided. It is expected that an increasing number of different input images improves the final reconstruction of the object because they carry, in overall, more information to be retrieved. This is exactly was it is observed in Figure 6-6.

Figure 6-6: Comparison of the different estimations obtained for a varying number of input images for the Lenna image.

When only one image is fed, the algorithm can not create information out of nothing so the solution found of the bilinear problem is $\hat{o} = i$ and $\hat{p} = \delta$ (which is the trivial solution). However, when more images are fed, the information fusion can be carried out achieving (normally) a lower error with more input images (see Figure 6-7).

In the previous Figure, it is observed a certain dip in the error evolution. This phenomena is

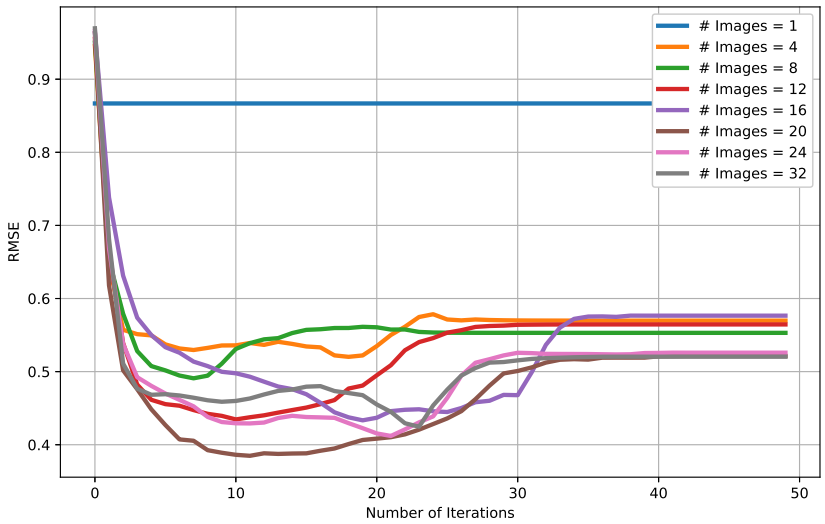


Figure 6-7: Comparison of the error evolution obtained for a varying number of input images for the Lenna image.

analyzed in the following section.

6-3-3 Dip analysis

The dip observed in the convergence of the algorithm in Figure 6-7 is produced due to the metric, which in some cases produces a value that is not realistic according to the human perception. This may be produced by a not good estimation of the DC term in the Fourier transform, as it changes the average intensity of the object.

Figure 6-8: Evolution of the object estimation through the iterative process.

Figure 6-8 illustrates the evolution of the estimated objects with different iterations; the lowest error is found after 10 iterations (see Figure 6-10) but, as it can be seen, the object is darker than the real object (see Figure 6-9) and suffers of certain ringing.

Figure 6-9: Real object used for creating the aberrated input images.

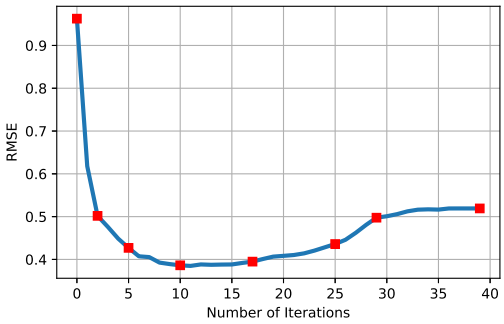


Figure 6-10: Graph showing the different iterations at which the object estimation is compared in Figure 6-8.

6-3-4 Noise analysis

As we are dealing with a physical system, it is worth determining the robustness of the algorithm to different levels of input noise. Figure 6-11 shows that when the noise level is too high, the estimated object is very noisy and blurred; when the level decreases the algorithm can filter the noise but not deconvolve until the level is below a certain threshold. In addition, the PSNR is calculated for the input image with and without noise.

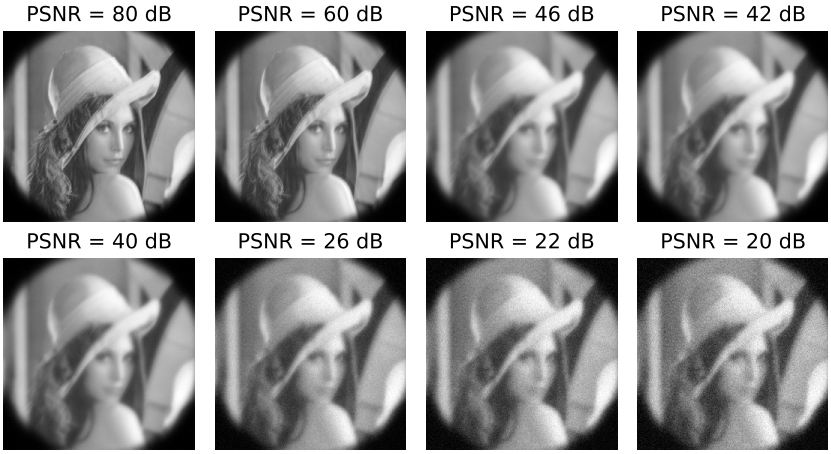


Figure 6-11: Comparison of the different estimations obtained for a varying noise level for the Lenna image.

The deconvolution process only works below certain noise levels (see Figure 6-12) and for the other cases, it tries to remove the noise fusing the images but no deconvolution is achieved. In addition, due to the non-converge of the algorithm for very low PSNR values, the error evolution is not shown.

Finally, the noise is treated extensively in Wilding et al. [10]. The reader is encouraged to refer to the original article for further details.

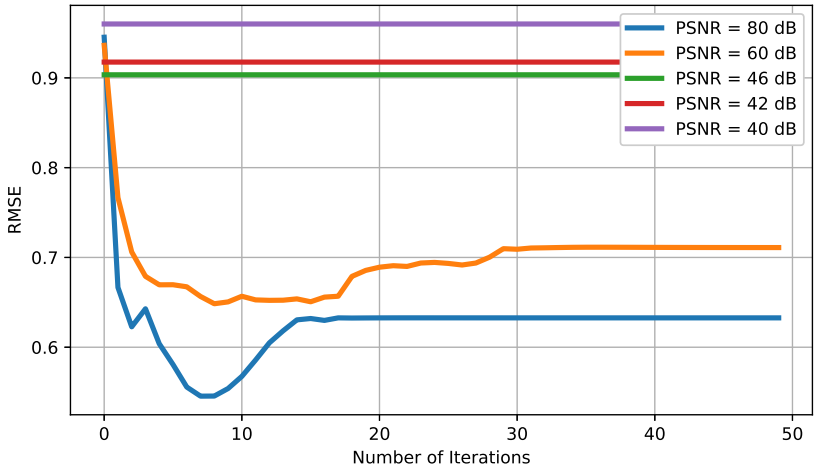


Figure 6-12: Comparison of the error evolution obtained for a varying noise level for the Lenna image.

6-3-5 Influence of *gamma*

As it was explained in Chapter 3, the TIP algorithm can converge to a local minima and stays there (the error evolution stalls at that moment). It was proposed the idea of applying random noise to all OTFs in each iteration in order to help the algorithm to leave that local minima and move to a better estimation. The control of the noise added is done thanks to a gain parameter γ, which influence is analyzed for the cases where Super-Resolution (SR) is required ($scale = 2.0$) or not ($scale = 1.0$).

Figure 6-13 illustrates how more details are retrieved with an increasing value of γ. However, although it is not shown, a too high value produces divergence in the algorithm, so it is always required to tune it. The error stall mentioned before is seen in Figure 6-14 for $\gamma = 0.0$.

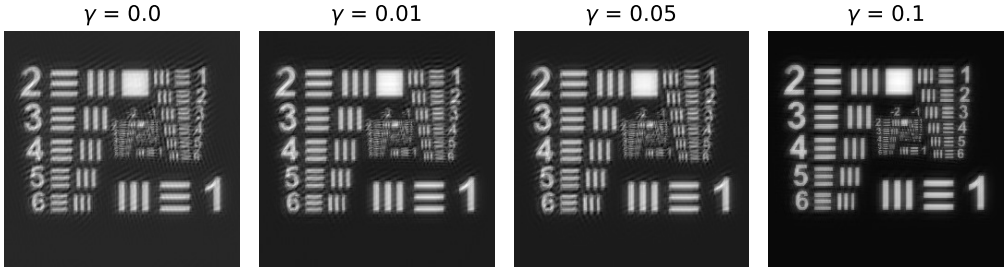


Figure 6-13: Comparison of the different estimations obtained for a varying γ value for the USAF image with $scale = 1.0$.

However, when $\gamma > 0.0$, the error keeps decreasing; this means that more information is being extracted of the input images.

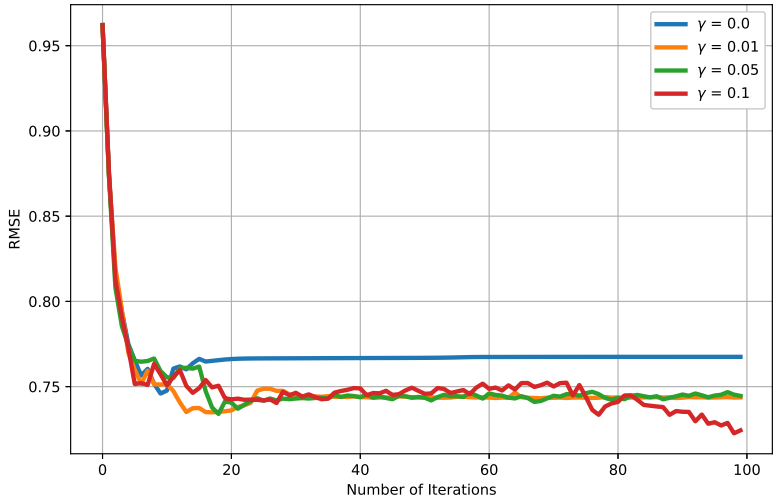


Figure 6-14: Comparison of the error evolution obtained for a varying γ value for the USAF image with *scale* $= 1.0$.

As doing SR turns an already very difficult problem into a more difficult one, this parameter helps to obtain a better estimation as seen in Figure 6-15.

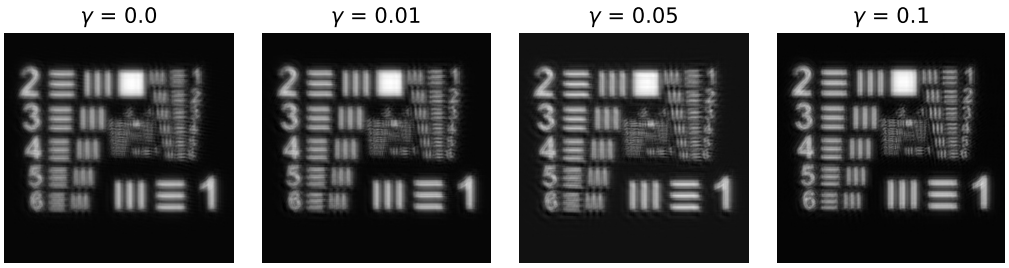


Figure 6-15: Comparison of the different estimations obtained for a varying γ value for the USAF image with *scale* $= 2.0$.

In addition, Figure 6-16 shows how this parameter sometimes produces small *hops* in the error (see the orange line), meaning that the optimization has escaped from a local minima and continues its convergence. It is also observable how sometimes the algorithm diverges momentarily (due to bad values combination in the OTFs) but is back on track a few iterations later thanks to the convergence properties of the algorithm.

6-4 Summary

In this chapter the behavior of the TIP algorithm working with the multi-aperture optical device has been tested by means of numerical simulations.

The image generation model is explained. Showing how the aberrated images plus noise are obtained from the optical system.

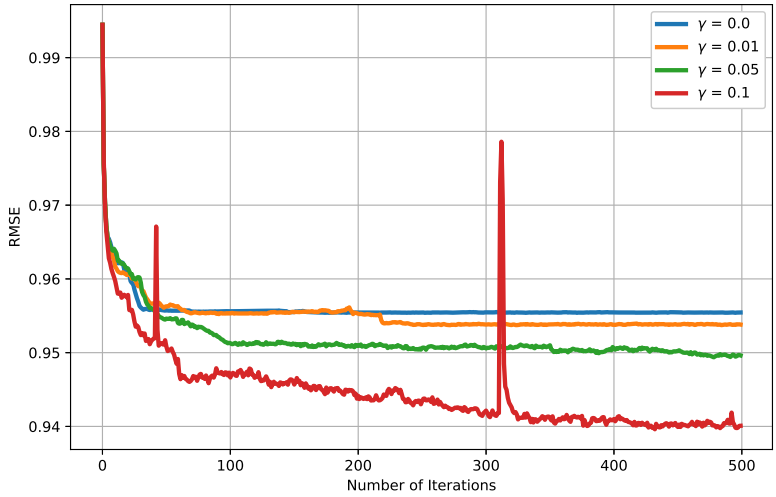


Figure 6-16: Comparison of the error evolution obtained for a varying γ value for the USAF image with *scale* $= 2.0$.

The final part of the chapter is a comparison of the convergence properties of the algorithm for different parameter values:

- The PSF lower bound produces a faster convergence for high values but poor reconstruction and, for too low values, it does not converge.
- The number of images fed to the algorithm is also analyzed; it is observed that an increasing number of images produces a better reconstruction, as it is expected.
- The noise is also treated, while the algorithm can deal with it and still obtain a good reconstruction with low levels, when the PSNR is too low, only fuses the information from the input images without any further improvement after each iteration.
- In order to get out of local minimas, the *gamma* parameter is introduced, showing that it actually allows the optimization method to retrieve more information and, thus, obtaining a better reconstruction.

Chapter 7

CPU Implementation

The main topic of this chapter is about the implementation in CPU of the TIP algorithm explained in Chapter 4.

One of the main decisions needed to be taken is about which programming language to use; in this case Python has been chosen because: it is open source (unlike MATLAB), it is very versatile and fast to develop new ideas (unlike C/C++), has a large variety of free libraries for science (`numpy, scipy, matplotlib`) and has a big and active community of developers. However, Python is far from perfect and in order to have a fast and stable algorithm, certain modifications are necessary.

7-1 Code performance

Although the language chosen is Python [39], it is still necessary to obtain a fast code due to the nature of the algorithm and the desired goal. One of the main problems inherent to the algorithm, it is the use of images because, although society is getting used to work with HR devices such as Full-HD (1920×1080) screens, only using 512×512 images means that there are 250000 data points to which the algorithm has to iterate several times and apply different mathematical operation. This is high computational extensive even on modern computers so a good code optimization and a wise selection of libraries is necessary.

The TIP algorithm can be divided in two blocks: one dedicated to the forward and inverse Fourier transform and the other focused in the mathematical operations required in each iteration.

Fourier transform As it was explained before, the TIP algorithm is used in the *deconvolution* of a signal from another one. Due to the easiness of convolve/deconvolve in the frequency domain compared to the spatial domain, the use of the Fourier transform is mandatory in this case. Therefore, the Fast Fourier Transform (FFT) is going to be used a great part of the time during the iterative process.For the previous reasons, the right selection of the FFT algorithm [40] and the right CPU implementation is critical for the final performance.

Mathematical operations The second big part of the algorithm is related to all the mathematical operations that each pixel has to undergo in each iteration. This are the LS solution, the affine projection and the projections for constraining. One way of speeding up these operations would be to compile the Python and other would mean to parallelize it.

7-2 Introduction to Python

Python is a high-level general purpose programming language, making it suitable for developing any kind of program while maintaining several layers of abstraction with the hardware. It is *interpreted* instead of *compiled*, this means that the whole code is not compiled into machine code directly but it takes pieces of it and executes them directly.

Being a high-level interpreted language makes it ideal for writing quick scripts for testing new ideas but, as a drawback, it is not directly applicable to real-time applications and it is not suited for low-level implementations that are optimized to the hardware available. However, during the years, there has been progress in this direction leading to the development of new libraries that allows the user to make a better use of the computer resources.

7-2-1 GIL

One of the mayor drawbacks for code efficiency in Python is due to the Global Interpreter Lock (GIL) [41]. This GIL imposes several restrictions on threads meaning that several CPUs can not be used at the same time in multi-thread programs. The reason it was implemented was because the C compiled Python routines (also called CPython) are run by a memory manager that is not thread safe; then, without GIL, there could be data corruption. However, it is still possible to use a single-thread program launched several times in different cores.

7-3 FFT

As stated before, the FFT implementation in Python is going to have a great impact in the final speed of the code.

7-3-1 Complexity calculations of FFT

Before continuing with the explanation of the FFT implementation used, a complexity analysis of the convolution in the spatial domain compared to the frequency domain is done.

In systems theory, the convolution operator is used to obtain the output of a system when a certain input is applied. In imaging, the input is the object while the system is the PSF, this model is explained in Eq. (2-3).

In the spatial domain, convolving the input of the optical system (object) and the kernel (PSF) is a very computational costly operation. For an object (o) of size $M \times N$ and a PSF (h) of size $m \times n$, the number of operations required for convolving is:

$$\text{Operations of } (h * o) = (mn) \cdot (MN) = \mathcal{O}(mnMN) \tag{7-1}$$

However, assuming that all sizes are the same ($M = N = m = n$), the complexity turns to $\mathcal{O}(N^4)$, which becomes terribly large for standard image sizes. Usually the PSF is smaller than the object, showing a lower complexity but still being computationally very expensive.

One way of decreasing this complexity is by convolving in the frequency domain because it is a point-wise multiplication with $\mathcal{O}(MN)$ (the PSF is padded with zeros until it has the same size than the object) but it is also necessary to use the FFT and the IFFT.

$$i = h * o = \mathcal{F}^{-1}\left[\mathcal{F}[h] \cdot \mathcal{F}[o]\right]. \tag{7-2}$$

Eq. (7-2) shows the equivalence between the spatial and frequency domain convolution. Assuming that the efficient FFT is used, with complexity $\mathcal{O}(k \cdot MN \cdot \log_2(MN))$ and k being a small integer dependent on the FFT algorithm implemented ($k = 5$ for Cooley-Tukey), the final complexity is (with $N = M$);

$$\begin{aligned} \text{Complexity} &= 3 \cdot \mathcal{O}(FFT) + \mathcal{O}(\text{Mult}) \\ &= 3 \cdot \mathcal{O}(k \cdot MN \cdot \log_2(MN)) + \mathcal{O}(MN) \\ &= \mathcal{O}(3k \cdot N^2 \cdot \log_2(N^2) + N^2). \end{aligned} \tag{7-3}$$

For most situations, $\mathcal{O}(3k \cdot N^2 \cdot \log_2(N^2) + N^2) << \mathcal{O}(N^4)$. It can be the case that the PSF is very small in comparison to the object but still, in the high majority of situations, the spatial convolution will be slower. Using the frequency domain approach means that the TIP algorithm can do a fast deconvolution for estimating the object and PSF.

7-3-2 FFT implementation

Initially, there are two main open source libraries in Python that are used for scientific purposes: `numpy` [42] and `scipy` [43].

`numpy` This library was designed mainly with the goal of providing support for large, multi-dimensional arrays and matrices, using a bast collection of functions that operate mathematically on them. Although it uses CPython, and hence has a non-optimized code, it provides functions and operators that can work efficiently on multidimensional arrays. Internally, the Basic Linear Algebra Subprograms (BLAS) and Linear Algebra Package (LAPACK) libraries are used in order to have high performance while doing linear algebra computations.

`scipy` This other library offers more advanced functions than the previous one. The modules offered are: optimization, linear algebra, integration, interpolation, FFT and ODE solvers among others. It is build on `numpy` so it can do efficient computations on arrays.

GIL limitation This libraries already offer a very good FFT implementation but after testing them, it is observed that they are not making use of the multi-core capabilities. In short, the GIL is limiting the performance of these libraries so only one core is used during the FFT computation.

A possible solution to this limitation is to directly call the precompiled functions, in which the libraries are based on, because then they would not be limited by the GIL and, therefore,

making possible to use several cores of the computer. This functions are provided in packs and the most famous ones are the Fastest Fourier Transform in the West (FFTW) and the Intel Math Kernel Library (MKL).

FFTW and MKL FFTW [44] is a specific library for only computing the Discrete Fourier Transform (DFT) of a data set. Being a very fast implementation that can compute transforms of real and complex arrays of any size and any dimension $\mathcal{O}(n \log n)$ time. The way it keeps a low computational time is by choosing the right algorithm depending on the properties of the input data. It usually works better when the array sizes are composed of small prime factors being always the optimal the powers of two. It supports several variants of the Cooley-Tukey, Rader and Bluestein FFT algorithm. Having as core an unrolled FFT.

On the other hand, MKL [45] is an Intel library which comprises several algorithm for mathematical computations that are specifically optimized for Intel CPUs. This library supports the BLAS and LAPACK functions for linear algebra and also different algorithms of the FFT.

Both libraries could be used for the FFT but using the MKL is specially advised because is updated regularly, it supports natively multi-core capabilities and it has been specifically optimized for running on Intel CPUs.

7-3-3 `mklfft`

The `mklfft.py` library has been developed with the purpose of supporting the use of the Intel MKL library, specifically the FFT implementation, in Python with multi-threading capabilities.

Configuration The core of this library is the file `mkl.dll`, which is a C++ compiled open source code provided by Intel. In order to make use of it, it is necessary to access it using C data types, this is done with the `ctypes` library. Once the library has been loaded in Python, the next step is to create a *handle* where all parameters are stored:

- *Threads availables*: This enable multi-core capabilities independent of the GIL.
- *Precision*: Selects *single* or *double* float precision for the computation.
- *Placement*: Selects if the result is returned in the input array or in another array.
- *Batch*: It is possible to launch several FFTs of different data sets creating a batch of arrays.

Launch the FFT Once the parameter *handle* is finished, now it is possible to launch the computation of the forward or backward FFT transform.

Deallocating memory As we are using a C++ library called from Python, there is no *garbage collector* available; therefore it is a good custom to liberate the memory of *handle* and other arrays used.

7-4 Code compilation

The first important block of the algorithm was the FFT transform, which has been implemented in the previous section. In this section, the focus is on the second block, the mathematical operations.

`numpy` already provides very fast functions (it uses precompiled C functions) that can be used for doing the required computations of the TIP algorithm. However, it is still possible to do a smarter implementation in certain operations that can be speed up being precompiled in C before execution.

The idea of use C compiled code is because: C language is closer to the hardware, so it allows a better utilization of the computer due to low-level optimization but it also makes it harder to develop; Python interpret each line of code before its execution, creating certain lag that could be avoided if the code was already compiled.

Although Python is a high-level language, there are certain libraries that allow the compilation of functions in C-code: mainly `Cython` and `Numba` [46] (from Anaconda Accelerate).

`Cython` From the developers webpage: "`Cython` is a programming language that makes writing C extensions for the Python language as easy as Python itself." This library works adding static type declarations (the data types used in C), then translating the code into optimized C/C++ code that then is compiled as a Python extension module; afterwards, Python call this functions when executing the main code. Although it is still being maintained and improved, there are other modules that offer a faster C compiled code and better support.

`Numba` This library offers the same properties as `Cython` getting also better performance and a even simpler syntax (all code can be written in standard Python language and then, by means of a preprocessor directive or wrapper, it can be compiled). `Numba` is recommended when array-oriented and/or very math-heavy computations code is developed in Python. The code is Just In Time (JIT) compiled (it is dynamically compiled but also cached for the next executions) to machine code, achieving similar performance to a low level language like C.

7-4-1 Python compilation with `@jit`

`Numba` compiles the desired code putting the decorator `@jit` on top of the function. There are several ways of invocation modes that produces a different compiled behavior:

- Lazy compilation: If the `@jit` decorator is used alone like in the following code.

```
1  @jit
2  def f(x, y):
3    return x + y
```

  The compilation will be done when the function is called for the first time (causing a certain overhead). `Numba` will infer the data types used and generate a compiled code optimized for it.

- Eager compilation: This means using a function's signature to specify the argument types of the function.

```
1 @jit(int32(int32, int32))
```

  Although that there are other parameters that can be specified in the function's signature:

  - `nopython`: `Numba` offers two modes for compiling the function: *nopython mode* and *object mode*. The former is required for achieving the highest performance because the code does not access the Python C API (because it uses Python types instead of C equivalent types).
  - `nogil`: If the optimized compiled code works on native types (no Python types), the previously explained GIL can be released, making it possible to run the code concurrently in several threads.
  - `cache`: In order to avoid the compilation of the function after the first call, it is possible to store the compilation file in a cache file that then is called anytime it is requested without the need of compiling again.

Using the previous explained options and characteristics, the application of `Numba` for speeding up the computations of the TIP algorithm would be:

- Describe the mathematical operations in arrays (which are already point-wise) in a loop manner, so the compiled code applies optimization in the loops (basically a coalesced access to the memory and prefecht of values into the CPU cache).
- Unroll certain mathematical operations for avoiding expensive non-native CPU operations. For example, in the LS the operation, the operation $|H|^2$ is performed several times, which for a complex value is equal to $(\sqrt{\text{Re}[H]^2 + \text{Im}[H]^2})^2$ but this is equivalent to $\text{Re}[H]^2 + \text{Im}[H]^2$. This last modification avoids the use of $\sqrt{}$, a very computational expensive non-native operation, and a ^{2}.
- Avoid reading several times the same array and maximize the number of modifications done to it in each read. For example, when constraining the PSF, it is possible to apply the lower bound constraint at the same time it is normalized.
- Use the options: `nopython` and `nogil` for obtaining the fastest code possible that can alse be run concurrently in several threads if necessary.

7-4-2 `mkltip`

In the library `mkltip.py`, there is a main function `mklTIP` where the TIP algorithm is implemented and then there are several smaller functions that are the ones compiled by `numba`.

The `mklTIP` code is divided 3 main sections:

- Initialization: The precision of the computations is decided according to the input data type, the input images are scaled to the required output size and the FFT plan of the `mklfft` library is done.

- TIP algorithm: In this section the iterative process of the TIP algorithm is written. In addition, the functions called are all of them `numba` compiled.
- Cleaning up: After all computations are done, the allocated memory for the FFT plan is released.

The functions developed for being compiled are:

- `lsLoop`: From a set of OTF and blurred images, it calculates implicitly the object doing a LS and then estimates the new set of OTF from it.
- `ls`: This function calculates the object using a set of OTFs and input aberrated images.
- `constraintLoop` and `constraint`: Function used for constraining the PSFs in a loop or just the object. It takes the real part, normalize from 0.0 to 1.0, set to 0.0 those values below the lower bound of the PSF/object.
- `spatialFilter`: Function used to filter the spatial frequency of the OTFs.
- `stochasticPSF`: Add a random complex value obtained multiplying a uniform distribution in [-1, 1] with the actual value times a control gain (gamma).
- `lsPSF` and `divPSF`: Calculates a new estimate of the OTF by means of safe division or from the LS solution.

7-5 Summary

In this chapter the CPU implementation of the TIP algorithm has been explained.

Python has been chosen as main programming language due to its versatility. However, as it creates non efficient code, several libraries have been used for solving this problem.

The first library used was the `Intel MKL`, which provides the fastest FFT implementations for Intel CPUs. This library was then called from the developed library, `mklfft`, in Python. In addition, in order to increase the performance of the mathematical computations of TIP, another library was developed, `mkltip`. The idea is to C-compile the slowest functions in order to achieve a shorter running time due to a better optimization to the hardware and better array accessing.

Chapter 8

GPU Implementation

The main topic of this chapter is about the implementation in GPU of the TIP algorithm explained in Chapter 4.

GPUs operate simultaneously in a large amount of data elements because of their parallel design. Operation parallelism is a concept that appears when the same computation is done to several data elements. Initially, GPUs were conceived in order to do the several easy and repetitive computations needed for graphics problems. However, due to the parallel nature of GPUs, it is possible to apply them to different problems; this is known as General-Purpose computing on GPU (GPGPU) [47]. As the TIP algorithm works with images (that are basically 2D arrays of numbers) which are convolved and constrained using point-wise operations. The idea of implement this algorithm in a GPU arises naturally.

CPUs are specially designed to offer a low latency in complex computations due to the high clock frequency and large CPU instructions set. Therefore, CPUs needs a large cache (memory on chip) and complex control structures so the latency can be minimized, being specially efficient in sequential programming (which is the majority of the code). In contrast, GPUs are more focused on raw computational power output due to the high number of low power processing units although it suffers from latency problems because the cache is and control is kept to minimum. Figure 8-1 illustrates that the CPU devotes more area to the control circuitry and cache, while the GPU is more focused on achieving more computations in parallel.

Before continuing with the rest of the chapter, the reader may refer to Appendix B for a brief introduction in GPU programming.

8-1 CUDA in Python

In the previous chapter, a CPU optimized code was developed for the TIP algorithm. As Python is very versatile due to the big community behind writing different libraries, it is possible to use it for developing CUDA code as well (instead of using C++, which is the actual standard for this situation).

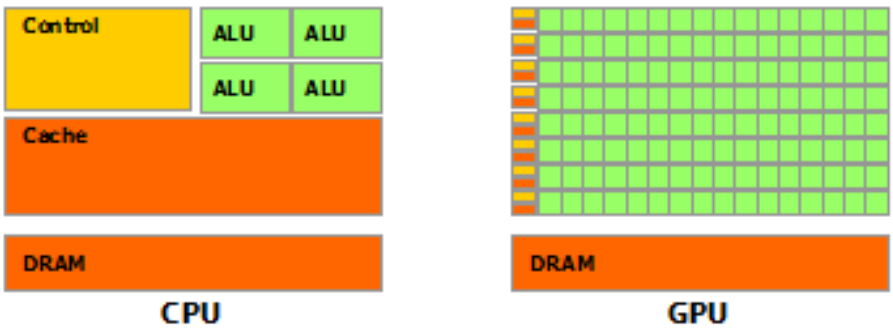


Figure 8-1: Comparison of the chip area devoted to different functions in CPUs and GPUs. Image credit: NVIDIA

Nowadays there are several libraries that interface Python with the CUDA API. This would make the CPU running Python the host, which would take care of the communications and the launching kernels, while the GPU with CUDA would be the device in charge of executing those kernels in parallel.

The most used libraries up to date are: PyCUDA [48], Scikit-CUDA [49] and Anaconda Accelerate. The latest is the one used because:

- The module *Numba* was already used for developing the CPU optimized code, so the library is already available and working in the system. In addition, the CUDA code is build with very similar preprocessor directives.
- The GPU code is written in Python syntax unlike PyCUDA.
- It has access to the CUDA libraries CuFFT and CuRAND, which are necessary for the full implementation of the algorithm.

8-2 `gputip`

This section is focused on the code developed for using the TIP algorithm in a GPU. As mentioned before, Python is used as programming language and the CUDA interface is done with the Anaconda Accelerate toolbox.

There are two basic functions: `gpuConfig` and `gpuTIP`.

8-2-1 `gpuConfig`

Before running the TIP algorithm on the GPU, it is necessary to configure it. The configuration file created with `gpuConfig` comprises:

- *Precision*: Depending on the input data type, the computations will be done with single or double precision.
- *Stream*: Depending on the number of input images, the same number of streams will be created so several kernels can be run concurrently using different arrays.

- *FFTplan*: The use of the CuFFT library needs to create a plan before computing the FFT. In this case, two plans are created: one for a single FFT and another for several single FFT done in batch.
- *Reduction Block Size*: As it is necessary to calculate the *max* and *min* of certain arrays, for constraining purposes, a reduction kernel is used. For each iteration of the kernel, a different block size is calculated beforehand.
- *Number of iterations of reduction kernel*: Depending on the input size, a different number of iterations of the reduction kernel may be necessary.

8-2-2 `gpuTIP`

This function is where the TIP algorithm is implemented. It has as input values the aberrated input images, the configuration file and the different parameters of the algorithm (e.g., the lower bound of the PSF).

The function first makes an intialization of the system: unpacks the configuration file, build the (scaled) input array, preallocate memory on the GPU for the arrays needed and seeds the random number generators of CuRAND. Afterwards, the iterative process of TIP starts. While Python is run in the host, being in command of issuing the required computations by means of the GPU; the GPU executes the kernels. In order to fully implement TIP on the GPU, the following kernels were developed:

- `initArray`: As there is no native function for initializing an array to a specific value, this kernel was written.
- `lsLoop`: Using H and I as inputs, a next estimated of H is calculated first by obtaining O and then doing a safe division between I and O.
- `ls`: In the case that only O is required.
- `constraint`: Apply the constraints on the object or PSF.
- `spatialFilter`: Apply the spatial frequency filter on the OTF.
- `max` or `min`: Reduction kernel for computing the maximum or minimum value of an array.
- `stochasticPSF`: Modify the OTFs randomly according to Eq. (4-7).
- `lsPSF` and `divPSF`: Calculate the next estimation of the OTF by using the division or LS solution between I and O.

8-3 Summary

In this chapter the GPU implementation of the TIP algorithm has been explained.

An introduction to GPU and CUDA can be find in Appendix B. The main ideas explained were: parallelization of computations, heterogeneous programming (GPGPU), CUDA architecture, execution hierarchy, memory hierarchy, streams and kernel reductions.

Afterwards, the library implemented in Python for the TIP algorithm in the GPU, `gputip`, is presented. Although, it has not been mentioned before, another library, `gpufft`, has been developed for calculating the FFT on the GPU. The main purpose of this library is for comparing their performance with other libraries in the following chapter.

Chapter 9

Computational Performance Analysis

In Chapter 7 and Chapter 8, the TIP algorithm was implemented using two different approaches: CPU and GPU programming. For this chapter, a computational speed comparison between the different implementations is done. The main idea is to know which library is more efficient depending on the input characteristics and hardware limitations. As the FFT computation is a core feature of the algorithm, it is going to be tested independently in Section 9-1. Afterwards, the TIP implementation is compared for both implementations (see Section 9-2). The main reason behind this analysis is to see if it is possible to achieve the goal of a fast object reconstruction (< 1 s) in the final optical system.

Before starting with the comparison, it is necessary to specify the conditions in which the tests are run, making it reproducible in other machines. Table 9-1 shows the software versions relevant to the comparison while Table 9-2 and Table 9-3 do the same but for the hardware specifications of both CPU and GPU.

Table 9-1: Software specifications of the system

Software	Details
OS	Windows 7 Professional SP1 - 64 bits
Python interpreter	Anaconda Python v2.7.13
MKL library	v11.3.3
Numpy library	v1.11.2
Scipy library	v0.18.1
Numba library	v0.30.1

9-1 FFT comparison

In this section, several FFT implementations in Python are tested, mainly from different libraries. As the TIP algorithm makes use of images, the tests showed in this section are only focused on 2D arrays.

Table 9-2: CPU hardware specifications.

CPU Hardware	Details
Processor	Intel i7-4790 CPU
Cores / Threads	4 / 8
Clock frequency	3.60 $\sim$ 4.00 GHz
Cache	8 MB Smart Cache
Bus Speed	5 GT/s
RAM quantity	4 $\times$ 8 GB (DDR3)
RAM frequency	1866 MHz

Table 9-3: GPU hardware specifications.

GPU Hardware	Details
Graphic card	Nvidia Geforce GTX 980
CUDA cores	2048
Clock	1126 $\sim$ 1216 MHz
Memory quantity	4 GB (GDDR5)
Memory clock	7.0 Gbps
Memory interface	256 bit
Memory bandwidth	224 GB/s

The libraries tested are:

- `numpy`: Standard library for computing the FFT in Python.
- `scipy`: Another standard library with a different FFT implementation.
- `mklfft`: A library previously developed in Chapter 7.
- `gpufft`: Library previously developed for computing the FFT in the GPU. Two timings are done for this code, the FFT alone and the FFT plus the data transfer from the *host* to the *device* and vice versa.

In order to see the performance in different conditions, the same test is done with different parameters. These parameters are the following:

- *Precision*: Input arrays of *single* (`complex64`) and *double* (`complex128`).
- *Input size*: Two sets of sizes will be used, one with powers of two (the FFT is optimized for this input data size) and the other with other combinations that include prime numbers (the worst case scenario).
- *Number of images*: The TIP algorithm uses more than one image at a time so it is necessary to see if a correct configuration of the algorithms can take advantage.

In addition, two measurements are going to be made. The first is the total computational time and the second is the *flops* as measured by the creators of FFTW on their webpage. Although it is not the real count of Floating Point Operations per Second (FLOPS), it is an approximation done using the complexity of the Cooley-Tukey algorithm:

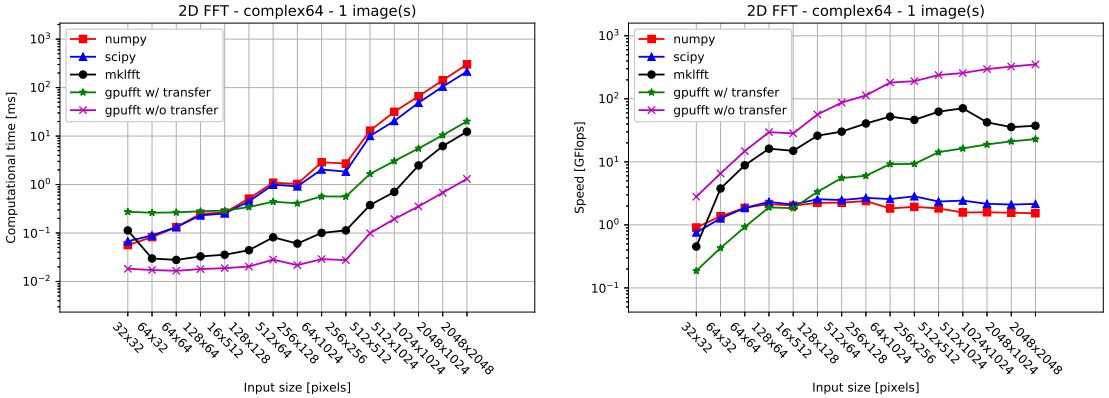
$$\text{FLOPS} = \frac{5 \cdot N \cdot \log_2(N)}{\text{Elapsed time [s]}} \tag{9-1}$$

9-1-1 `complex64`

First, all computations are done with `complex64` values (each complex number is composed of two `float32` values).

Before showing all the different comparisons, Figure 9-1a is analysed. It is observable that both `numpy` and `scipy` offers similar results than `mklfft` with small input sizes. However, when the input arrays grow in size, `mklfft` maintains its computational time (due to the multi-thread implementation) until it is around 10x faster than `numpy` and `scipy` (when the *cache* saturates).

About the GPU implementation (`gpufft`), it is reasonable to measure the time it needs to do the FFT with and without the memory transfers between the host (CPU) and device (GPU). Due to this memory transfers, it is not worth using the GPU for doing only one FFT when `mklfft` is available. However, if several FFT computations are done in the GPU, it would be faster than any CPU implementation; in addition, the FFT implementation in GPU can be around 10x faster than the best available in the CPU.



(a) Computational time.

(b) FLOPS count.

Figure 9-1: Performance comparison for solving one FFT of `complex64` values with power of two input sizes.

The other analysis performed is about the relative FLOPS that each implementation provides. Figure 9-1b illustrates this FLOPS comparison. It is observed that for the basic Python libraries, `numpy` and `scipy`, there are almost no change for the different input sizes, meaning that they are already limited by the Hardware (HW) and their implementation. On the other hand, the other two libraries, apart from showing a higher number of FLOPS, also have an increasing performance with the image size until they are bound by the HW. Comparing both `mklfft` and `gpufft`, the memory transfer from and to the GPU reduces to a great extent the performance of the GPU for doing the FFT (as explained before).

Analysis for several input images Figure 9-2 illustrates the computational time of the different implementations for 1, 4, 9 and 16 inputs images. As the number of images increases, `mklfft` and `gpufft` performs better for a low image size in comparison with the other two libraries. The previous behavior is also observed when comparing the FLOPS (see Figure 9-3)

Non power of two input sizes It is also worth showing how the implementations perform when the input sizes are not power of two and even some of them are prime numbers.

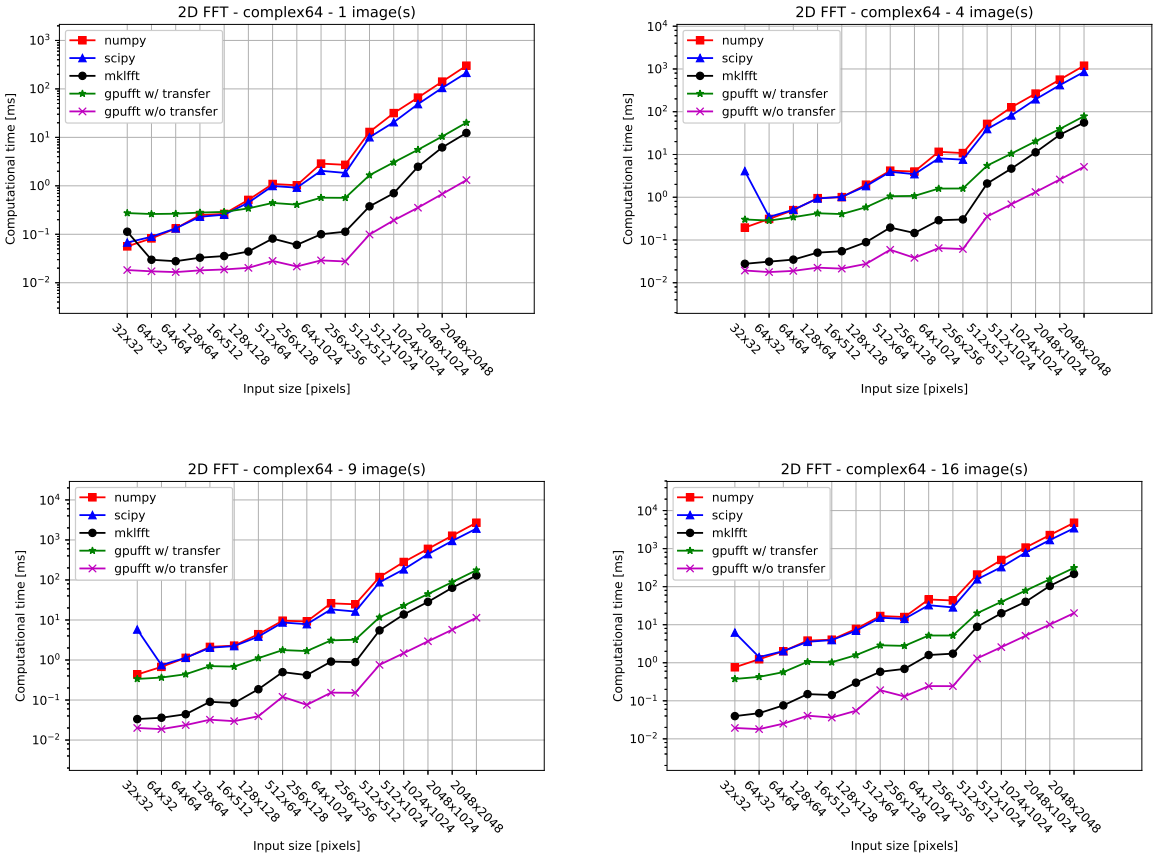


Figure 9-2: 2D-FFT computational time comparison of the different implementations with a varying number of input images.

Figure 9-4 illustrates how these non standard sizes produces a far worse performance. Depending on the input size, the library choses the right FFT algorithm for computing the FFT but, as it can be observed, these algorithms are not as efficient as the standard Cooley-Tukey. Although previously it was shown that the GPU implementation needed larger input sizes for having a similar performance as `mklfft`. Now there are cases (mainly prime numbers) where this performance is practically equal. This shows the robustness of `gpufft` for varying and non standard input sizes.

9-1-2 `complex128`

The same analysis has been done for `complex128` input values. As the results are very similar, they have been moved to the Appendix C.

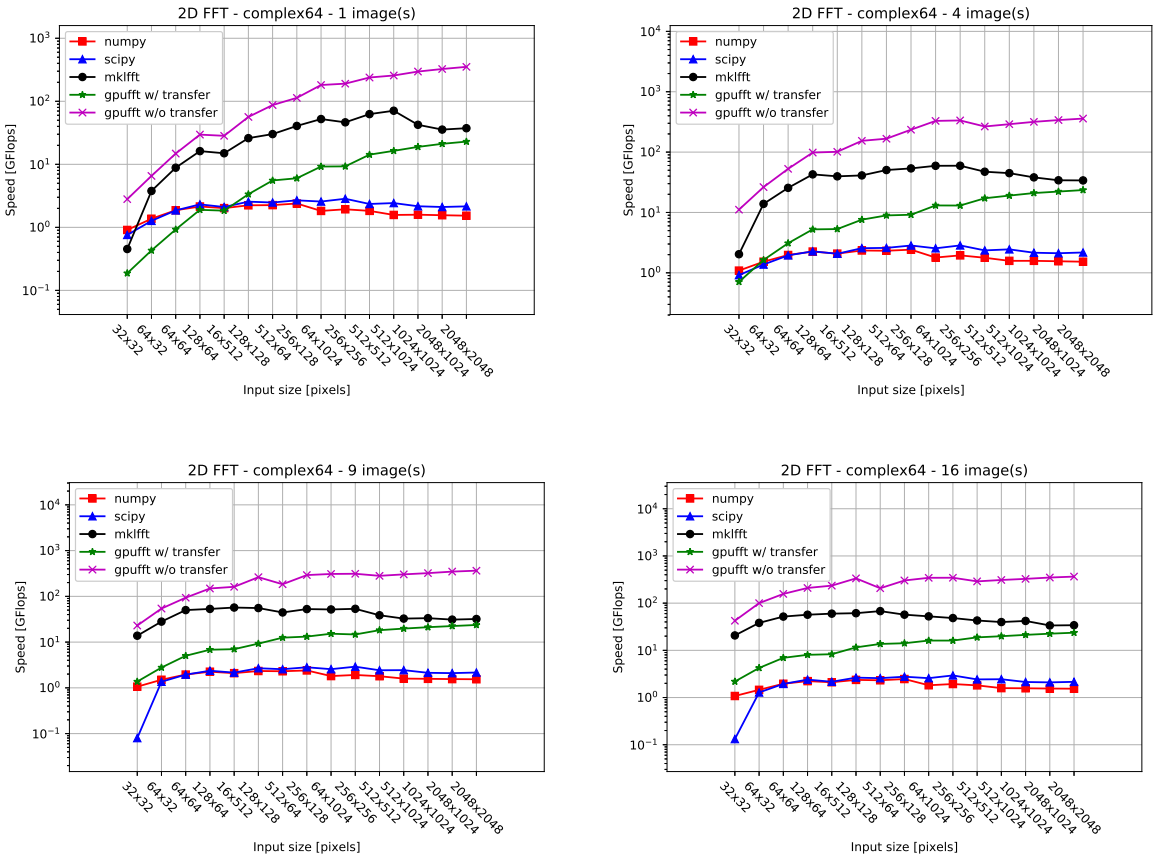
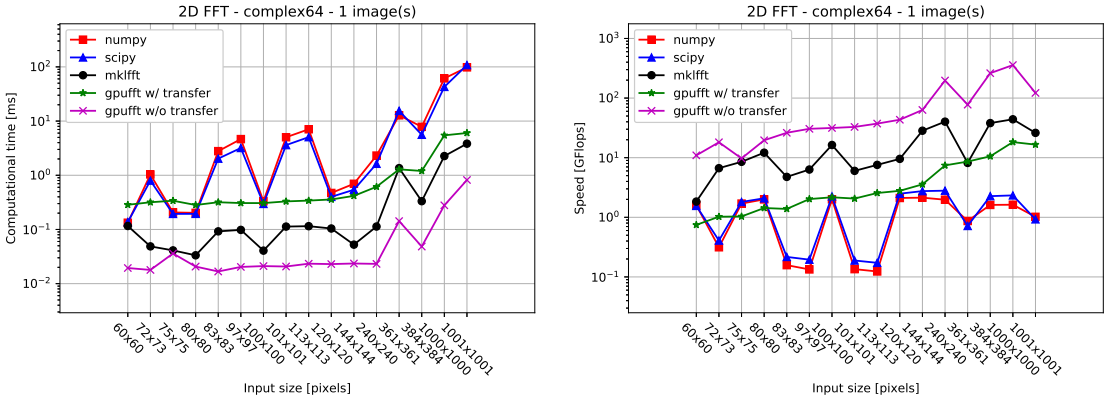


Figure 9-3: 2D-FFT FLOPS comparison of the different implementations with a varying number of input images.



(a) Computational time. **(b)** FLOPS count.

Figure 9-4: Performance comparison for solving one FFT of `complex64` values with non power of two input sizes.

9-2 TIP comparison

In this section, the computational time analysis is going to be done for the TIP algorithm and its different implementations:

- `Original TIP`: Explained in Algorithm 1 (see Chapter 3).
- `mkltip`: Implementation of Algorithm 2 explained in Chapter 7
- `gputip`: Implementation of Algorithm 2 explained in Chapter 8

For the previous implementations, there are several parameters that will be compared:

- *Precision*: Input arrays of *single* (`float32`) and *double* (`float64`) precision.
- *Input size*: A set of different sizes (of square images) that are divisible by 2.
- *Number of images*: The algorithms are tested for 4 and 9 input images.
- *Number of iterations*: Also a different number of iterations are used because some implementations can take advantage of the hardware architecture.

9-2-1 `float32`

First, the TIP algorithm is going to be tested with `float32` images and a different combination of number of images and number of iterations.

First results Figure 9-5 illustrates the expected behavior of the different implementations. Firstly, the `Original TIP` is almost always the slowest implementation because it was not optimized with the `mklfft` and the C-compiled functions. The other CPU implementation, `mkltip`, performs as expected, achieving an average computational time decrease of ∼x10 in comparison with the `Original TIP`.

About the GPU implementation, `gputip`, it starts being the slowest of the algorithms (due to the memory transfers between the host and device) but, with an increasing image size, the parallelized TIP functions begin to outperform the CPU implementation. It is expected to observe an even better performance when the number of iterations grows.

Analysis for 4 input images For a fixed number of input images (4 in this case). The different implementations are run with a varying number of iterations: 10, 25, 50 and 100. Figure 9-6 illustrates the comparative for the different iteration number. It is observable that both `Original TIP` and `mkltip` show the same linear behavior in all cases. However, this is not the case for `gputip` because with an increasing number of iterations, the increase in computational time is smaller in comparison. As it was mentioned before, this is the expected result because the memory transfer time is fixed for the number of images while the real TIP computation takes more due to the higher number of iterations.

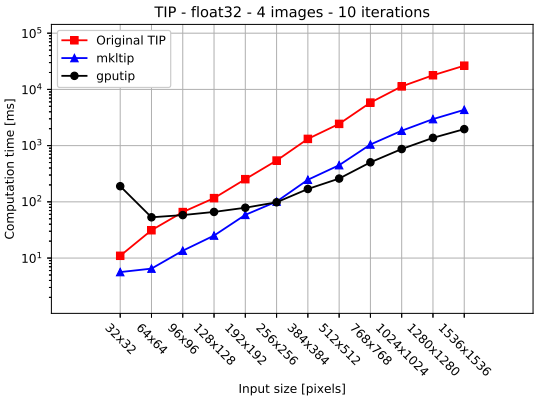


Figure 9-5: Comparison of the different TIP implementations using 4 input images.

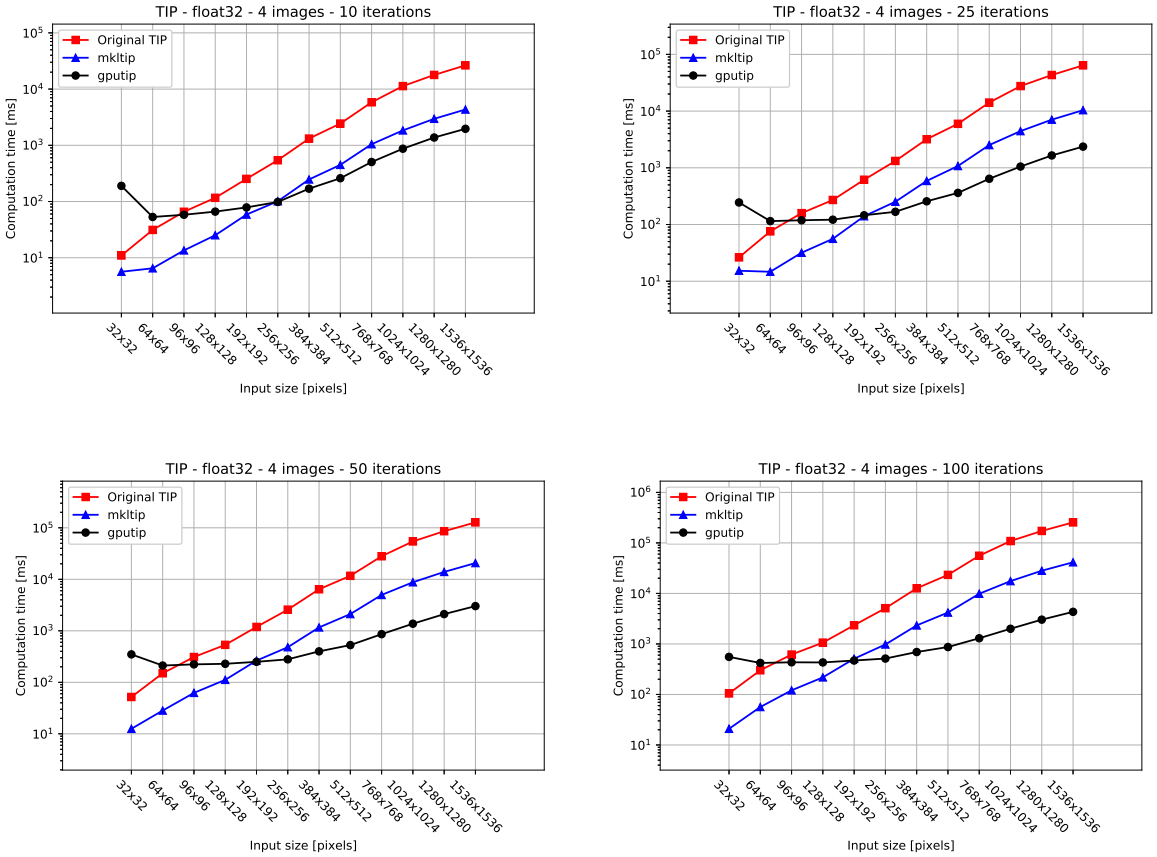


Figure 9-6: Computational time comparison using 4 input images.

Analysis for 9 input images The same analysis is done using 9 input images. As the results are very similar, they have been moved to Appendix C.

9-2-2 `float64`

The same analysis has been done for `float64` input values. As the results are very similar, they have been moved to Appendix C.

9-3 Summary

In this chapter, a computational speed comparison has been made for the different FFT implementations (either already available in Python or using the developed libraries) and for the TIP algorithm implementations.

As it was expected, the `mklfft` library outperforms in all scenarios the standard Python libraries (`numpy` and `scipy`) due to the multi-thread capabilities. In addition, it has been shown that the data transfer between CPU and GPU reduces drastically the performance of the `gpufft` library, making it useful only when more than FFT (or other computations) are carried out in the GPU.

Regarding the TIP algorithm implementations, there is a notable speed up using the `mkltip` library (around 10 times faster) and different performance for `gputip` depending on the input size, number of inputs and number of iterations. The `gputip` library shows the highest performance (around 10 times faster than `mkltip`) when a high number of iterations is required in high number of images with a large size, because in that cases the transfer time is negligible in comparison with the computational time required.

Thanks to the use of a better CPU code implementation and the use of a GPU. The TIP algorithm can be run 10 or 100 (in certain situations) times faster respectively.

Finally, it has been shown that it is possible to achieve the object reconstruction with a computational time under 1 second for certain number of images and iterations. For small image size ($\leq 256 \times 256$), the `mkltip` is the fastest but, for bigger images, `gputip` shows a better performance.

Chapter 10

Optical Device Experimentation

This chapter is focused on the results obtained using the TIP algorithm in the optical system explained in Chapter 5. The idea is to measure the performance (the object and PSF estimation) of the system using the algorithm in different real situations (i.e., as a microscope or as a telescope).

10-1 Graphical User Interface

In order to do the testing a Graphical User Interface (GUI) was developed. An initial working GUI was provided by Dean to which certain modifications were performed, so the testing of the different implementations (CPU/GPU) with varying parameters could be done in an easy and fast manner.

Figure 10-1 illustrates this GUI where different numbers can be found:

1. Live feed of one of the sub-apertures of the system. As it has the same size as number 2 (TIP estimation), it is easy to compare both images.
2. Latest object estimated using the TIP algorithm.
3. Live feed of the 4 sub-apertures after being apodized and centered (more about this later). These images are the ones fed to the TIP algorithm.
4. Latest PSF estimation of the 4 input images.
5. Buttons used for turning on/off the camera and the TIP algorithm and for saving the 4 main images.
6. Control of the centering algorithm.
7. Control of the camera exposure, number of images to feed to TIP and the computer time (in ms) required for the last estimation.

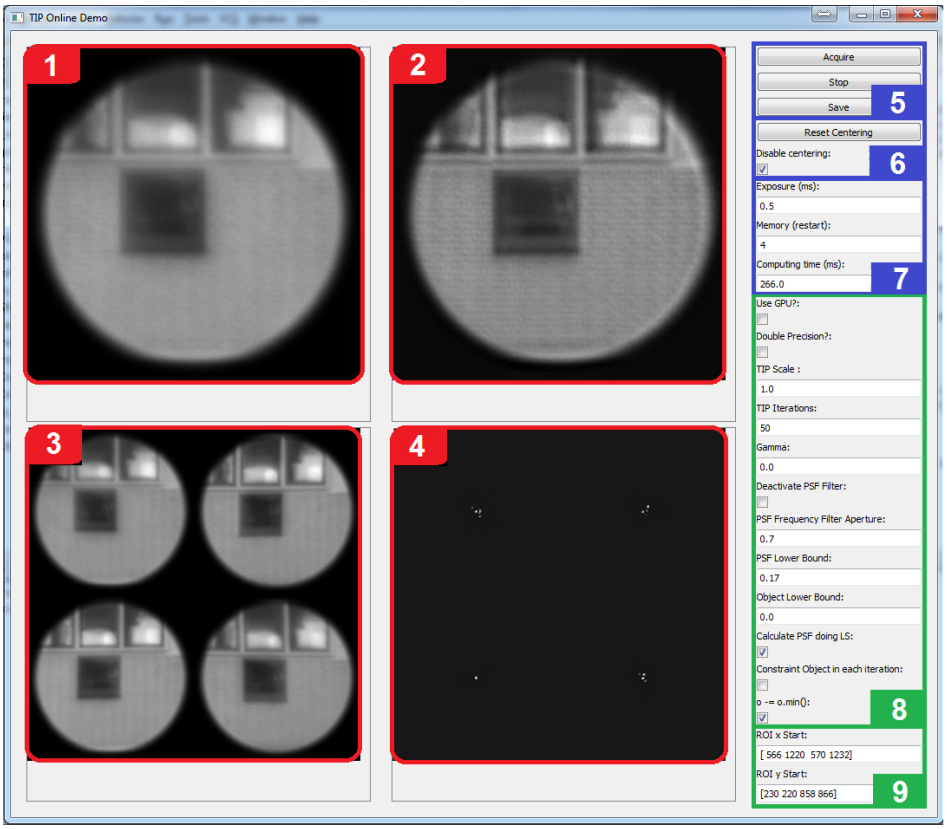


Figure 10-1: GUI used for testing the optical system.

8. List of the different parameters that can be modified of the algorithm.
9. Region of Interest (ROI) shows the actual image centers of each sub-aperture using the coordinates of the input image from the camera.

10-2 Image alignment

When running the algorithm, one may be tempted to use the whole size of the sub-apertures and let TIP center and fuse the images by itself (which it actually does). However, this might be a problem as shown in Figure 10-2.

One of the problems of the multi-aperture systems is its difficulty to obtain a perfect alignment of the lenses, meaning that in the image from the camera, the different sub-apertures are looking at the same object but with a slightly spatial offset (and sometimes even slightly different perspective). This difference in the spatial position (see Figure 10-2a) produces sub-images which show some extra areas of the object while others do not; in the end, the TIP algorithm fuse all the information available, producing the effect seen in the borders of Figure 10-2b.

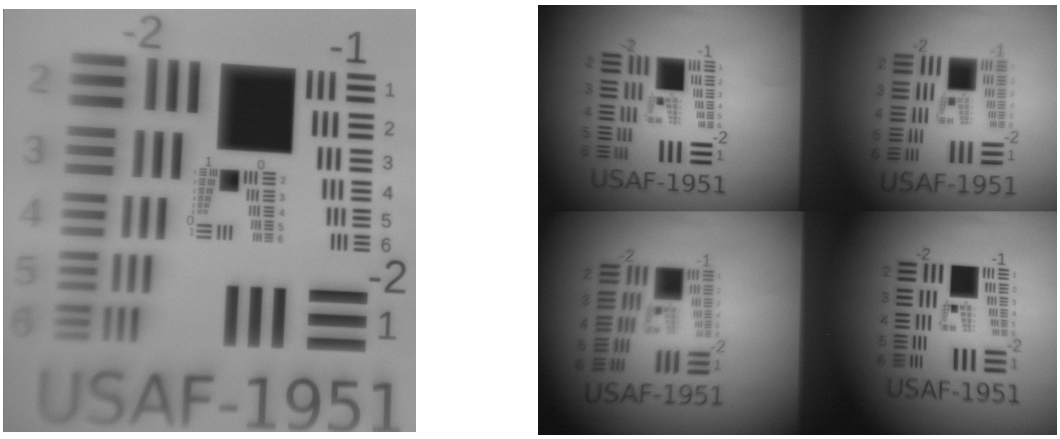
The way this is solved is by reducing the image size (cropping), apodize it for avoiding ringing in the edges and then using a centering algorithm (explained in detail in Appendix D so all sub-images fed to the algorithm shown the same areas of the object.

(a) Input unprocessed image from the camera. **(b)** TIP reconstruction.

Figure 10-2: Object estimation obtained when no alignment and apodization is done before feeding the images to the TIP algorithm.

10-3 Full-aperture vs multi-aperture

In this section, the images obtained by a full-aperture system and the multi-aperture one are compared. First, the object is a printed version of the USAF target for which a diffraction limited image (or, at least, the sharpest image obtained) is shown in Figure 10-3.

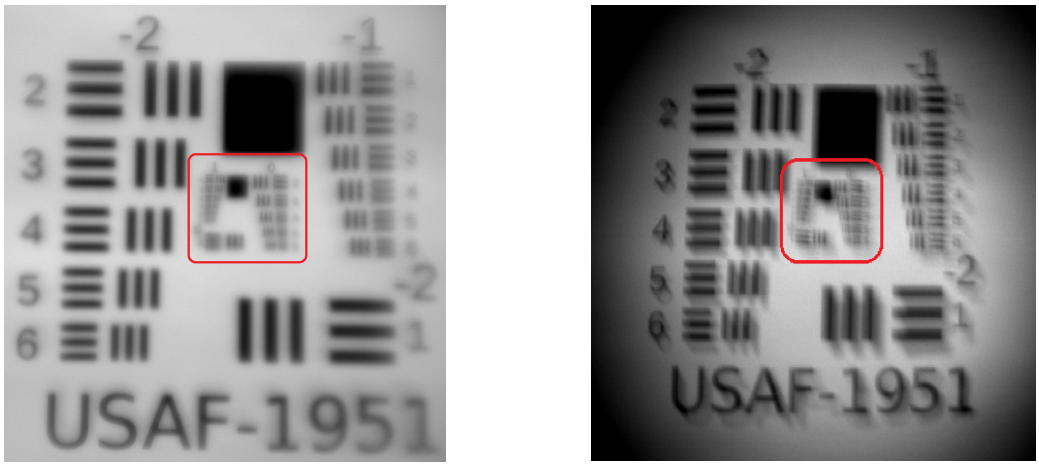


(a) Full-aperture. **(b)** Multi-aperture.

Figure 10-3: Sharpest images obtained of the USAF target for the different optical systems.

Figure 10-4 illustrates both images recorded by the different systems. It is observable that, although the full-aperture image shows an overall less aberrated image, one of the sub-images allows the distinction of the smallest features (within the red rectangle). Although the exact same aberration between the two images was not achieved (because changing between systems imply assembling and disassembling certain lenses), it proves the point that for some aberrations the multi-aperture system shows less distortion and, hence, keeps more details

than the full-aperture system (keeping in mind the differences in resolution, which is 3 times in this case).



(a) Full-aperture of 1000×1000 pixels. **(b)** Multi-aperture. Sub-image of 330×330 pixels.

Figure 10-4: Aberrated images obtained of the USAF target for the different optical systems.

Finally, using the TIP algorithm for the multi-aperture system. It is possible to obtain a sharper image (see Figure 10-5) than a full-aperture system affected by a static aberration (in which a MFBD algorithm can not be applied).

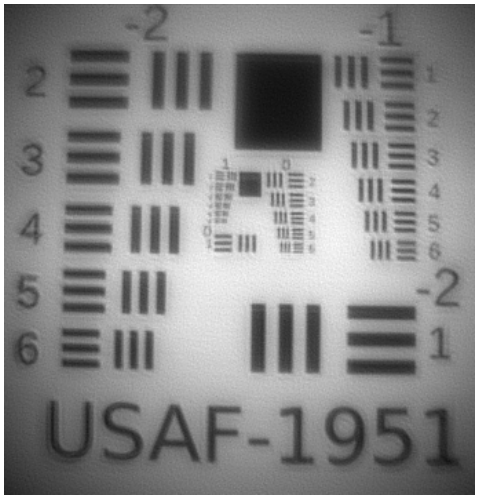


Figure 10-5: TIP estimation for the images in Figure 10-4b.

10-4 Computational speed

One of the main goals of this thesis is to obtain a fast enough object reconstruction, ideally below 1 second. The previously introduced GUI is used to measure the computational time

Table 10-1: Computational time of the TIP algorithm for CPU and GPU run in the previous GUI.

Size [pixels]	**CPU Time [ms]**			**GPU Time [ms]**		
96 × 96	16	37	146	86	182	630
128 × 128	25	63	250	92	186	650
192 × 192	62	146	558	119	243	818
256 × 256	112	261	1047	136	260	848
384 × 384	359	687	2413	223	378	1049
512 × 512	619	1228	4629	306	443	1194
768 × 768	1382	3073	10586	542	703	1580
1024 × 1024	2464	4916	17910	903	1113	2148
Iterations	*10*	*25*	*100*	*10*	*25*	*100*

required when using the CPU and GPU with a varying number of aberrations. Table 10-1 shows a list with the measured computational times.

In addition, Figure 10-6 illustrates the previous measured times in a graph. It is observed that for images of 256 × 256 pixels or below, the CPU shows the fastest performance. However, with bigger images, the GPU is recommended. Also a horizontal line at 1 second mark is drawn to show when this goal is achieved.

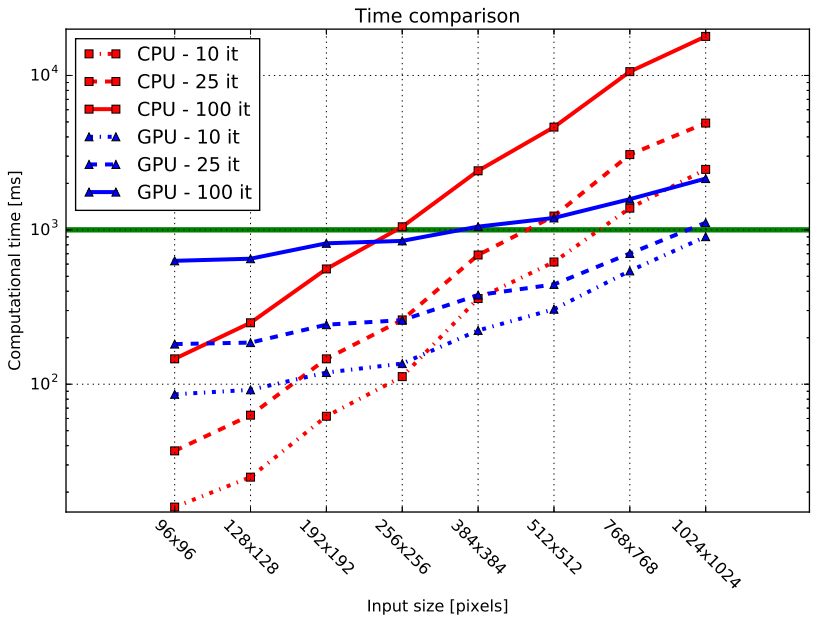


Figure 10-6: Time required in the GUI for running the TIP algorithm in the CPU or GPU with a varying number of iterations. The green horizontal line indicates the goal of 1 second of computational time.

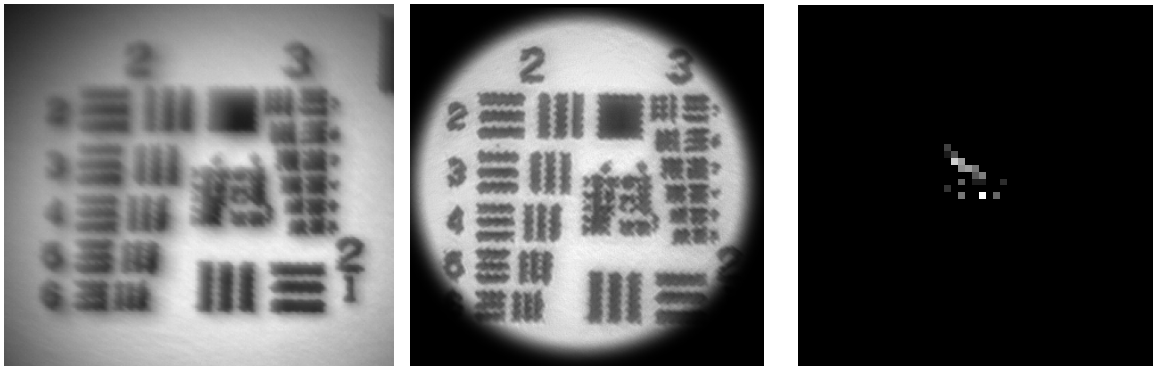
10-5 Microscopy and telescope applications

The multi-aperture system has been tested acting as a microscope (looking at a close object through a static aberration) and telescope (looking at a far object through atmospheric turbulences).

10-5-1 Microscopy

In order to test the behaviour as microscope, the optical system is installed in the optics lab, where it is easier to perform the alignment and to introduce aberrations. Aberrations are produced using a transparent disk where a hair spray has left some rests; afterwards, this disk is set at the pupil plane before the partitioned aperture.

Again, a printed version of the USAF target is used as sample with the optical system acting as a microscope. Figure 10-7 shows the result obtained, it is remarkable how the deconvolved object presents sharper features (it is possible to see the ink flow of the printer) and a more uniform illumination (the vignetting of the different sub-apertures is averaged).



(a) Best input image. **(b)** Object reconstruction. **(c)** PSF estimation.

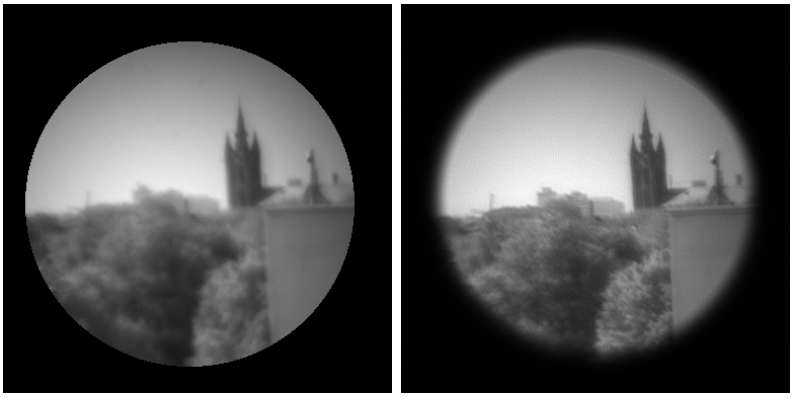
Figure 10-7: Performance of the optical system with the USAF target.

10-5-2 Telescope

Using the system as a telescope poses certain problems: the extra difficulty of alignment because it is on a tripod and also the lack of strong atmospheric turbulences during the time of testing. However, the object estimated still shows improvement with respect to the unprocessed images. Figure 10-8 illustrates the performance when looking through the window with very low aberrations. As one of the properties of this device is to obtain several images with different aberrations of the same object, it is possible to reconstruct dynamical objects. Although it can not be seen in Figure 10-9 but those images where taken a windy day when the trees were moving. Anyway, the reconstructed object is sharper than the input image in both trees and background buildings.

(a) Best input image. **(b)** Object reconstruction. **(c)** PSF estimation.

Figure 10-8: Performance of the optical system looking through the window.



(a) Best input image. **(b)** Object reconstruction.

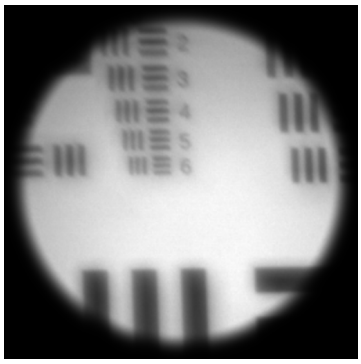
Figure 10-9: Performance of the optical system looking at the Oude Kerk from 3mE during a windy day.

10-5-3 SR capabilities

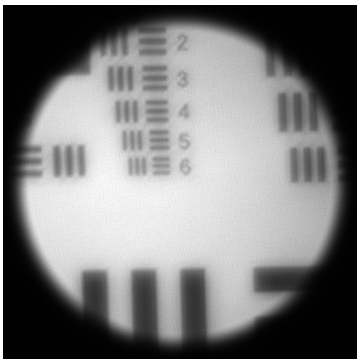
As it was introduced in Chapter 4, the TIP algorithm now supports SR. In addition, it was claimed that this increased the retrieved information from the input image set.

In order to do a fair comparison between the reconstructions obtained from TIP with and without SR, the optical system is set in the optics lab, where the same aberration is applied when looking at the USART target. As only the SR capabilities are being tested, the aberrations chosen are very mild, basically a small defocus and distortion produced by a poor alignment of the setup.

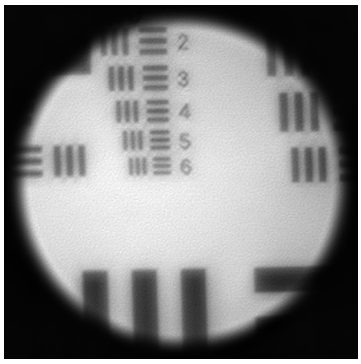
Figure 10-10 illustrates the different images obtained. The best multi-aperture image and the TIP reconstruction are upscaled linearly in order to match to output size of the SR-TIP reconstruction.



(a) Best multi-aperture image. Upscaled from 256×256 to 512×512 pixels.



(b) TIP reconstruction. Upscaled from 256×256 to 512×512 pixels.

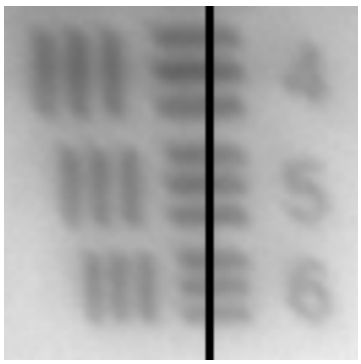


(c) SR-TIP reconstruction. Original size: 512×512 pixels.

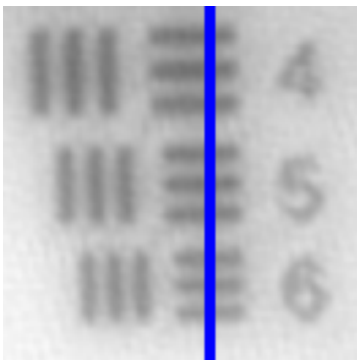
Figure 10-10: Comparison between the best multi-aperture input image and the two TIP reconstruction with and without SR.

It is observable that even with low aberrations TIP achieves a less blurred object reconstruction. Moreover, the SR-TIP estimation looks sharper.

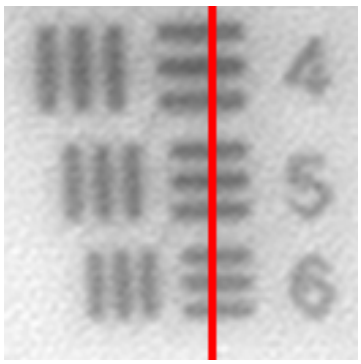
In order to check which reconstruction obtained a better result, a small area in the center of the image is zoomed (see Figure 10-11). In this area, the values of the vertical line are plotted Figure 10-12 for further analysis.



(a) Best multi-aperture.



(b) TIP.



(c) SR-TIP.

Figure 10-11: Zooming in the central area.

As the drawn lines goes through an area with several edges, one way of comparing the reconstruction is to measure the variance or standard deviation. As most aberrations produce an spread of the image, this average the areas with strong edges, so a low quality image is expected to have a low standard deviation while a good reconstruction has a higher one. This is observed in Figure 10-12, the original input image shows a flatter profile (hence, a smaller STD), while the TIP reconstructions manage to recover part of those edges. Finally, it can be seen that the SR-TIP obtains a better reconstruction because the STD is higher than TIP, meaning that more information has been retrieved and that the modified algorithm perform as expected.

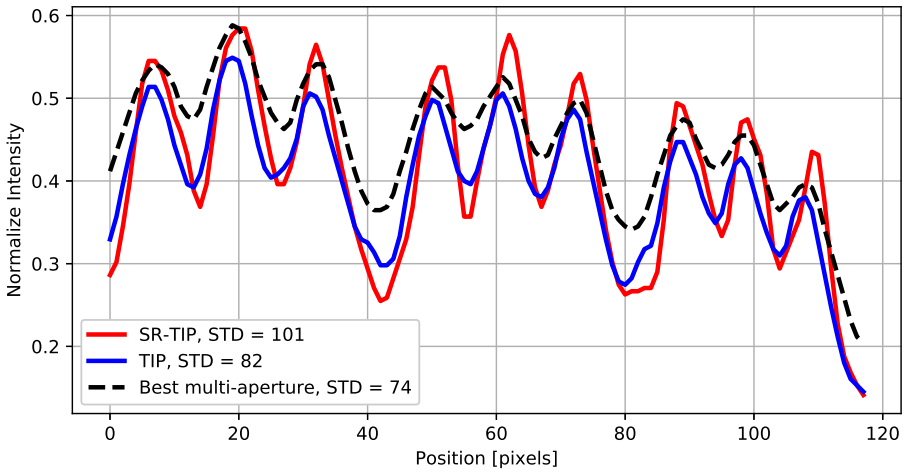


Figure 10-12: Comparison of the differente intensity levels

10-6 Summary

In this chapter, the multi-aperture optical device designed in Chapter 5 is tested along the modified version of TIP (see Chapter 4) in different conditions.

Firstly, a GUI was modified from a previous one, in order to analyze in real-time the effects of the different parameters in the object reconstruction.

Once the GUI is available, the multi-aperture system is compared to the full-aperture one, showing the same results as in Chapter 6. In this case, the best multi-aperture image shows higher details than the full-aperture one despite the lower resolution.

It has been proved that a real-time object reconstruction is possible, showing a computational time of less than one second for several image sizes and number of iterations.

The optical device has been tested in two different modes showing satisfactory results:

- Microscope: Looking at an object through static aberrations.
- Telescope: Looking through dynamic aberrations to dynamical objects.

Finally, the last test comprises a reconstruction comparison between TIP and SR-TIP, in order to see if more information can be retrieved with the SR method. In this case, the addition of the SR capabilities to TIP increased the information retrieval from the input images, increasing the sharpens of the reconstructed image.

Chapter 11

Conclusions and Future Work

The main goal of this thesis is to build a fully operational prototype of a non conventional optical systems that makes use of the TIP algorithm. This system has been proved to work as expected, showing novel characteristics in comparison to other standard optical systems. However, it is possible to depict more specific conclusions that arises throughout the work done in the thesis.

11-1 Conclusions and discussion

In this section several conclusions of the different parts of the thesis are presented with a discussion.

11-1-1 TIP

TIP is a novel and very promising algorithm that allows the object and PSF reconstruction from a set of aberrated images. It has very interesting properties:

- Due to the use of the FFT, the deconvolution process can be achieved doing point-wise operations in the pixels, this is translated into a low computational power requirement in comparison with other methods.
- The algorithm shows a fast initial convergence, as shown in Chapter 6. During the first iterations is when the highest decrease of error can be found, making possible to run the algorithm with a small number of iterations and even get a good reconstruction.
- As it uses the LS solution for estimating the object, it acts as the optimal noise removal filter for Gaussian noise. In addition, the algorithm can be expanded to increase further its noise robustness using the LS solution for estimating as well the PSF and also constraining the object in each iteration.

- SR is a technique that can be implemented in a very straight forward manner into the algorithm. This technique was proven to increase the retrieved information from the input images, allowing the increment in fine details of the final reconstruction.
- The TIP algorithm, is an optimization method which does not explicitly converges to the global minima of the solutions space. The way of solving this problem, implemented in the algorithm, is to randomly vary the OTF parameters in each iteration in order to allow the escape from local minima. This technique do not ensure the converge to the global minima but allows a better reconstruction by finding better local minimas.

11-1-2 Algorithm implementation

In order to produce a working prototype of the optical system, it is necessary to implement efficiently the TIP algorithm. The implementation developed is focused on the computational time required to obtain a solution from the algorithm, this allows the use of TIP in a real-time environment. From this process, the following conclusions arose:

- Python is powerful programming language, being very useful for testing ideas in a fast manner, but it does not support natively high-performance code. However, this can be solved using one of the several libraries that allows: the development of C-compiled code, multi-core programming, accessing pre-compiled high-performance libraries and GPU programming.
- It has been found that the most famous Python libraries for calculating the FFT suffer from the limitations imposed by the Global Interpreter Lock (GIL) (see Chapter 7), not using completely the hardware of modern CPUs. Although this has been solved with the development of the `mklfft` library.
- When doing repetitive specific mathematical computations, it is still possible to gain a speed up if, instead of using the optimized available mathematical libraries (see `numpy` and `scipy`), this code is C-compiled before calling (using the `numba` library in this case). This lead to the development of the `mkltip` library.
- Due to the large amount of point-wise operations required by the TIP algorithm, it is possible to take advantage of the parallel nature of a GPU in order to speed up the computation. In this case, Python can be used as host commanding which operations the GPU needs to do; this is known as General-Purpose computing on GPU (GPGPU) and led to the development of the `gputip` library.
- Analysing the different FFT implementations, the use of the `mklfft` library showed a speed increase of 10 times on average when comparing it to the standard Python libraries. In addition, the use of the GPU for only doing the FFT displayed that the data transfer between the *host* and the *device* limits the performance. In this case, it is recommended to do several different computations in the GPU in order to overcome this limit.
- When analysing the TIP implementations, it was observed a exponential increase in computational time when using the CPU with different input image sizes. However, the GPU showed a smaller increase, making it ideal for using big images sizes and a high number of iterations.

11-1-3 Optical system prototype

Once that the optical system has been designed, assembled and tested, it is possible to draw some conclusions from its behaviour and performance:

- During the numerical simulations phase (see Chapter 6), it was shown that a partitioned aperture helps to imaging through areas with a lower phase aberrations, allowing the recording of sharper images. However, the drawback of this approach is the loss in final resolution.
- Analysing the parameter α (or PSF lower bound), a trade-off was observed between when a low value of α is used; it improves the PSF estimation but decreases the convergence properties of the algorithm.
- It was found that an increasing number of input images, also increases quality of the object reconstruction. This was expected because the more different input images are used, the more information the whole set carries and, thus, more information is retrieved.
- The γ parameter, used for increasing the object estimation performance by impeding the algorithm to stay in a local minima, was proved to work. However, it was observed that too low values do not produce the effect desired and that, if the value is too high, it leads to a divergent algorithm. In addition, due to the stochastic nature of this solution, it is not possible to determine when the maximum information has been retrieved.
- The prototype has been tested in several real situations, being able to reconstruct the object satisfactorily from only 4 input images in less that 1 second (in most situations).
- Using the SR capabilities of the algorithm helps to retrieve more information from the input images and makes possible to retrieve a sharp object of the same size as if the system made use of a full-aperture. However, the result is not always acceptable because this increases the complexity of the algorithm when it is already trying to solve a bilinear ill-posed problem.
- The final conclusion is that this system allows the object reconstruction with only one frame (that it is divided by several apertures, containing different information). This systems can be useful when looking at continuously changing objects that impedes the registration of several images with different aberrations in a full-aperture system. In addition, there is also the case in which the aberrations are static were this system allows the registration of one image per aperture with different aberration, enough to use a Multi-Frame Blind Deconvolution (MFBD) method such as TIP.

11-2 Future work

Although this is a satisfactory proof of concept, future work is necessary in order to improve the performance and robustness of the system.

For the TIP algorithm, future contributions could be:

- Try to eliminate the α parameter without loosing the convergence properties. Dean is already working on it using a change of basis that allows the shaping of the PSF to the desire size without the need of thresholding (only the positivity constraint is used).
- It would be mathematically valuable to proof the convergence of the algorithm with the γ value.
- More methods for escaping from the local minima of the optimization algorithm should be investigated.
- In each iteration of the algorithm, a new object estimation is obtained from the Least Squares (LS) solution. An interesting modification for lowering the number of computations would be the use of a Recursive LS scheme.

Regarding the implementation of the code:

- Python has been proved to be very versatile for quick developing of ideas,. However, now that the TIP algorithm is more mature and adapted to an specific system, the next step would be to implement it in a language that can offer better performance, such as C/C++, for the CPU.
- In addition, as the `gputip` was developed in Python using a specific library which does not support all CUDA features, the next step is to implement the algorithm in CUDA C/C++, where the API fully supports all capabilities.

Finally, the future work proposals for the optical system are:

- Analyse the performance of the system for different aperture segmentations (i.e., varying number of division, different lens position, etc).
- As the prototype has been built using the optical lab equipment, a future improvement would be to make a specific mechanical design so it gets closer to a final product structurally more stable. Two designs could be done, one for using the system as a telescope and other for using it as a microscope.

Appendix A

Appendix: Numerical Simulations Extended

In this appendix, the rest of the figures comparing the TIP performance in numerical simulations are shown. The same conclusions found in Chapter 10 are applicable to the following figures.

A-1 Numerical simulations

A-1-1 PSF lower bound analysis

See Figure A-1 for the comparison for the Lenna image and Figure A-2 for the error evolution.

A-1-2 Number of images analysis

See Figure A-3 for the comparison for the USAF image and Figure A-4 for the error evolution.

A-1-3 Noise analysis

See Figure A-5 for the comparison for the USAF image and Figure A-6 for the error evolution.

A-1-4 Influence of *gamma*

See Figure A-7 for the comparison for the Lenna image and Figure A-8 for the error evolution when $scale = 1$.

See Figure A-9 for the comparison for the Lenna image and Figure A-10 for the error evolution when $scale = 2$.

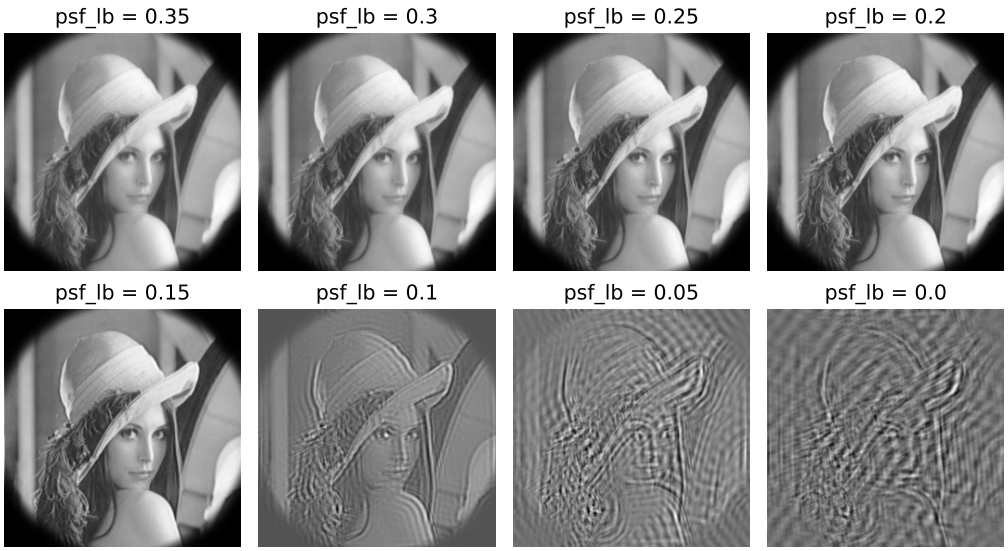


Figure A-1: Comparison of the different estimations obtained for a varying value of the PSF lower bound for the Lenna image.

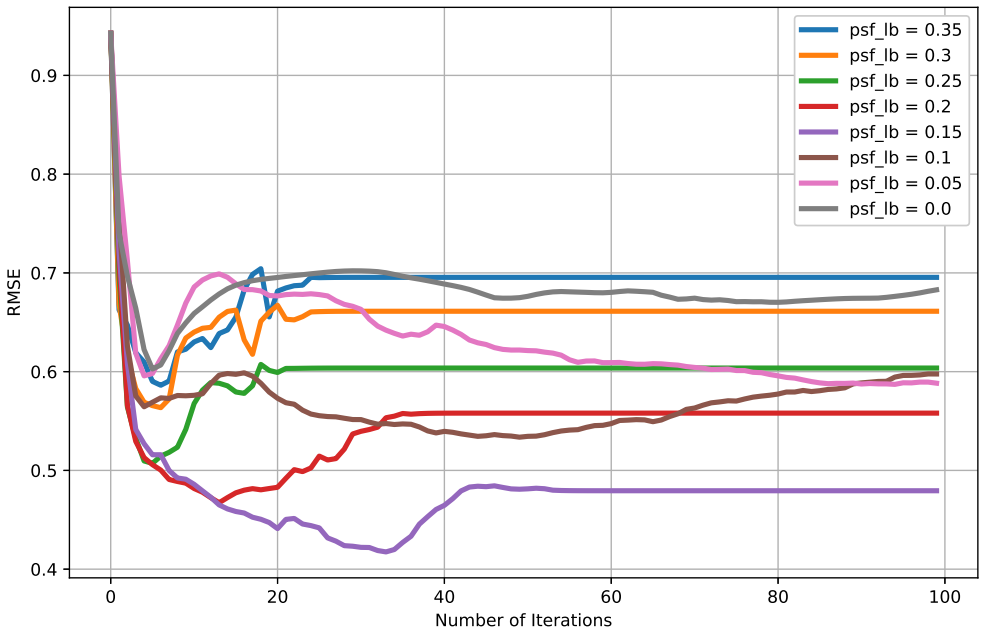


Figure A-2: Comparison of the error evolution obtained for a varying value of the PSF lower bound for the Lenna image.

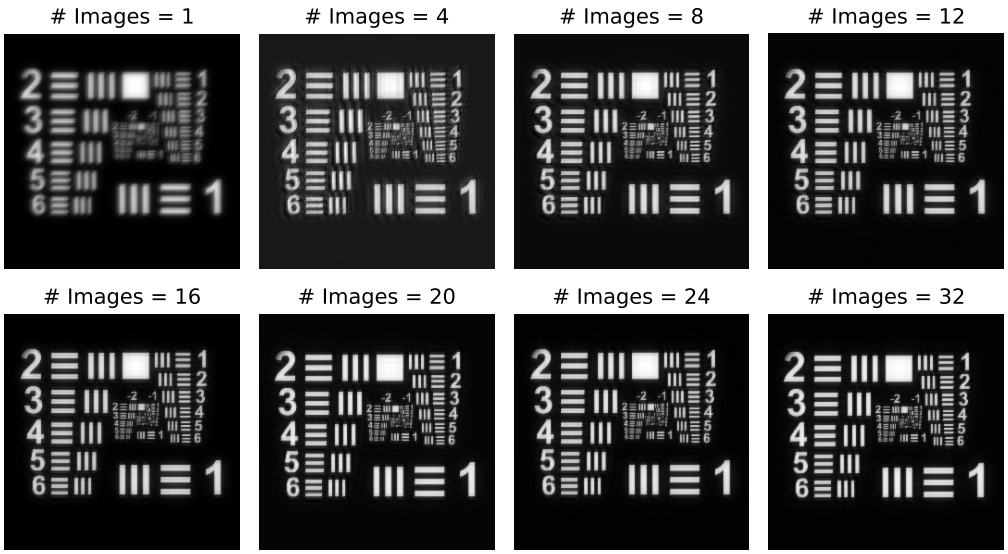


Figure A-3: Comparison of the different estimations obtained for a varying number of input images for the USAF image.

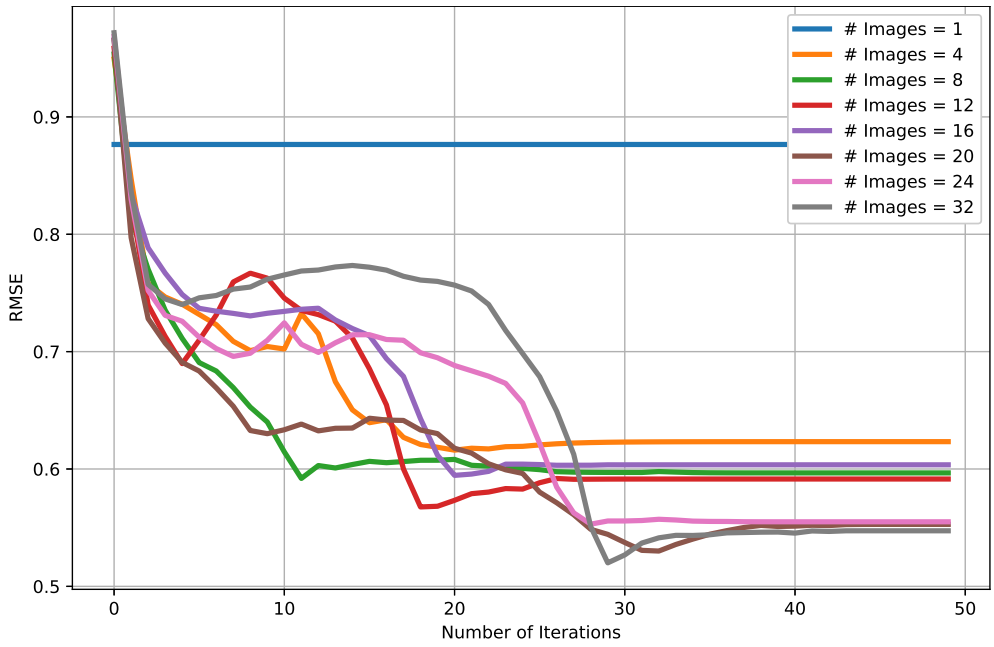


Figure A-4: Comparison of the error evolution obtained for a varying number of input images for the USAF image.

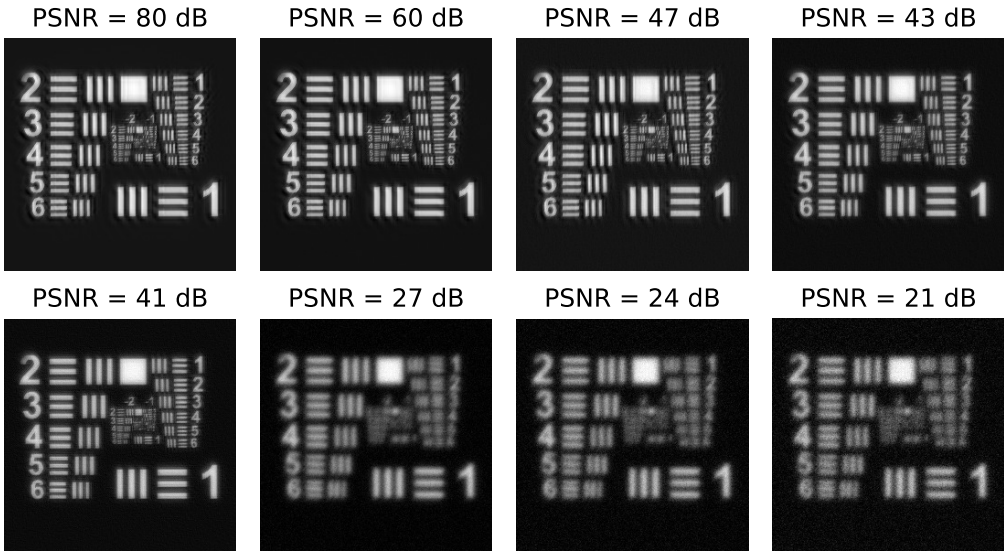


Figure A-5: Comparison of the different estimations obtained for a varying noise level for the USAF image.

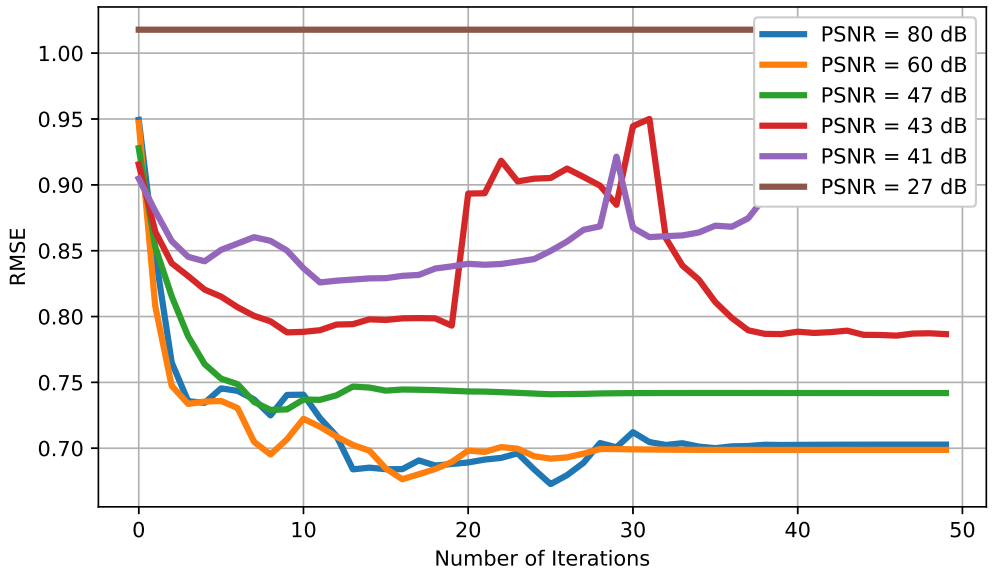


Figure A-6: Comparison of the error evolution obtained for a varying noise level for the USAF image.

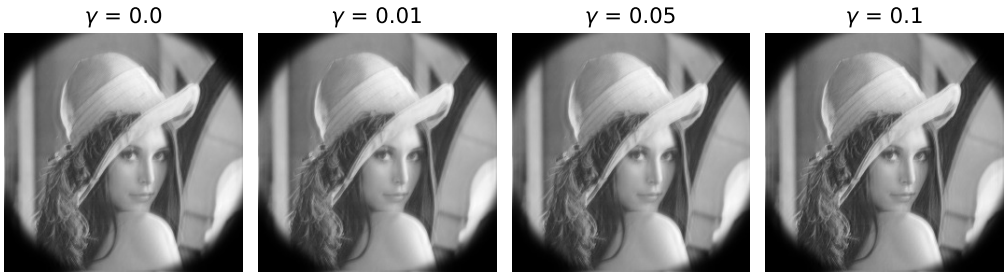


Figure A-7: Comparison of the different estimations obtained for a varying γ value for the Lenna image with *scale* $= 1.0$.

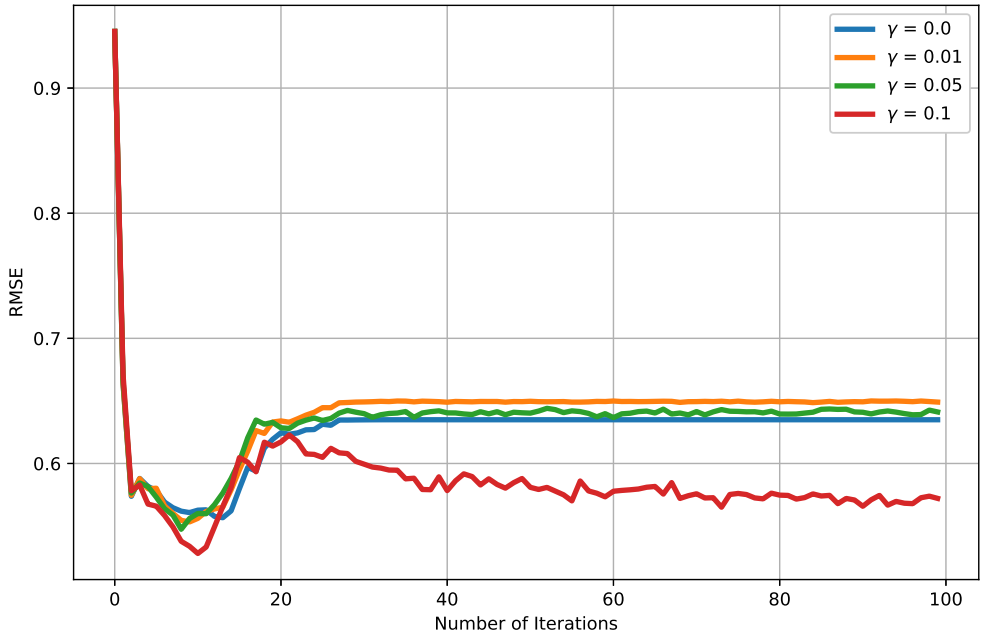


Figure A-8: Comparison of the error evolution obtained for a varying γ value for the Lenna image with *scale* $= 1.0$.

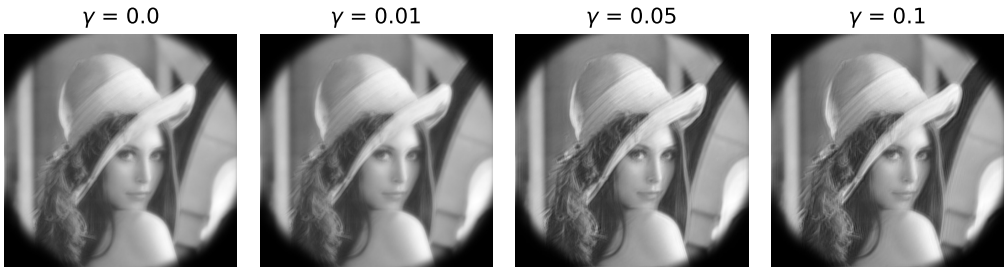


Figure A-9: Comparison of the different estimations obtained for a varying γ value for the Lenna image with *scale* $= 2.0$.

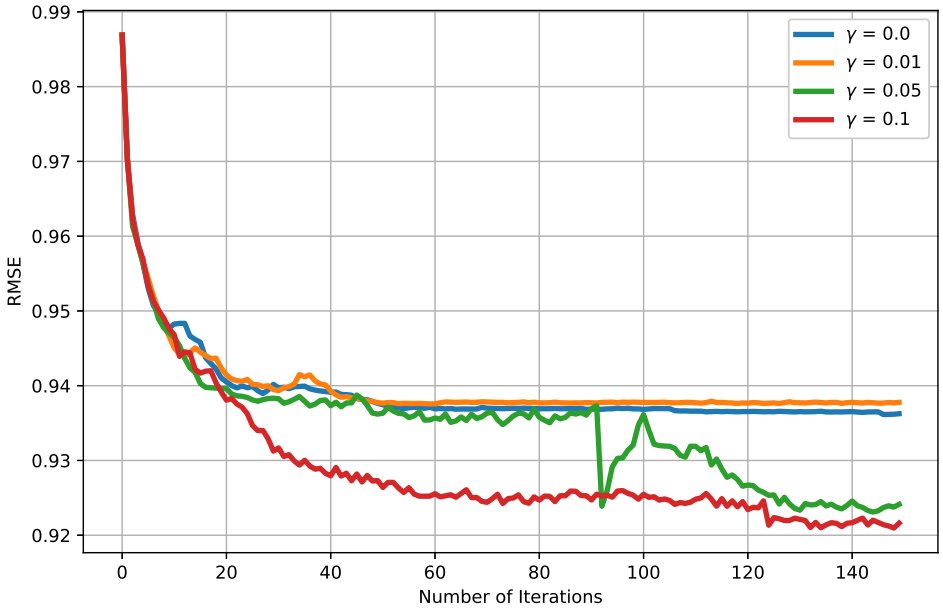


Figure A-10: Comparison of the error evolution obtained for a varying γ value for the Lenna image with *scale* $= 2.0$.

Appendix B

Appendix: GPU Programming

In this appendix a brief introduction is made on how to use a GPU for increasing the computational speed of a highly parallelizable code.

B-1 Introduction to GPU programming

Performance In order to have a high performance on a GPU it is necessary to *hide* the latency making use of the parallelization properties. This implies having all cores sufficiently busy doing computations such that there is enough time to access the memory (which is usually slow). Due to the latencies for accessing the main memory and the inter processor communications it is usually better to recompute certain values than fetching them.

Metrics There are several ways of measuring the performance of the GPU. Usually the maximum number of Floating Point Operations per Second (FLOPS) is a good estimate of the maximum computational output; this is achieved when all cores are busy all time. However, the GPU memory (GDDR) can be a bottleneck if it is not able to transfer the data on time; a memory with a high a low latency and high bandwidth is desired.

How the GPU is used When doing GPGPU, the CPU is still the main control unit, meaning that the GPU is used as a special processor for running operations in parallel. The CPU is in charge of controlling the code run on the GPU and the communications with the other components via the PCI-E bus (this is known as heterogeneous programming, see Figure B-1). In order to have the desired performance, communications between the host (CPU) and device (GPU) must be kept at a minimum. The idea is to only use the GPU when the computational speed up is higher than the time spent sending data and instructions back and forth. Figure B-2 shows this data flow.

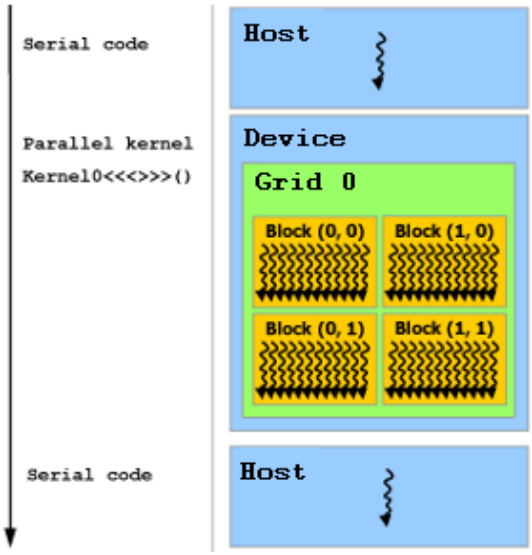


Figure B-1: Explanation of heterogeneous programming. Image credits: NVIDIA

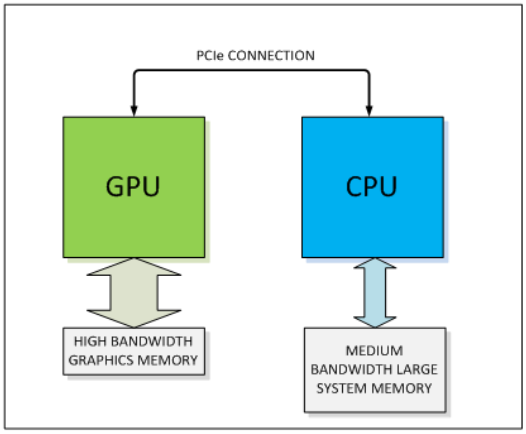


Figure B-2: Communication bus between the host and device. Image credits: NVIDIA

B-2 CUDA

Compute Unified Device Architecture (CUDA) [50] is the free programming interface provided by NVIDIA but only available for their own hardware, although it is the best development software available nowadays.

B-2-1 Architecture

In a CUDA environment, there are certain key-words that are used constantly.

Host and device For referring to the CPU and its memory, the term *host* is used. The counterpart for the GPU is *device*.

Streaming Multiprocessor (SM) A CUDA enabled GPU (a GPU that can be used with CUDA) has a certain number of SM which has a shared memory area. Inside a SM there is a Single Instruction Multiple Data (SIMD) processor (a processor that executes the same operation at the same time for different data sets). The SIMD processor contains several scalar processors that can work in a multi-threat fashion. Figure B-3 illustrates the architecture inside a SM.

Scalar Processors (SP) The basic component that does the arithmetic calculations are the SPs. Each one of them can perform a floating point operation each one or two clock cycles (depending of the single or double precision floats are used).

Figure B-3: SM close up, it is composed by several scalar cores. Image credits: NVIDIA

Kernel As explained before, the CPU is in command of the computations done by the GPU (with CUDA as interface). The computations desired are written in CUDA kernels. This kernels are the functions launched in the GPU that are parallelized (executed in several cores at the same time), meaning that each core does the same computations but with different data (a SIMD machine). Kernels have certain properties that makes them different to standard functions written for a CPU: only GPU memory is accessed and no value is returned.

B-2-2 Execution hierarchy

Threads and its groups The execution of a kernel can be seen from different level perspectives. A thread is the basic unit that executes a kernel on a SP (which has its own private and local memory) and there can be many of them executing the same code but on different data. In addition, all threads can access the main memory (global memory) of the GPU. One layer above, several threads can be grouped into blocks, allowing the use of shared memory with other blocks. In CUDA, when a kernel is launched, it is specified the number blocks used and the number of threads per block. The blocks used can be executed in any order in the SM, the threats in that block are only executed in the same SM and the block can not be called again until all threads have finished computing. Finally, blocks can be grouped in grids, being each grid a physical core(s) of the GPU(s). Figure B-5 illustrates the grouping of execution hierarchy.

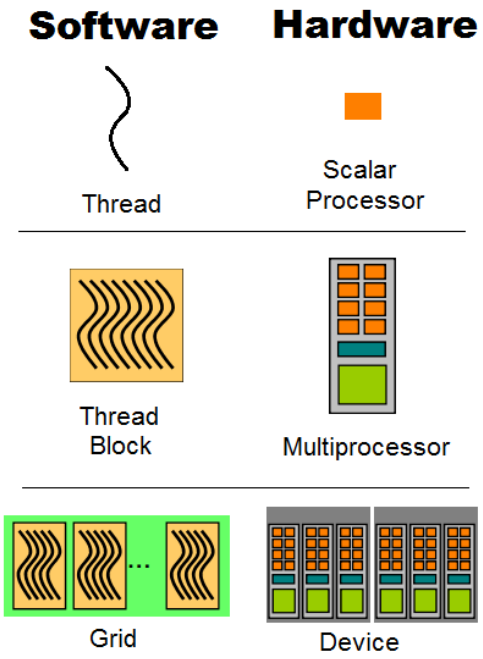


Figure B-4: Execution hierarchy from the software and hardware side. Image credits: NVIDIA

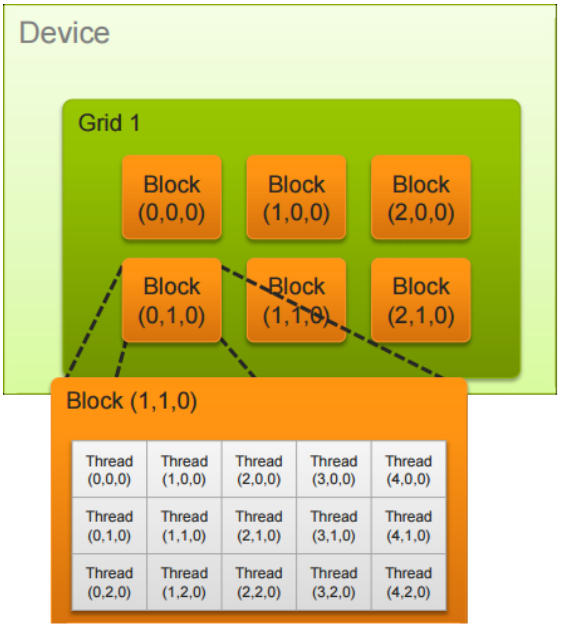


Figure B-5: Different thread grouping. Image credits: NVIDIA

Warps Each thread is not launched independently within a block but it is done in groups of 32 (warps), running the same code at the same time. It is advisable to use a number of thread per block multiple of 32 for achieving the maximum performance. In addition, when a thread within a wrap executes different code (probably due to a triggered condition), the program waits until all threads are done computing (even the divergent ones).

Thread accessing When a kernel is launched, several threads run the same code but with different data. One way of accessing the independent elements of the data for each thread is by using the global thread index. This index is usually calculated at the beginning of a kernel as seen in Figure B-6. As this index in unique, each data element will be loaded once.

B-2-3 Memory hierarchy

Each GPU has several memory levels which can not been accessed from all different components of the execution hierarchy. Figure B-9 illustrates the memory hierarchy in a GPU.

Global memory The large and slow memory located outside of the GPU in DDR chips is the global memory. It does not have any kind of cache and can be read and write indistinctly. In order to have the fastest accessing, it is required to do sequential and aligned 16 bytes reads/writes, this is known as coalesced accessing.

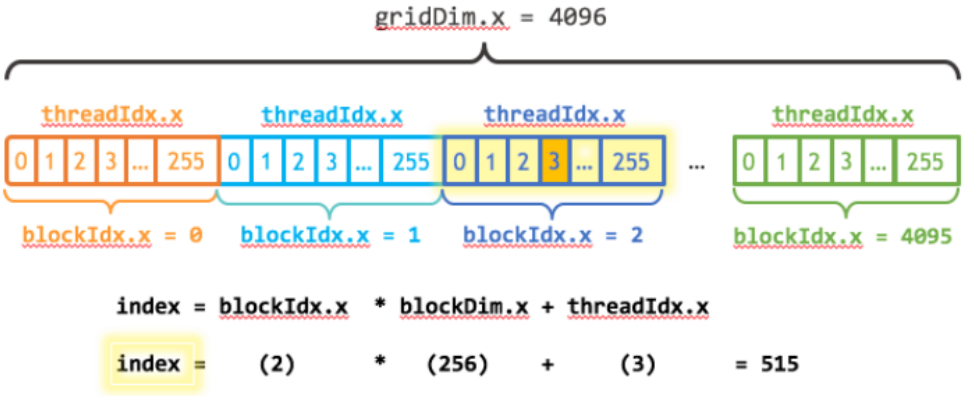


Figure B-6: Calculation of the global thread index. Image credits: NVIDIA

Texture memory When using the graphical capabilities of the GPU a texture memory is often used for storing common textures. Meaning that this memory is cache optimized for 2D accessing pattern.

Constant memory There is a special part of the memory that is used for storing the argument of the kernels and constant values, this is the constant memory. It is slow but it is cached.

Shared memory Each SM has its own memory, called shared memory. This memory can be accessed by all the threads actives in each block that are in that specific SM. It is an on-chip resource, so it is very fast to access although it does not have a big size.

Local memory Each thread can have a private memory for its own purposes when no more registers are available, the local memory. However, this memory is part of the global memory, so it is slow (and uncached) to access and it is automatically coalesced when used.

Registers The fastest (and scarce) memory available for a thread are the registers. They are located on-chip, with a specific number of them per SM.

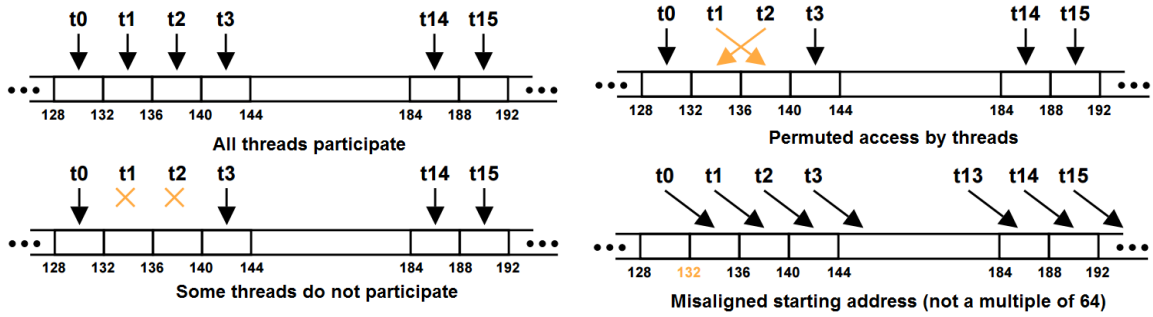
B-2-4 Coalesced memory access

Global memory holds the highest storing capacity for the GPU but has a high latency (slow fetching). In order to avoid as much as possible the performance bottleneck created by this memory it is necessary to maximize the bandwidth.

This memory is implemented with Dynamic Random Access Memory (DRAM) which uses a parallel process. Each time a location is fetched, consecutive locations are also accessed (coalesced access). This means that if the application retrieve data from consecutive locations as well, the final bandwidth will be close to the maximum possible.

When a warp is launched, all threads within it execute the same kernel. When a load instruction is executed, the hardware detects if the threads are trying to fetch consecutive memory

locations. If it is that way, the hardware does a coalesced memory access. Figure B-7 shows how different accessing patterns result in a coalesced and non-coalesced memory access.



(a) Coalesced **(b)** Non-coalesced

Figure B-7: Comparison between coalesced and non-coalesced memory access. Image credits: NVIDIA

B-2-5 Streams

In order to increase the performance, it is necessary to make use of all the parallelizing tools available. For example, when some data transfer between the host and device is happening, the GPU can be launching a kernel not related to that data. This overlap of operations can be achieve with the use of *streams*.

A CUDA stream executes the operations issued by the host in the same order they are launched. Operations in a stream are guaranteed to be executed in the issued order but, when there are several streams, operations can be overlapped and even run concurrently. Kernels and data transfers can be run in a stream as shown in Figure B-8.

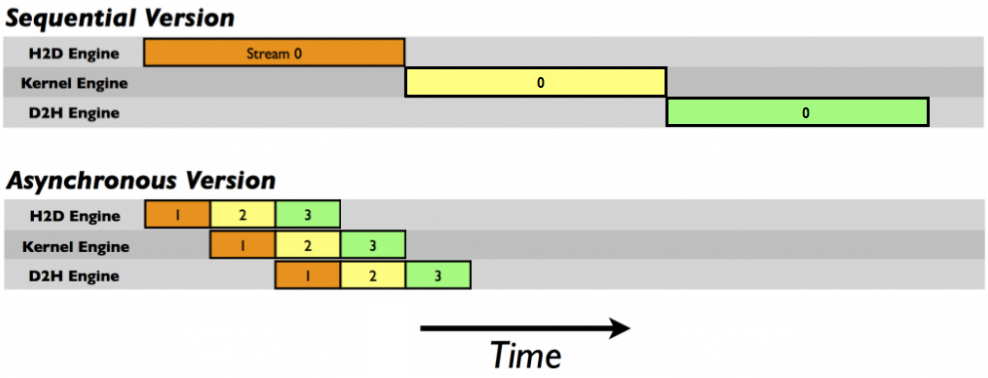


Figure B-8: Time comparison between sequential and concurrent computations using streams. Image credits: NVIDIA

B-2-6 Reductions

CPUs are very good at doing sequential operations, for example when computing the sum of an array. The sum is calculated loading a value of the array, adding it to an accumulator and then load the next value until the end of the array has been reached. However, this simple sum function is not efficient at all when it is carried out in a GPU; mainly because of the slow fetching speed of the memory and that only one thread is used.

In order to increase the speed of this computations on a GPU, the idea of the reduction kernels arises (it can also be applied to the calculation of maximum or minimum values). This approach divides the problem such that it can be solved in a parallel and recursive manner.

For solving the array sum, first it is necessary to assume that each thread starts with one value of the array. Secondly, create a partial sum adding all values of the treads within a block. Finally, add together all the partial sums. This last step can be seen as a recursion of the same procedure. Figure B-10 illustrates an example of this approach.

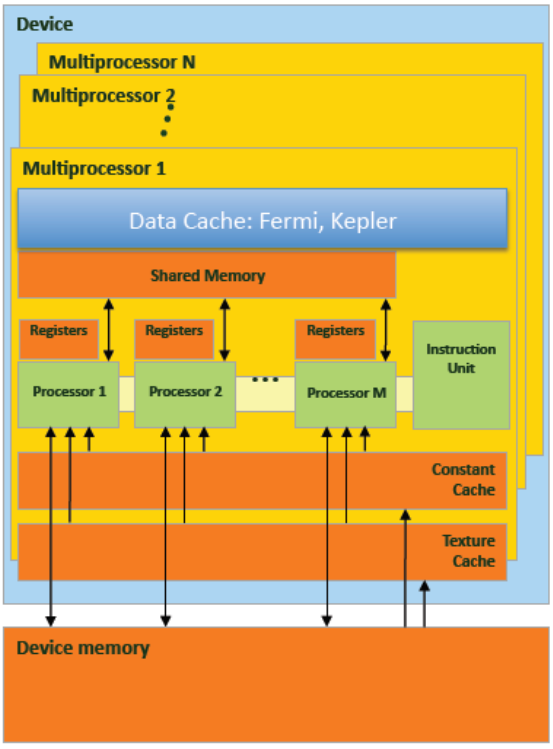


Figure B-9: Memory hierarchy in a GPU. Image credits: NVIDIA

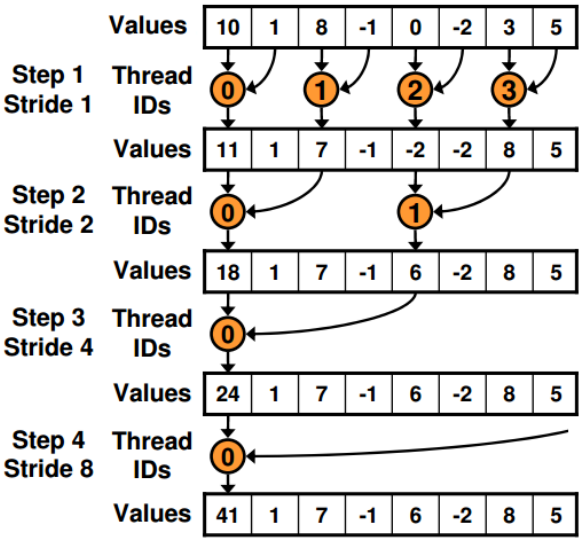


Figure B-10: Array sum reduction example. Image credits: NVIDIA

Appendix C

Appendix: Computational Performance Extended

In this appendix, the rest of the graphs comparing the FFT and TIP performance are shown. The same conclusions found in Chapter 9 are applicable to the following graphs.

C-1 FFT comparison

C-1-1 `complex64`

Non powers of two input size

See Figure C-1 for time comparison and Figure C-2 for FLOPS comparison.

C-1-2 `complex128`

Powers of two input size

See Figure C-3 for time comparison and Figure C-4 for FLOPS comparison.

Non powers of two input size

Figure C-5 for time comparison and Figure C-6 for FLOPS comparison.

C-2 TIP comparison

C-2-1 `float32`

9 input images

See Figure C-7.

C-2-2 `float64`

4 input images

See Figure C-8.

9 input images

See Figure C-9.

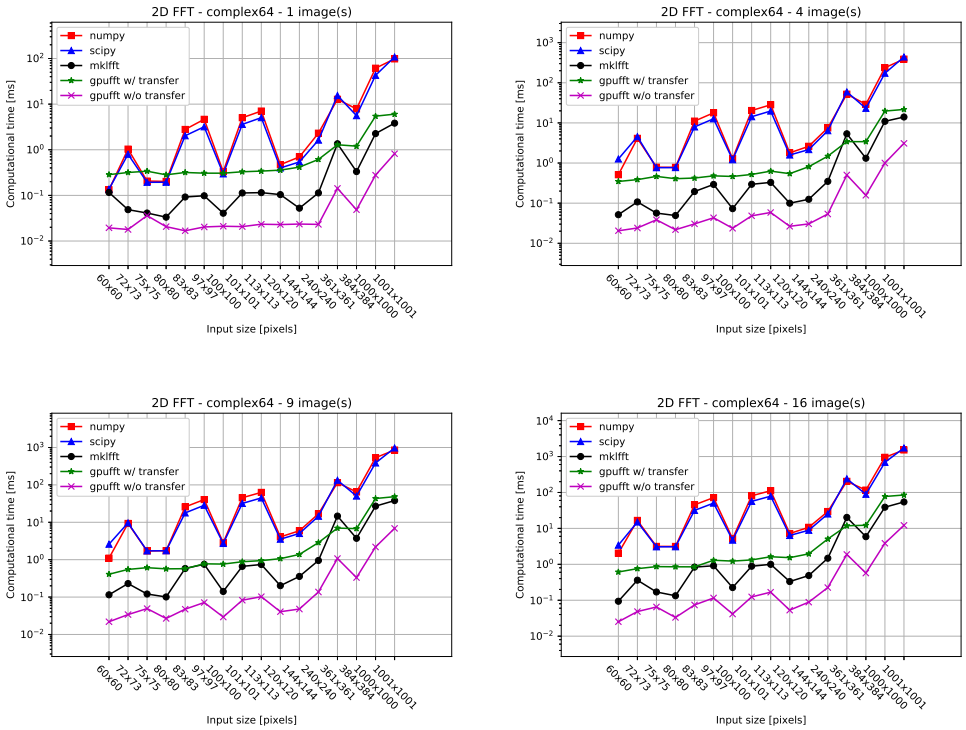


Figure C-1: 2D-FFT computational time comparison of the different implementations with a varying number of input images whose sizes are non powers of two.

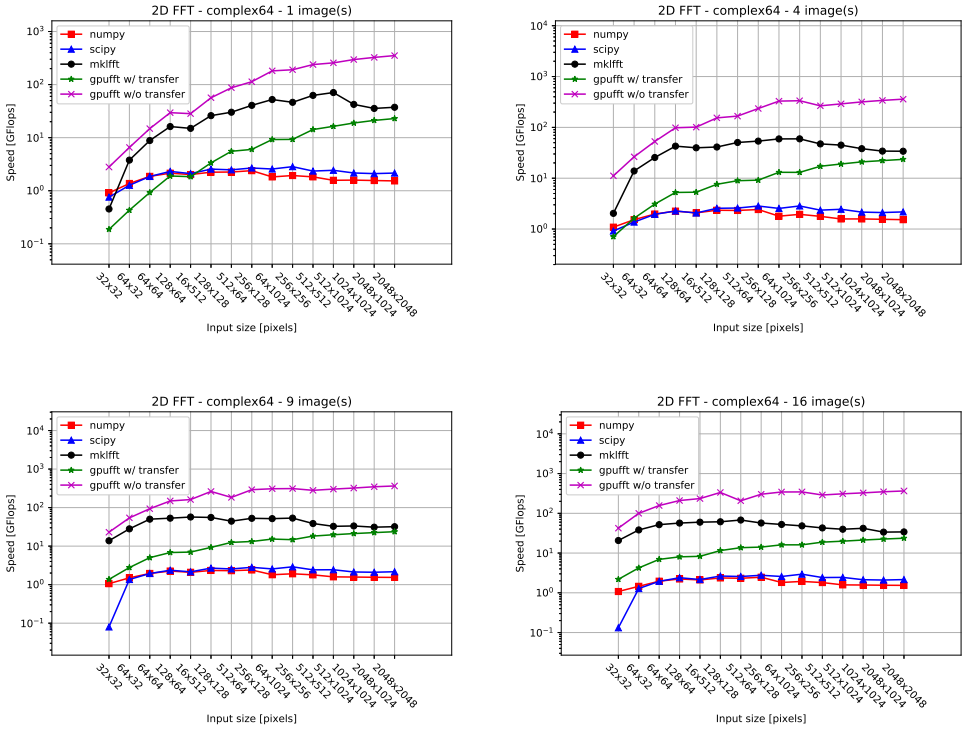


Figure C-2: 2D-FFT FLOPS comparison of the different implementations with a varying number of input images whose sizes are non powers of two.

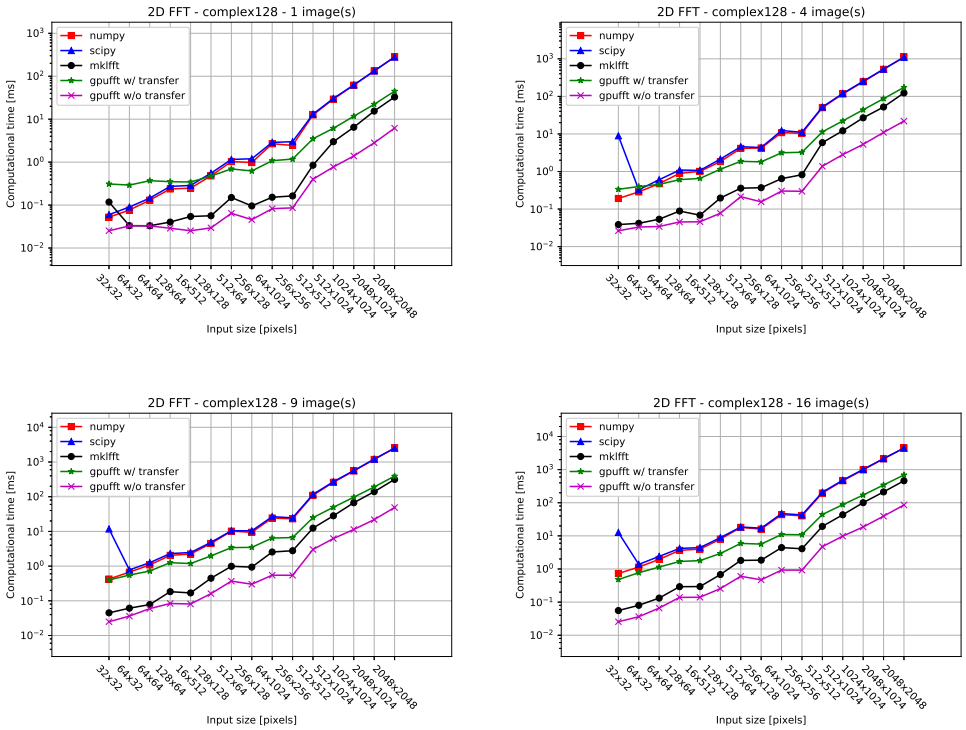


Figure C-3: 2D-FFT computational time comparison of the different implementations with a varying number of input images.

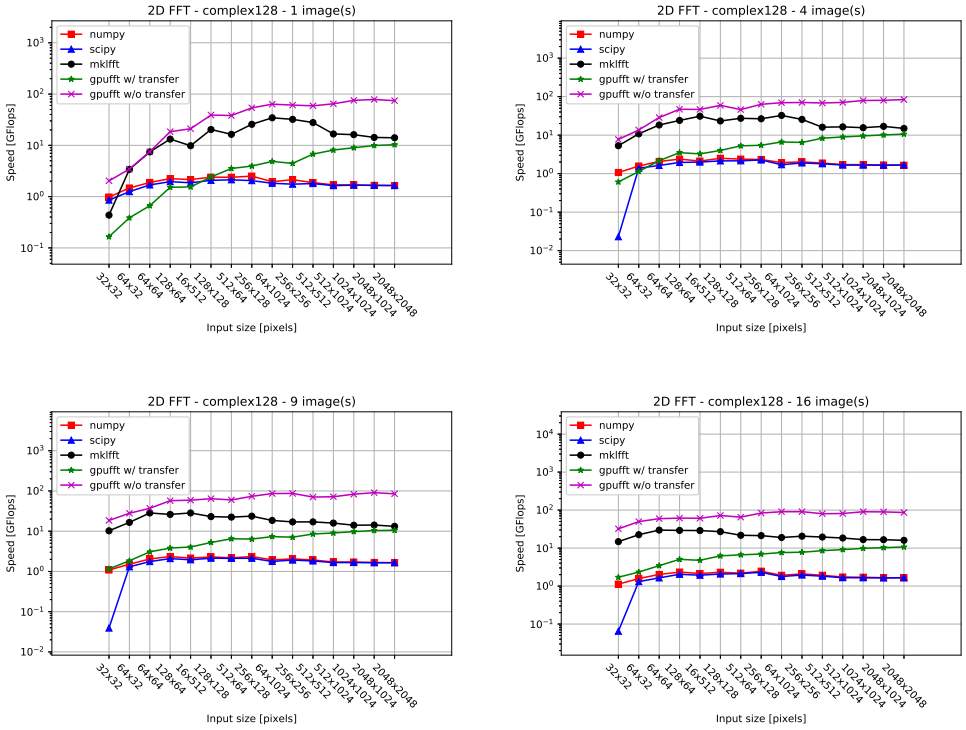


Figure C-4: 2D-FFT computational time comparison of the different implementations with a varying number of input images.

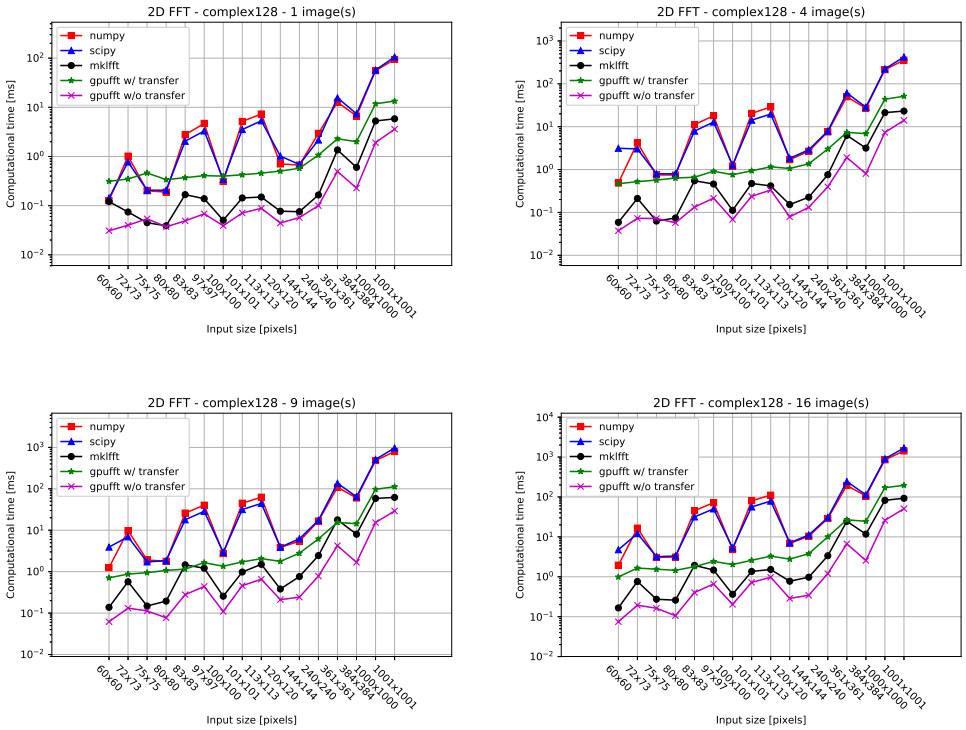


Figure C-5: 2D-FFT computational time comparison of the different implementations with a varying number of input images.

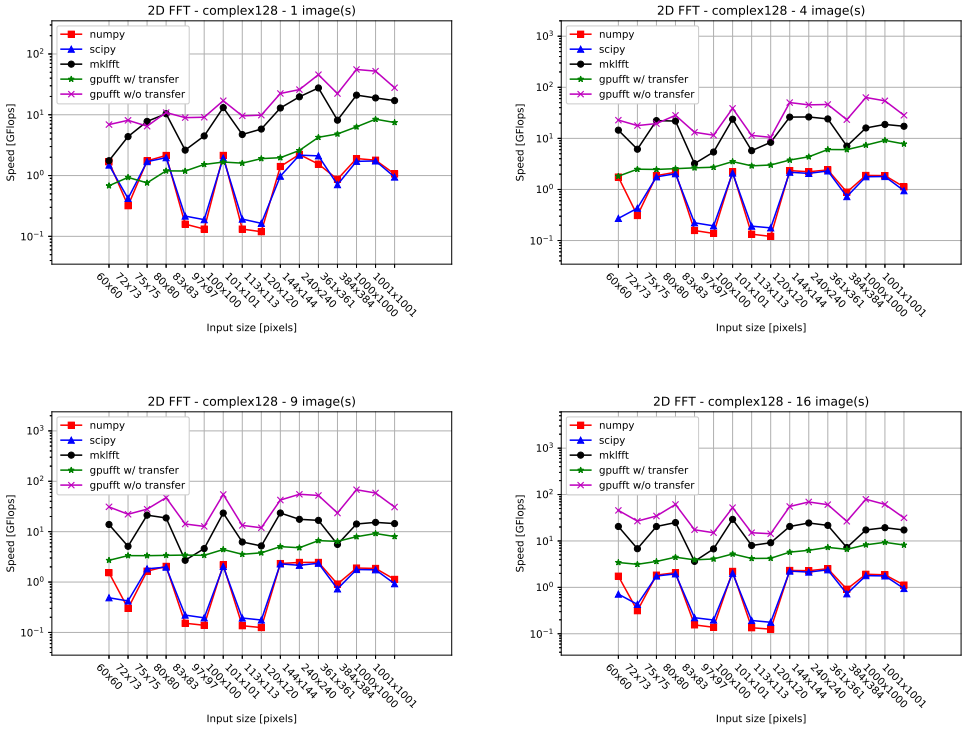


Figure C-6: 2D-FFT computational time comparison of the different implementations with a varying number of input images.

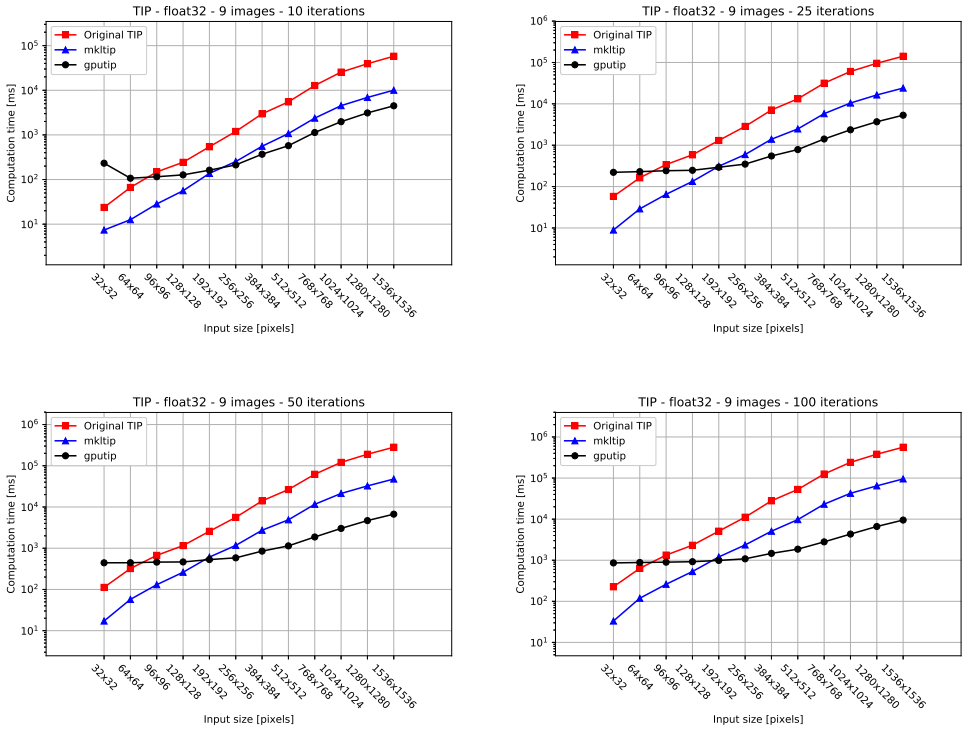


Figure C-7: Computational time comparison using 9 input images.

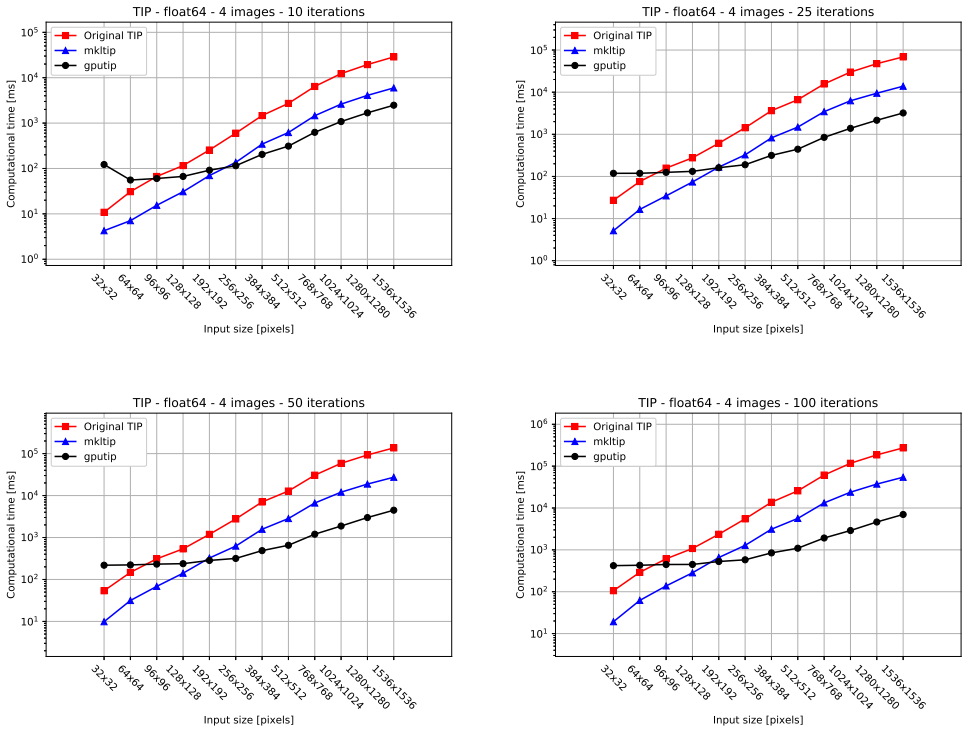


Figure C-8: Computational time comparison using 4 input images.

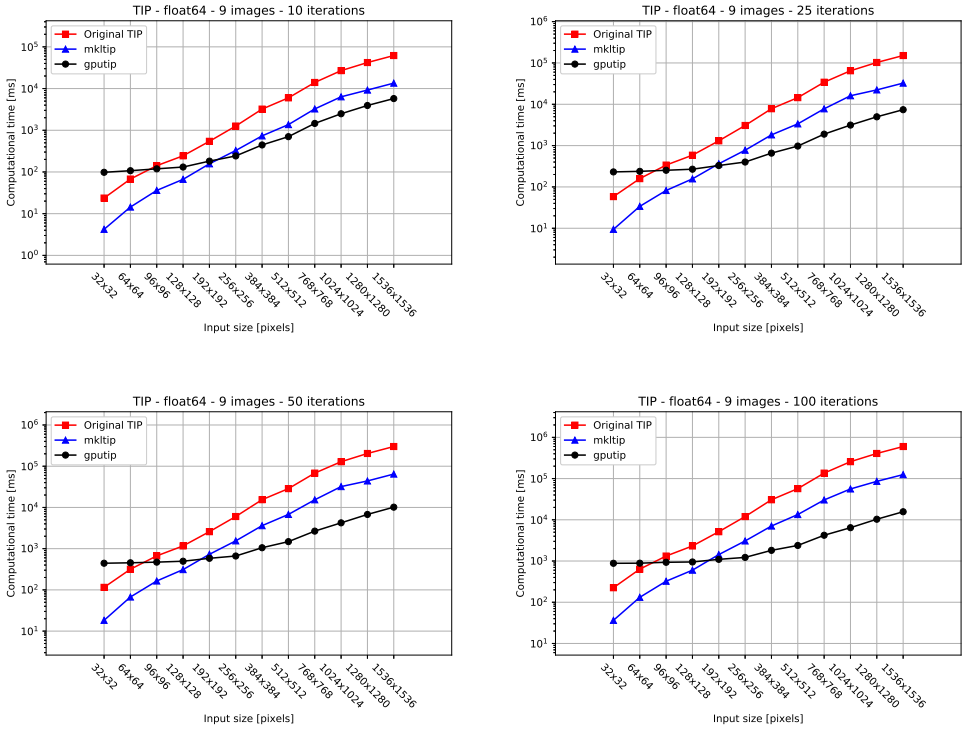


Figure C-9: Computational time comparison using 9 input images.

Appendix D

Appendix: Centering algorithms

In Chapter 10 it was mentioned that a centering algorithm was needed for aligning the different images of each sub-aperture so they have the same field of view.

Several algorithms were evaluated in different light conditions for this task but only one was implemented in the final GUI code. These algorithms are:

- `phaseCorrelation`: Makes use of the *Fourier shift theorem.*
- `ORB`: A computer vision algorithm used for finding and matching similar features in different images.
- `crossCorrelation`: Uses basic pixel cross correlation between images.

In the end, `crossCorrelation` was finally used because it showed a more robust performance for low light images with high noise levels.

D-1 Phase correlation

In Fourier theory, the *shift theorem* [51] indicates that an image o_n multiplied by a linear phase $\exp[\frac{2\pi i}{N}nm]$ for an integer m correspond to a circular shift of $O_k \rightarrow O_{k\_m}$. Assuming that $O_k = \mathcal{F}(\{o_n\})_k$, the previous explanation means that $O_{k\_m} = \mathcal{F}(\{o_n \cdot \exp[\frac{2\pi i}{N}nm]\})_k$ and that $\mathcal{F}(\{o_{n\_m}\})_k = O_k \cdot \exp[-\frac{2\pi i}{N}km]$.

Thanks to this property, it is possible to write an algorithm [52] to calculate the displacement (in pixels) of the different images (o_1 and o_2). Algorithm 4 describes the procedure in pseudocode.

Algorithm 4 Find the images displacement using the *phase correlation* approach

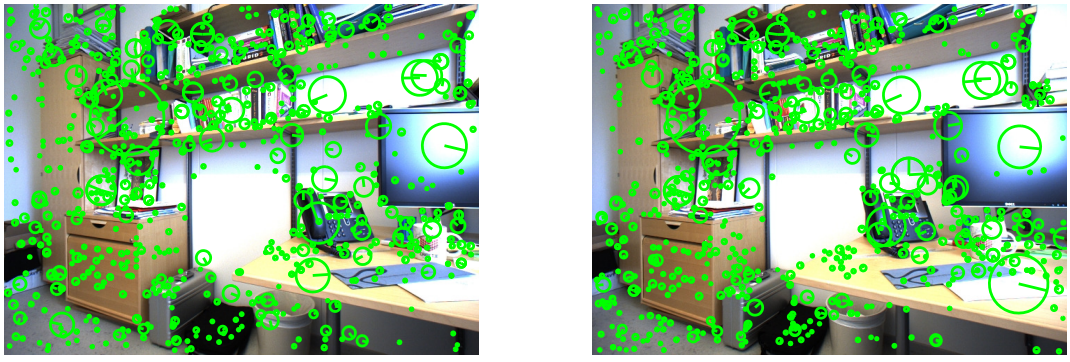
procedure PHASECORRELATION(o_1, o_2)
  Fourier transform:
  $O_1 \leftarrow \mathcal{F}(o_1)$
  $O_2 \leftarrow \mathcal{F}(o_2)$
  Cross-power spectrum: ▷ Recommended to apply a window function beforehand
  $C \leftarrow \dfrac{O_1 \circ O_2^*}{|O_1 \circ O_2^*|}$ ▷ The symbol $\circ$ is the Hadamard product
  $c \leftarrow \mathcal{F}^{-1}(C)$
  Find the peak of c:
  $(\Delta i, \Delta j) \leftarrow \underset{i,j}{\textbf{arg max}}(c)$
  return $(\Delta i, \Delta j)$
end procedure

One curiosity of this algorithm is that the cross-power spectrum C can be seen as the OTF (or PSF if we talk about c) which only produces a movement of the image (a delta function not centered).

D-2 ORB

In many computer vision problems, feature matching is at the base (i.e. object recognition and structure from movement). Nowadays, many methods rely on descriptors detection and matching (BRIEF[53], SIFT[54], SURF[55], ORB[56], among others). However, only a few of them are open source, free to use and robust against noise; in this case, the ORB descriptor is chosen.

This algorithm creates a set of keypoints for different images. Assuming that these images are only a moved version of the others (see Figure D-1), there are common objects in them, making it possible to match keypoints of the different sets (see Figure D-2).



(a) Original image. **(b)** Moved version of the original image.

Figure D-1: Feature detection using the ORB algorithm.

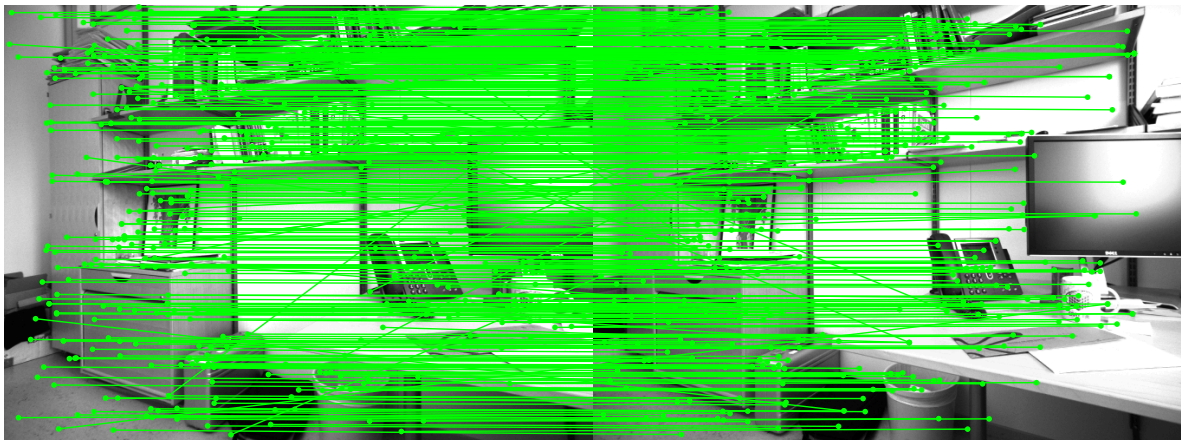


Figure D-2: Matched points of the office images.

Knowing which keypoint is the same in the different images and knowing their spatial position, it is possible to calculate the displacement. The implementation in pseudocode is shown in Algorithm 5.

Algorithm 5 Find the images displacement using the *ORB* descriptor

procedure ORB(o_1, o_2)
    Calculate the keypoints and descriptors:
    $(kp_1, des_1) \leftarrow$ **ORB.detect**(o_1)
    $(kp_2, des_2) \leftarrow$ **ORB.detect**(o_2)
    Match and sort the descriptors:
    $matches \leftarrow$ **ORB.match** (des_1, des_2)
    $matches \leftarrow$ **ORB.sort**($matches$, **quality**) ▷ Sort using a quality metric
    Calculate the distance:
    $(\Delta i, \Delta j) \leftarrow$ **ORB.distance**($kp_1[matches], kp_2[matches]$)
    return $(\Delta i, \Delta j)$
end procedure

D-3 Cross correlation

The last method analyzed, which is also the one implemented in the final code, is the correlation between image patches.

Taking a small patch of the original image p_1 and using the error metric

$$\text{SSD} = -\sum_i \sum_j \left(p_1[i,j] - p_2[i,j]\right)^2 \tag{D-1}$$

known as Sum of Squared Differences (SSD), it is possible to build a heat map calculating the SSD for each patch p_2 of the moved image with p_1. Once the heat map is available, it is possible to find the coordinates of them maximum, which in this case are the coordinates of the displacement between the images. This procedure is explained in Algorithm 6.

Algorithm 6 Find the images displacement using the *cross correlation* approach

procedure CROSSCORRELATION(o_1, o_2, w) ▷ w is the patch size
  Initialize:
  $p_1 \leftarrow$ **patch**$(o_1, w,$ **centered**$)$ ▷ A patch of size $w \times w$ is taken at the center
  Loop for all patches in o_2*:*
  for $i \leftarrow 1,$ **size**(o_1) **do**
    for $j \leftarrow 1,$ **size**(o_1) **do**
      $p_2 \leftarrow$ **patch**$(o_2, w, [i, j])$ ▷ A patch of size $w \times w$ is taken at $[i, j]$
      $heatMap[i, j] \leftarrow$ **SSD**(p_1, p_2)
    end for
  end for
  Find the peak of heatMap:
  $(\Delta i, \Delta j) \leftarrow \underset{i,j}{\textbf{arg max}}(heatMap)$
  return $(\Delta i, \Delta j)$
end procedure

Bibliography

[1] V. F. Sofieva, F. Dalaudier, and J. Vernin, "Using stellar scintillation for studies of turbulence in the Earth's atmosphere," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 371, no. 1982, 2012.

[2] M. Verhaegen, G. Vdovin, and O. Soloviev, *Control for High Resolution Imaging*. 2016.

[3] G. R. A. J. C. Dainty, "Iterative blind deconvolution method and its applications," *Optics Letters*, vol. 13, no. 7, pp. 547–549, 1988.

[4] D. a. Fish, a. M. Brinicombe, E. R. Pike, and J. G. Walker, "Blind deconvolution by means of the RichardsonâĂŞLucy algorithm," *Journal of the Optical Society of America A*, vol. 12, no. 1, p. 58, 1995.

[5] A. Levin and R. Fergus, "Deconvolution using natural image priors," *ACM Transactions on ...*, no. 6, pp. 0–2, 2007.

[6] F. Šroubek and J. Flusser, "Multichannel blind deconvolution of spatially misaligned images," *IEEE Transactions on Image Processing*, vol. 14, no. 7, pp. 874–883, 2005.

[7] F. Sroubek, G. Cristóbal, S. Member, and J. Flusser, "A Unified Approach to Superresolution and Multichannel Blind Deconvolution," *IEEE Transactions on Image Processing*, vol. 16, no. 9, pp. 2322–2332, 2007.

[8] M. Loktev, O. Soloviev, S. Savenko, and G. Vdovin, "Speckle imaging through turbulent atmosphere based on adaptable pupil segmentation," *Optics letters*, vol. 36, no. 14, pp. 2656–2658, 2011.

[9] Sung Cheol Park, Min Kyu Park, and Moon Gi Kang, "Super-resolution image reconstruction: a technical overview," *IEEE Signal Processing Magazine*, vol. 20, pp. 21–36, may 2003.

[10] D. Wilding, O. Soloviev, P. Pozzi, G. Vdovin, and M. Verhaegen, "Blind multiframe deconvolution by tangential iterative projections," *IEEE Transactions on Image Processing*, 2016.

[11] A. N. Kolmogorov, "The Local Structure of Turbulence in Incompressible Viscous Fluid for Very Large The local structure of turbulence in incompressible viscous fluid for very large Reynolds numberst," *Source: Proceedings: Mathematical and Physical Sciences*, vol. 434, no. 1890, pp. 9–13, 1991.

[12] D. L. Fried, "Optical Resolution Through a Randomly Inhomogeneous Medium for Very Long and Very Short Exposures," *Journal of the Optical Society of America*, vol. 56, no. 10, p. 1372, 1966.

[13] D. L. Fried, "Probability of getting a lucky short-exposure image through turbulence," *Optical Society of America*, vol. 68, no. May 1977, pp. 1651–1658, 1978.

[14] S. Chaudhuri, R. Velmurugan, and R. Rameshan, *Blind Image Deconvolution*. 2014.

[15] R. a. Gonsalves, "Phase retrieval and diversity in adaptive optics," *Optical Engineering*, vol. 21, no. 5, p. 829, 1982.

[16] R. A. Gonsalves, "Phase retrieval by differential intensity measurements," *Journal of the Optical Society of America A*, vol. 4, no. 1, pp. 166–170, 1987.

[17] R. G. Paxman, T. J. Schulz, and J. R. Fienup, "Joint estimation of object and aberrations by using phase diversity," *Journal of the Optical Society of America A*, vol. 9, no. 7, p. 1072, 1992.

[18] C. R. Vogel, T. Chan, and R. Plemmons, "Fast algorithms for phase diversity-based blind deconvolution," *in Proceedings of SPIE Conference on Adaptive Optical System Technologies*, vol. 3353, no. March, pp. 994–1005, 1998.

[19] M. Van Noort, L. R. Van Der Voort, and M. G. Löfdahl, "Solar image restoration by use of multi-frame blind de-convolution with multiple objects and phase diversity," *Solar Physics*, vol. 228, no. 1-2, pp. 191–215, 2005.

[20] A. A. Ramos and A. L. Ariste, "Image Reconstruction with Analytical Point Spread Functions," *arXiv*, vol. 6, pp. 1–9, 2010.

[21] L. Zhang and a. Cichocki, "Blind Deconvolution of Dynamical Systems: A State-Space Approach," 2001.

[22] C. Yu and M. Verhaegen, "Blind multivariable ARMA subspace identification," *Automatica*, vol. 66, pp. 3–14, 2016.

[23] Y. Yitzhaky, R. Milberg, S. Yohaev, and N. S. Kopeika, "Comparison of direct blind deconvolution methods for motion-blurred images.," *Applied optics*, vol. 38, no. 20, pp. 4325–4332, 1999.

[24] I. J. Myung, "Tutorial on maximum likelihood estimation," *Journal of Mathematical Psychology*, vol. 47, no. 1, pp. 90–100, 2003.

[25] A. M. Brinicombe, D. A. Fish, E. R. Pike, and J. G. Walker, "Blind deconvolution by means of the RichardsonâĂŞLucy algorithm," *JOSA A, Vol. 12, Issue 1, pp. 58-65*, vol. 12, no. 1, pp. 58–65, 1995.

[26] M. Jiang and G. Wang, "Development of blind image deconvolution and its applications.," *Journal of X-ray Science and Technology*, vol. 11, no. 1, pp. 13–9, 2003.

[27] A. Levin, Y. Weiss, F. Durand, and W. T. Freeman, "Understanding blind deconvolution algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 12, pp. 2354–2367, 2011.

[28] D. Krishnan and R. Fergus, "Fast image deconvolution using hyper-laplacian priors," *Advances in Neural Information Processing Systems (NIPS)*, pp. 1–9, 2009.

[29] A. Ahmed, B. Recht, and J. Romberg, "Blind Deconvolution Using Convex Programming," *Information Theory, IEEE Transactions on*, vol. 60, no. 3, pp. 1711–1732, 2014.

[30] B. Harmeling, Stefan; Sra, Suvrit; Hirsch, Michael; Scholkopf, "Multiframe Blind Deconvolution, Super-resolution, and Saturation Correctional via Incremental EM," in *IEEE 17th International Conference on Image Processing*, (Hong Kong), pp. 3313–3316, 2010.

[31] T. Schulz, B. Stribling, and J. Miller, "Multiframe blind deconvolution with real data: imagery of the Hubble Space Telescope.," *Optics express*, vol. 1, no. 11, pp. 355–362, 1997.

[32] T. J. Schulz, "Multiframe blind deconvolution of astronomical images," *Journal of the Optical Society of America A*, vol. 10, no. 5, p. 1064, 1993.

[33] V. P. Pauca, D. Chen, J. van der Gracht, R. J. Plemmons, S. Prasad, and T. C. Torgersen, "Pupil phase encoding for multi-aperture imaging," *Proceedings of SPIE*, vol. 7074, pp. 70740D–70740D–10, 2008.

[34] D. R. Gerwe, D. J. Lee, and J. D. Barchers, "Supersampling multiframe blind deconvolution resolution enhancement of adaptive optics compensated imagery of low earth orbit satellites," *Optical Engineering*, vol. 41, no. 9, p. 2238, 2002.

[35] P. H.S., S. H.L., and B. M. K.N., "Image Scaling Comparison Using Universal Image Quality Index," in *2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies*, pp. 859–863, IEEE, dec 2009.

[36] Hsieh Hou and H. Andrews, "Cubic splines for image interpolation and digital filtering," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, pp. 508–517, dec 1978.

[37] D. Jain, "Superresolution using Papoulis-Gerchberg Algorithm,"

[38] Xue-fen Wan and Yi Yang, "Super-resolution image reconstruction," in *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, pp. V8–351–V8–355, IEEE, oct 2010.

[39] G. van Rossum, "Scripting the Web with Python," *Scripting Languages: Automating the Web*, vol. 2, 1997.

[40] M. Soni, "A General Comparison Of Fft Algorithms,"

[41] D. Beazley, "Inside the Python GIL." 2009.

[42] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation," *Computing in Science & Engineering*, vol. 13, pp. 22–30, mar 2011.

[43] E. Jones, T. Oliphant, and P. Peterson, "SciPy: Open Source Scientific Tools for Python," 2001.

[44] M. Frigo, "A Fast Fourier Transform Compiler," 1999.

[45] I. corporation, "Intel Math Kernel Library," 2017.

[46] S. Kwan Lam, A. Pitrou, and S. Seibert, "Numba: A LLVM-based Python JIT Compiler,"

[47] S. Mittal and J. S. Vetter, "A Survey of CPU-GPU Heterogeneous Computing Techniques," *ACM Computing Surveys*, vol. 47, pp. 1–35, jul 2015.

[48] A. Klöckner, "PyCUDA: Even Simpler GPU Programming with Python," 2010.

[49] L. Givon, "scikit-cuda Documentation," 2017.

[50] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable Parallel PROGRAMMING," 2008.

[51] J. O. J. O. Smith, *Mathematics of the discrete Fourier transform (DFT) : with audio applicaitons*.

[52] E. De Castro and C. Morandi, "Registration of Translated and Rotated Images Using Finite Fourier Transforms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, pp. 700–703, sep 1987.

[53] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary Robust Independent Elementary Features,"

[54] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, 2004.

[55] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded Up Robust Features," pp. 404–417, Springer, Berlin, Heidelberg, 2006.

[56] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: an efficient alternative to SIFT or SURF,"

Glossary

List of Acronyms

AM	Alternating Minimization
AO	Adaptive Optics
BID	Blind Image Deconvolution
BLAS	Basic Linear Algebra Subprograms
CSI	Control for Scientific Imaging and Instrumentation
CUDA	Compute Unified Device Architecture
DCSC	Delft Center for Systems and Control
DFT	Discrete Fourier Transform
DM	Deformable Mirror
DRAM	Dynamic Random Access Memory
EM	Expectation Maximization
FFT	Fast Fourier Transform
FFTW	Fastest Fourier Transform in the West
FLOPS	Floating Point Operations per Second
GIL	Global Interpreter Lock
GPGPU	General-Purpose computing on GPU
GUI	Graphical User Interface
HR	High Resolution
HW	Hardware

IBD Iterative Blind Deconvolution

IRLS Iterative Re-weighted Least Squares

JIT Just In Time

LAPACK Linear Algebra Package

LR Low Resolution

LS Least Squares

LSI Linear Shift Invariant

MAP Maximum A Posteriori

MFBD Multi-Frame Blind Deconvolution

MIMO Multiple-Input Multiple-Output

MKL Math Kernel Library

MLE Maximum Likelihood Estimator

NUI Non-Uniform Interpolation

OTF Optical Transfer Function

PD Phase Diversity

PDF Probability Density Function

PG Papoulis-Gerchberg

POCS Projection Onto Convex Sets

PSF Point Spread Function

RL Richardson-Lucy

RMS Root Mean Square

ROI Region of Interest

SDP Semi-Definite Program

SIMD Single Instruction Multiple Data

SIMO Single-Input Multiple-Output

SM Streaming Multiprocessor

SNR Signal-to-Noise Ratio

SP Scalar Processors

SR Super-Resolution

SSD	Sum of Squared Differences
TIP	Tangential Iterative Projections
WFS	Wave-Front Sensor

List of Symbols

β	Noise regularization parameter
γ	Gain parameter for *stochastic* constraint.
$\phi(f)$	Phase Diversity
θ	Wavefront phase
$w(x)$	Noise
(f_x, f_y)	Spatial frequency in 2D
(x, y)	Spatial position in 2D
$*$	Spatial convolution operator
$\hat{h}(x)$	Estimated Point Spread Function
$\mathcal{F}[\cdot]$	Fourier transform
$A(f)$	Pupil function
$h(x)$	Point Spread Function
$i(x)$	Image measured
$J(x)$	Cost function
$o(x)$	Object
$p(x)$	Coherent system function
r_0	Fried parameter.
D	Downsampling operator.
U	Upsampling operator.
$_k$	Image index of a set of images.
^{n}	Current iteration number.