ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

ECOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

MASTER THESIS

Evaluation of Optical Aberrations using Phase Diversity

Author:
Jordan VOIRIN

Supervisors:
Dr. Laurent JOLISSAINT
Dr. Jean-Paul KNEIB

A thesis submitted in fulfillment of the requirements for the degree of Master in Applied Physics

in the

Astrophysics laboratory
Basic Sciences

March 5, 2018

Declaration of Authorship

I, Jordan VOIRIN, declare that this thesis titled, "Evaluation of Optical Aberrations using Phase Diversity" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"For the sake of persons of ... different types, scientific truth should be presented in different forms, and should be regarded as equally scientific, whether it appears in the robust form and the vivid coloring of a physical illustration, or in the tenuity and paleness of a symbolic expression. "

James Clerk Maxwell

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor...

Contents

List of Figures

List of Tables

Introduction

???

Chapter 1

Theoretical background

In this chapter, we will present the theory upon which this work is based. First, the light propagation formalism will be reminded through the scalar diffraction theory based on the Goodman (1988). Then we will describe in general an imaging system and its properties. And finally we will discuss the wavefront aberration theory.

1.1 Scalar Diffraction Theory

1.1.1 Scalar Field and Helmholtz equation

A monochromatic wave, at position P and time t, can be represented by a scalar field $u(P,t)$ written as :

$$u(P,t) = A(P)exp\left[-j2\pi\nu t - j\phi(P)\right], \tag{1.1}$$

where $A(P)$ and $\phi(P)$ are the amplitude and phase, respectively, of the wave at position P and ν is the wave frequency.

The spatial part of eqt. (1.1), also called phasor in the literature,

$$U(P) = A(P)e^{-j\phi(P)}, \tag{1.2}$$

must verify the Helmotz equation :

$$(\nabla^2 + k^2)U = 0, \tag{1.3}$$

where k is the wave number given by

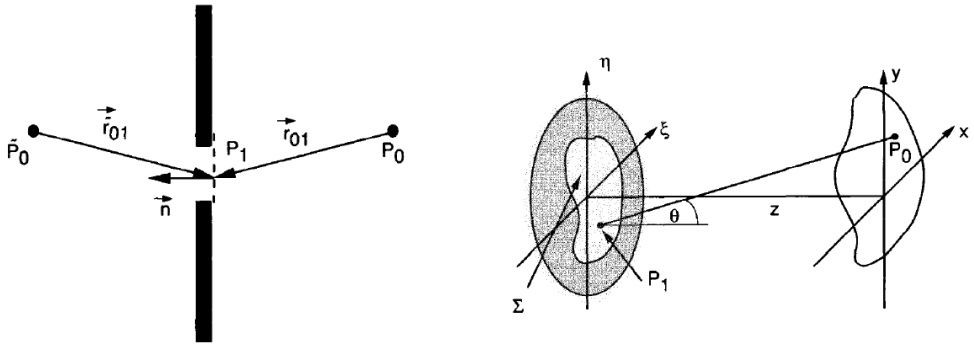
$$k = 2\pi n\frac{\nu}{c} = \frac{2\pi}{\lambda}, \tag{1.4}$$

and λ is the wavelength in the dielectric medium.

1.1.2 Rayleigh-Sommerfeld integral

Rayleigh and Sommerfeld developed a formalism using the Helmholtz equation and Green's Theorem to compute the induced diffraction by a plane screen. Let's suppose that we have a monochromatic source at $\widetilde{P_0}$ on the left of a plane screen with aperture Σ, the Rayleigh-Sommerfeld formula allows to compute the complex amplitude at P_0 on the right of the plane screen (see Figure 1.1a).

$$U(P_0) = \frac{1}{j\lambda}\iint_{\Sigma} U'(P_1)\frac{exp(jkr_{01})}{r_{01}}cos(\mathbf{n}, \mathbf{r_{01}})\mathbf{ds} \tag{1.5}$$



(A) Rayleigh-Sommerfeld formulation of diffraction by a plane screen, (Goodman, 1988, Chapter 3.5).

(B) Diffraction geometry, (Goodman, 1988, Chapter 4.1).

FIGURE 1.1: Diffraction Schemas

$U'(P_1)$ is the complex amplitude on the screen, $cos(\mathbf{n}, \mathbf{r_{01}})$ is the cosine of the angle between the aperture plane normal toward the source and the vector $\mathbf{r_{01}} = \mathbf{P_0P_1}$ given by

$$r_{01} = \sqrt{z^2 + (x-\xi)^2 + (y-\eta)^2}. \tag{1.6}$$

We can rewrite eqt. (1.5) using $cos(\mathbf{n}, \mathbf{r_{01}}) = cos(\theta) = \frac{z}{r_{01}}$ and the coordinate systems (ξ, η) and (x, y), see Figure 1.1b,

$$U(x,y) = \frac{z}{j\lambda} \iint_{-\infty}^{\infty} U(\xi,\eta) \frac{exp(jkr_{01})}{r_{01}^2} d\xi d\eta. \tag{1.7}$$

We can integrate from $-\infty$ to ∞, using $U(\xi,\eta) = P(\xi,\eta)U'(\xi,\eta)$ where $P(\xi,\eta)$ is the pupil function. The latter equals to one in the pupil and zero outside.

1.1.3 Fresnel approximation

To reduce eqt. (1.7), also known as the Huygens-Fresnel principle, one can approximate the distance r_{01} using the taylor expansion of the square root :

$$r_{01} = z\sqrt{1 + \frac{x-\xi}{z} + \frac{y-\eta}{z}} \approx z\left[1 + \frac{1}{2}\left(\frac{x-\xi}{z}\right)^2 + \frac{1}{2}\left(\frac{y-\eta}{z}\right)^2\right] \tag{1.8}$$

To obtain the Fresnel approximation, one has to replace r_{01} by eqt. (1.8) in eqt. (1.7). At the denominator, only the first term z is kept, since the introduced error is small, but in the exponential everything is kept. Then the final expression is given by,

$$U(x,y) = \frac{e^{jkz}}{j\lambda z} \iint_{-\infty}^{\infty} U(\xi,\eta) exp\left\{j\frac{k}{2z}\left[(x-\xi)^2 + (y-\eta)^2\right]\right\} d\xi d\eta. \tag{1.9}$$

In this form, the Fresnel approximation can be seen as a convolution between $U(\xi,\eta)$ and $h(x,y) = \frac{e^{jkz}}{j\lambda z} exp\left[\frac{jk}{2z}\left(x^2+y^2\right)\right]$.

Another form is found by developing $\left[(x-\xi)^2 + (y-\eta)^2\right]$,

$$U(x,y) = \frac{e^{jkz}}{j\lambda z} e^{j\frac{k}{2z}(x^2+y^2)} \iint_{-\infty}^{\infty} \left\{ U(\xi,\eta) e^{j\frac{k}{2z}(\xi^2+\eta^2)} \right\} e^{-j\frac{2\pi}{\lambda z}(x\xi+y\eta)} d\xi d\eta, \tag{1.10}$$

it is the Fourier transform of the complex field in the pupil multiplied by a quadratic phase exponential.

1.1.4 Fraunhofer approximation

In addition to the Fresnel approximation, we can introduce another approximation using the condition,

$$z >> \frac{k(\xi^2+\eta^2)_{max}}{2}. \tag{1.11}$$

If eqt. (1.11) is satisfied the Fresnel approximation simplifies, since the quadratic phase factor in (ξ,η) is approximately one on the entire pupil, as

$$U(x,y) = \frac{e^{jkz}}{j\lambda z} e^{j\frac{k}{2z}(x^2+y^2)} \iint_{-\infty}^{\infty} U(\xi,\eta) e^{-j\frac{2\pi}{\lambda z}(x\xi+y\eta)} d\xi d\eta. \tag{1.12}$$

For instance, at a wavelength of 637.5 nm and a pupil diameter of 3.6 mm the Fraunhofer approximation constrain z to be greater than 63 meters to be valid.

1.1.5 Converging lens introduction

The Fraunhofer conditions are severe as shown above, but one can reduce the distance z by observing at the focal plane of a converging lens. Indeed, using the paraxial approximation, i.e. small angles with respect with the optical axis, the lens transmission function is given by,

$$\begin{aligned} t_l(\xi,\eta) &= exp\left[jkn\Delta_0\right] exp\left[-jk\left(n-1\right)\frac{\xi^2+\eta^2}{2}\left(\frac{1}{R_1}-\frac{1}{R_2}\right)\right] \\ &= exp\left[-j\frac{k}{2f}(\xi^2+\eta^2)\right], \end{aligned} \tag{1.13}$$

where n is the refractive index of the lens material, R_1 and R_2 are the radii of curvature of the front and back surface of the lens, respectively and f is the focal length of the lens defined as,

$$\frac{1}{f} \equiv (n-1)\left(\frac{1}{R_1}-\frac{1}{R_2}.\right) \tag{1.14}$$

We can define $U_l(\xi,\eta) = U(\xi,\eta)t_l(\xi,\eta)$, which represents the complex amplitude passing through a lens. Finally, replacing $U(\xi,\eta)$ by $U_l(\xi,\eta)$ in Fresnel approximation and setting the observing distance to the focal length of the converging lens, we recover the Fraunhofer approximation,

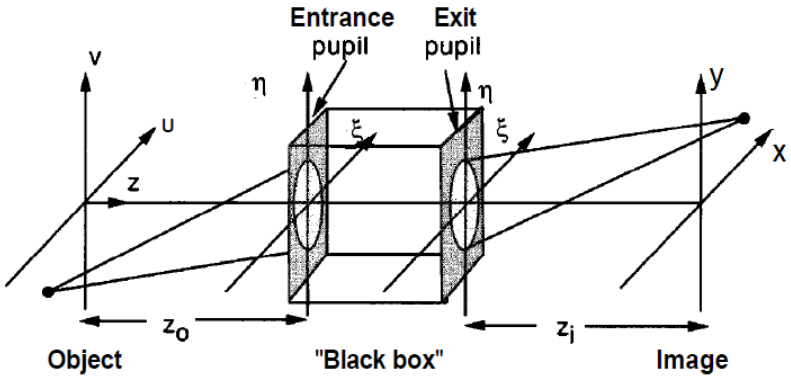


FIGURE 1.2: Schema of a imaging instrument, (Goodman, 1988, Chapter 6.1)

$$U(x,y) = \frac{e^{jkz}}{j\lambda z} e^{j\frac{k}{2z}(x^2+y^2)} \mathcal{F}\left\{ U(\xi,\eta) exp\left[j\frac{k}{2}(\xi^2+\eta^2)(\frac{1}{z}-\frac{1}{f}) \right] \right\}$$
$$\overset{z=f}{=} \frac{e^{jkz}}{j\lambda z} e^{j\frac{k}{2z}(x^2+y^2)} \mathcal{F}\left\{ U(\xi,\eta) \right\}. \tag{1.15}$$

1.2 Imaging system

An imaging system, such as a telescope, is used to acquire images of an object as perfectly as possible. An optical system, forming an instrument, is composed by lenses, mirrors, etc... and a detector (can be the human eye). A complex optical system can be reduced to a pupil, $P(\xi,\eta)$, and a focal length, f. The diffraction of the wave can be determined by the Fraunhofer approximation as long as the paraxial approximation is valid, see subsection 1.1.5. And the observed image of an incoherent object at the focal plane of the system is proportional to the square modulus of the complex amplitude $U(x,y)$,

1.2.1 Impulse Response (IR)

The impulse response or point spread function (PSF), $h(x,y;u,v)$, of an optical system is the field amplitude induced at coordinates (x,y) by a unit-amplitude point source at object coordinates (u,v). Using the linearity of the wave propagation, we can write the imaged amplitude as a superposition integral,

$$U(x,y) = \iint_{-\infty}^{\infty} h(x,y;u,v) U(u,v) du dv \tag{1.16}$$

1.2.2 Optical Transfer Function (OTF)

The optical transfer function, OTF, is defined as the Fourier transform of the impulse response, an example is shown in Figure 1.3. As we will use it later, for incoherent illumination and using the Fourier transform properties it can also be given by the

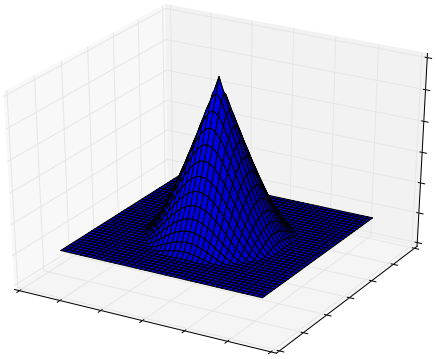


FIGURE 1.3: OTF of a perfect imaging system composed by a 3.6 mm pupil and a focal length of 80 mm at a wavelength of 637.5 nm.

autocorrelation of the pupil function as we will see in section 1.2.3 that the impulse response is given by the Fourier transform of the squared pupil function,

$$\widetilde{h}_{optical}(\xi,\eta) = \mathcal{F}\left\{h_{optical}(x,y)\right\} = (P \otimes P)(\xi,\eta), \tag{1.17}$$

where ξ and η are the conjugate variables of x and y with respect to the Fourier transform.

1.2.3 From Object to Image

A detector only senses the energy distribution produced by an electromagnetic wave. Therefore, an image is given by the square modulus of the complex amplitude at the focal plane,

$$i(x,y) = |U(x,y)|^2 = |\iint_{-\infty}^{\infty} h(x,y;u,v)U(u,v)dudv|^2. \tag{1.18}$$

This integral simplifies differently depending on the type of object we are observing. For a **coherent object**, Goodman (1988, Chapter 6.2) showed that the imaging is linear in complex amplitude, thus eqt. (1.18) becomes,

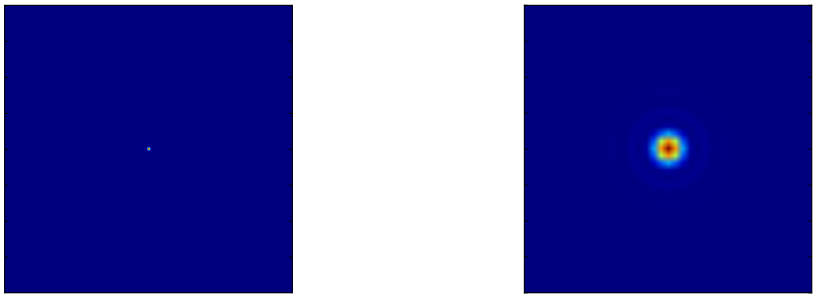
$$i(x,y) = |\iint_{-\infty}^{\infty} h(x-\widetilde{u},y-\widetilde{v})U(\widetilde{u},\widetilde{v})d\widetilde{u}d\widetilde{v}|^2, \tag{1.19}$$

where $(\widetilde{u} = Mu, \widetilde{v} = Mv)$ are the normalized object coordinates and M is the magnification of the imaging system. The image is given by the squared modulus of the convolution of the impulse response and the object complex amplitude.

For an **incoherent object**, Goodman (1988, Chapter 6.2) showed that the imaging is linear in intensity,

$$i(x,y) = \iint_{-\infty}^{\infty} |h(x-\widetilde{u},y-\widetilde{v})|^2 o(\widetilde{u},\widetilde{v})d\widetilde{u}d\widetilde{v} = (h_{optical} \otimes o)(x,y), \tag{1.20}$$

where $o(x,y) = |U(x,y)|^2$. We can recognize the convolution of an object with an intensity impulse response, $h_{optical}(x,y) = |h(x,y)|^2$.



(A) PSF (B) PSF zoom x10

FIGURE 1.4: PSF of a perfect imaging system composed by a 3.6 mm pupil and a focal length of 80 mm at a wavelength of 637.5 nm. The size, N, of the PSF is 400 and the pixel size is 5.3 μm.

In this study, we will study stars radiation which are object with an incoherent emission. The stars are point source objects given there distances. The image that an optical system gives of a point source is called the point spread function, PSF, or impulse response, IR, of the system, see Figure 1.4. A point source is characterized by an infinite distance to the instrument and therefore the wave is planar, which means that the phasor is reduced to $U(\xi,\eta) = P(\xi,\eta)$. The PSF or IR is given by,

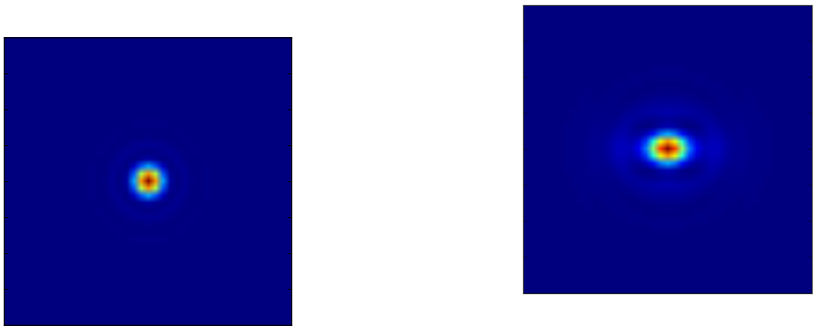
$$h_{optical}(x,y) = |\left[\mathcal{F}\left\{P(\xi,\eta)\right\}\right](x,y)|^2 \tag{1.21}$$

The domain where the impulse response is invariant under translation is called the **isoplanatic domain**.

In presence of aberrations, which will be discussed in section 1.3, the wavefront is deformed with respect to the perfect planar or spheric form. The pupil function at the exit of the imaging system is modified as following,

$$\mathcal{P}(\xi,\eta) = P(\xi,\eta)e^{-j\phi_{Ab}(\xi,\eta)}, \tag{1.22}$$

where $\phi_{Ab}(\xi,\eta)$ is the dephasing caused by the aberration present between the object and the image planes. Replacing the new pupil function in eqt. (1.21), we obtain the PSF of an imaging system having aberrations on the optical path.



(A) perfect PSF (zoomed x10)

(B) PSF with aberrations ($a_j = 50$nm for a_4, $a_6$and a_{11}, see subsec 1.3.2 for the definition of a_j, zoomed x10)

FIGURE 1.6: Comparison of perfect PSF and PSF with aberrations of an imaging system composed by a 3.6 mm pupil and a focal length of 80 mm at a wavelength of 637.5 nm. The size, N, of the PSF is 400 and the pixel size is 5.3 μm.

1.3 Aberrations

The aberrations present on the optical path between the object and the image planes decrease the quality of the resulting image. Indeed, they induce fluctuations of the amplitude and phase of the wave in the pupil plane. Since the amplitude fluctuations are negligible with respect to the phase fluctuations, we focus only on the latter. In figure 1.5, one can see the effect of the aberrations on an hypothetical spherical wavefront at the exit pupil of an imaging system. And in Figure 1.6, one can see the effect of aberrations on the PSF of an imaging system. The PSF is blurred due to the defocus and we can well recognize the astigmatism introduced as the shape of the PSF tends towards the ellipse. The Strehl ratio is often the used quantity to quantify the importance of the aberrations, it is given by the following expression,

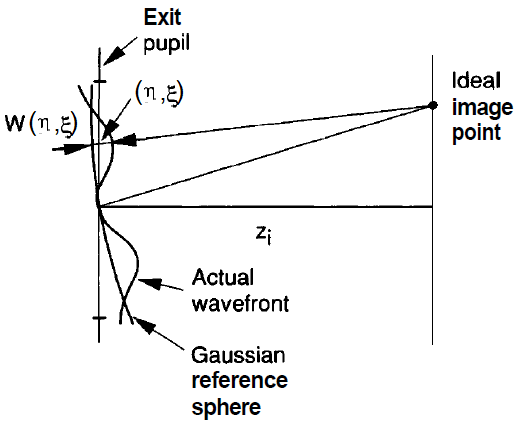


FIGURE 1.5: Gaussian reference sphere vs. aberrated Wavefront, (Goodman, 1988, Chapter 6.4). The dephasing is equal to the wave number multiplied by the aberration function, $\phi_{Ab}(\xi,\eta) = kW(\xi,\eta)$.

$$SR = \frac{\int \widetilde{h}_{optical}(\xi,\eta)\mathrm{d}\xi\mathrm{d}\eta}{\int \widetilde{h}_{perfect}(\xi,\eta)\mathrm{d}\xi\mathrm{d}\eta}, \tag{1.23}$$

where $\widetilde{h}_{perfect}(\xi,\eta)$ it the OTF of the perfect system with the same pupil as the optical system with the OTF $\widetilde{h}_{optical}(\xi,\eta)$.

1.3.1 Sources of aberration

The phase fluctuations introduced by aberrations are due to different kind of perturbation on the optical path. In ground based astronomy, the main source of aberrations is the **atmosphere**. Indeed, the atmosphere's temperatures fluctuations coupled with turbulent airflows are at the origin of the variations of the refractive index. Those variations induce then different optical path, in other words they generate perturbations on an electromagnetic wave passing through the atmosphere. The theory of the Atmospheric Turbulence is beyond the scope of this work, but we redirect the interested reader to Obukhov (1949), V.I. Tatarski (1961), and Kolmogorov (1968).

The other source of aberrations are the **defects of the instrument** themselves. The defaults can limit the resolution and decrease the quality of an diffraction-limited imaging system. The defects can have multiple origins as described by Blanc (2002). The first one is a **default during the fabrication process**, such as mirror polishing. This kind of defect is fixed and of high spatial frequency. Perturbation of the electromagnetic wave can also be due to **misalignment of the optical components of the instrument**. For instance, during the first stages of operation stages of the Hubble telescope, the mirrors were not aligned and the resulting images were blurry. These defects are of low spatial frequency and vary slowly with time. Finally, there are defects due to **mechanical stresses of the instrument**. The mirrors have to be held in place by different mechanical components. And under the influence of the gravity, a mirror deformation can arise resulting in an aberration introduction. This kind of defect also evolves slowly through time but has a large spatial frequency domain.

In this work, we focus on a method to correct the effect of the instrument defects. This method is particularly suited since it does not require a different path than the path to the scientific instrument, which means that we can correct the aberrations on the entire optical path.

1.3.2 Zernike polynomials

In order to study the aberrations present in an imaging system, F. Zernike (1934) introduced an orthonormal basis on which we can decompose the phase on a circular pupil, such has a telescope pupil. Those polynomials are the factor of a trigonometric function and a polynomial function (Noll, 1976).

$$Z_j(\mathbf{r}) = R_n^m(r)\Theta_n^m(\theta), \tag{1.24}$$

where $r = \frac{r'}{R_{pup}}$ is the normalized radius on the unit circle and θ is the azimuthal angle on the unit circle. j correspond to the Zernike index in the Noll order, for a specific j correspond only one couple (n, m). The values of n and m are always integral and satisfy $|m| \leq n$, $n - |m| =$ even.

The trigonometric function is given by,

$$\Theta_n^m(\theta) = \begin{cases} \sqrt{n+1} & if\, m = 0, \\ \sqrt{2(n+1)}cos(m\theta) & \text{if } m \neq 0 \text{ and } i \text{ even}, \\ \sqrt{2(n+1)}sin(m\theta) & \text{if } m \neq 0 \text{ and } i \text{ odd}, \end{cases} \tag{1.25}$$

and the radial function is given by,

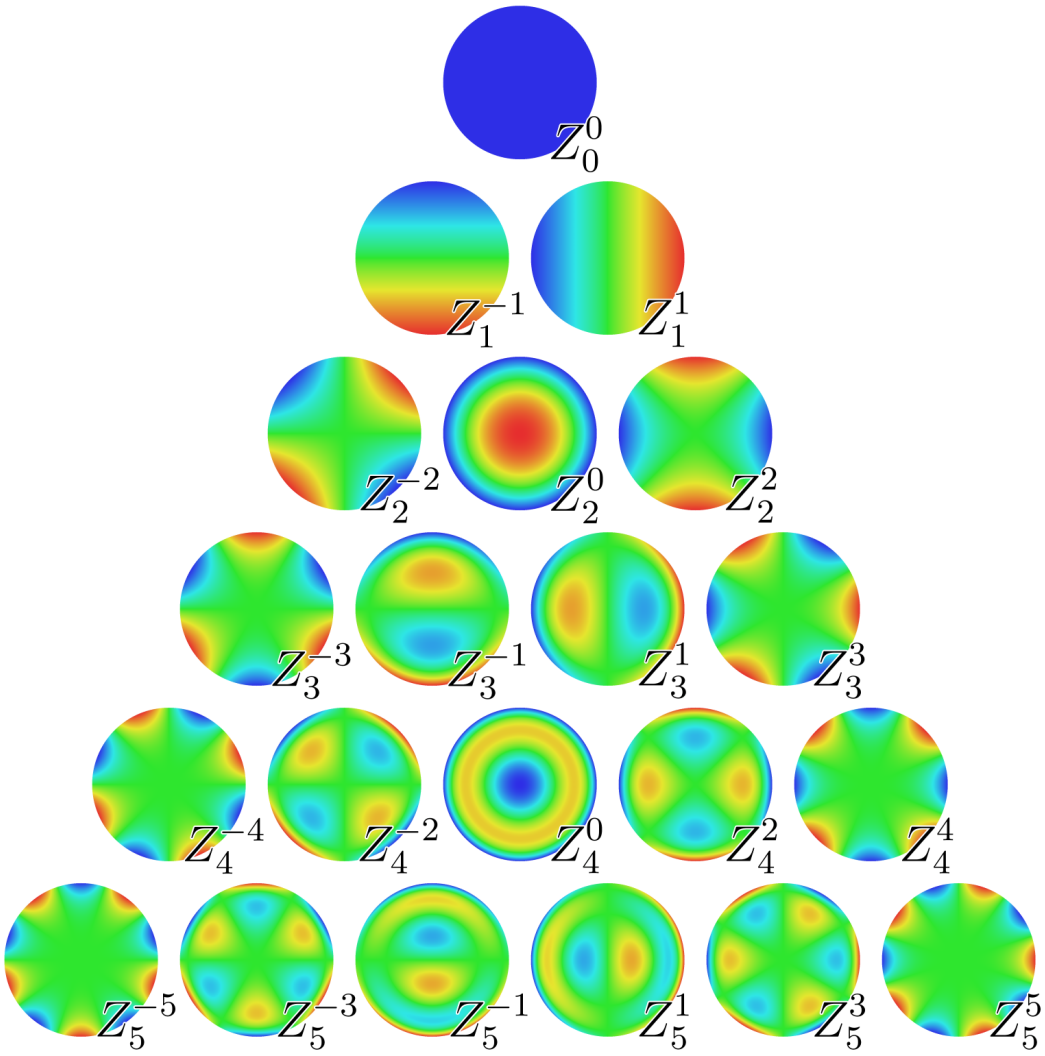


FIGURE 1.7: Representation of the 21^{st} Zernike polynomials (Wikipedia, 2018)

$$R_n^m(r) = \sum_{s=0}^{(n-m)/2} \frac{(-1)^s (n-s)!}{s![(n+m)/2-s]![(n-m)/2-s]!} r^{n-2s}. \tag{1.26}$$

The decomposition of the phase onto the Zernike polynomials is given by,

$$\phi(\mathbf{r}) = \sum_{i=1}^{+\infty} a_i Z_i(\mathbf{r}), \tag{1.27}$$

where the a_i's are the Zernike coefficient. And we characterize a wavefront by its root mean squared error without the piston component (a_1), which is given by,

$$\sigma_\phi = \sqrt{\frac{1}{S}\int_S \phi^2(\mathbf{r})d\mathbf{r}} = \sqrt{\sum_{i=2}^{+\infty} a_i^2}, \tag{1.28}$$

where S is the surface of the pupil.

1.4 Phase retrieval

The phase retrieval is a complicated process since the detectors are only sensitive to the intensity of a wave and not the wave itself, which renders impossible to have a direct relation between phase and image, thus we only have an indirect measurement of the wavefront. It is important to be able to estimate the aberrations of a system in order to correct them in real-time (Adaptive Optics systems) or in post-processing (Image restoration).

There are a couple of ways to characterize the form of a wavefront in order to determine the amplitude of the aberrations present in an imaging system. Some uses the optical geometric approximation, which says that locally the light rays are perpendicular to the wavefront. These achromatic methods measure the gradient of the wave surface, such as Shack-Hartmann (Hartmann, 1900; Shack and Platt, 1971; Fontanella, 1985), Curvature sensing (Roddier, 1988) and Pyramid sensing (Ragazzoni, 1996). An other kind of methods are called interferometric or focal plane methods, they use the interference patterns of the pupil to determine the form of the wavefront. The Phase Diversity is part of those kind of methods, it was discovered by Gonsalves (1982). The most popular method nowadays are the Shack-Hartmann, the curvature and the phase diversity. The latter will be explain in chapter 2 and we will explain the principle of a Shack-Hartmann sensor since we will use it later on.

1.4.1 Shack-Hartmann (Hartmann, 1900; Shack and Platt, 1971)

The Shack-Hartmann wavefront sensor measures the gradient of an aberrant phase. Figure 1.8 shows the principle. A micro-lenses array samples the wavefront passing through the pupil in a conjugated plane of the entrance pupil. Each micro-lens produces a dot on a CCD placed at the focii of the micro-lenses. The deformations of the wavefront induce a displacement of the dot with respect to a reference position obtained with a perfectly planar or spherical wavefront. By measuring this displacement Δx, it gives directly the local slope of the wavefront (Fontanella, 1985),

$$\frac{2\pi}{\lambda}\frac{\Delta x}{f} = \frac{1}{S}\int_S \frac{\delta\phi(x,y)}{\delta x}dxdy, \tag{1.29}$$

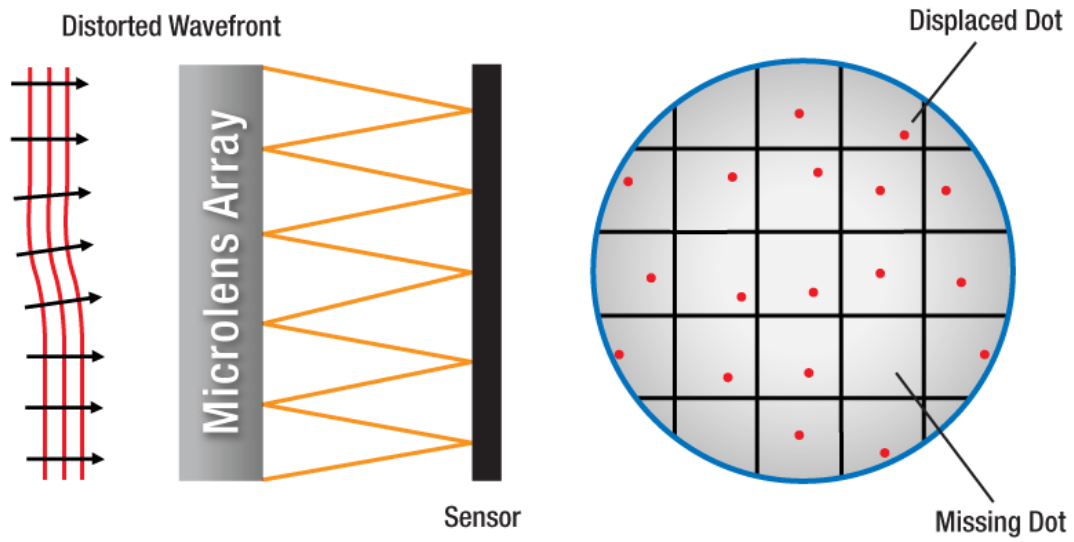


FIGURE 1.8: Shack-Hartmann principle (Thorlabs, 2018)

where S is the surface of a micro-lens and f is the focal length of the micro-lenses. The wavefront is reconstructed by integrating over all the local slope measurements. The advantage of this technique is that it do not require a lot of computation since it is a direct measurement. But it requires an important optical system to acquire the data. It is used in many fields, especially in adaptive optics system to correct for the atmospheric turbulence.

Chapter 2

Phase Diversity

The phase diversity was first implemented by R. A. Gonsalves in 1976 (Gonsalves, 1976; Gonsalves, 1982) to retrieve the phase of a wavefront coming from a point source. It uses two images, one at the focal plane and another one with a diversity introduced, such as defocus, in order to recover the phase of the wavefront. This chapter introduce the principle on which the phase diversity is based and then describes its implementation with two different algorithms. The first one is an algorithm developed at the ONERA by Mugnier, Blanc, and Idier (2006). The second algorithm presented is the one we developed in the frame of this study, which is based on an analytical approach.

2.1 Principle

Unlike Shack-Hartmann wavefront reconstruction, which is a pupil plane technique, the phase diversity uses data acquired at the focal plane. Using the non-linear relation between the phase of the wavefront and the image,

$$i(x,y) = (h_{optical} \otimes o)(x,y),\ with\ h_{optical}(x,y) = \left| \left[\mathcal{F}\left\{ A(\xi,\eta) e^{j\phi(\xi,\eta)} \right\} \right] (x,y) \right|^2, \tag{2.1}$$

one can determine the phase, i.e. the aberrations present in the imaging system, by solving an inverse problem. The major difficulty of this technique is that, as one can see in eqt. (2.1), there is not a unique solution to the problem at hand. This indetermination comes from the fact that the available detector can only sense the intensity of the wave, and not the wave itself, which is the modulus squared of the complex amplitude as exposed in section 1.2. Thus, $\phi(\xi,\eta)$ and $\phi'(\xi,\eta) = -\phi(-\xi,-\eta)$ give the same PSF. More specifically by decomposing the phase in its even and odd part and using the autocorrelation properties ($\Gamma_A = \Gamma_{A'}$ with $A'(t) = A * (-t)$), with only one image, one can not determine the sign of the phase even part. This leads to the introduction of a phase diversity to raise the indetermination. The idea is to add a known aberration $\delta\phi$ to the system and to use the two images to retrieve the phase of the wavefront.

The diversity between the two images is introduced for instance by defocusing one of the two images. In this work we will use this diversity, but having a more complex system, such has a deformable mirror, one could introduce any other even aberration such as an astigmatism, the only requirement is that the diversity introduced must have an even radial and azimuthal order. Figure 2.1 shows the principle of the phase diversity. Two images are acquired, one at the focal plane and another with a defocus. In this work, we introduce the defocus by sliding the detector along

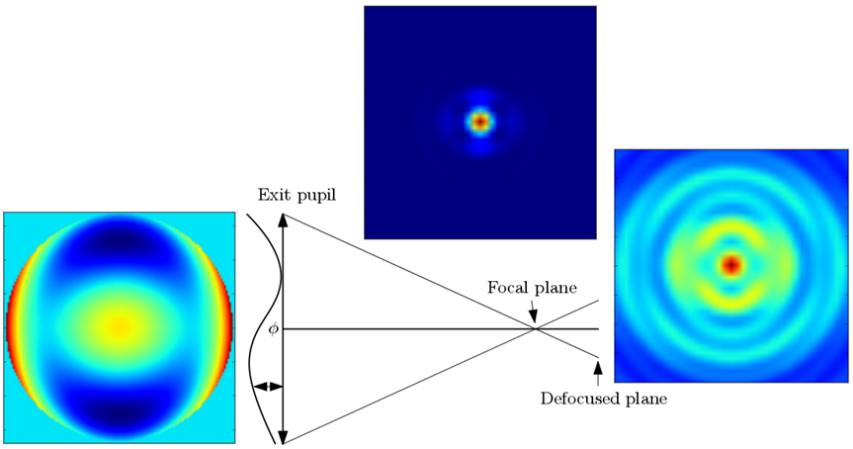


FIGURE 2.1: Schema of the phase diversity principle. The images from left to right are : the phase arriving on the exit pupil, the focused image and the defocused (2π) image.

the z-axis, others use a beamsplitter and another detector (Mugnier, Blanc, and Idier, 2006).

The phase diversity is a technique that is sensible to chromatism, since it is based on diffraction. The non-linear relation that links the image and the phase depends also on the object. This renders the inverse problem to solve more complicated, but it allows to retrieve the phase, the object or both, when the object is unknown (most of the time). Furthermore, it is very simple optically, do not require a complex optical components to acquire the data, one just uses the detector in place. And it depends directly on the images so there is no noncommon-path aberration between the adaptive optic system and the scientific detector.

In this work, the final application of the phase diversity algorithm will be to correct for static aberration in an optical system. We will have knowledge of the object, since it will be a point source that we introduce with a laser or choose in the sky (a star).

2.2 ONERA algorithm, Mugnier, Blanc, and Idier (2006)

In this section, we will present briefly the phase diversity algorithm developed at ONERA since we test and use it to retrieve the phase and the aberrations in the experiment conducted in laboratory.

2.2.1 Algorithm description

As explained in section 2.1, the phase diversity determines the phase of the wavefront, as well as the unknown object when needed, using two images of the same object with a phase diversity between each image. This gives the following equation system (Mugnier, Blanc, and Idier, 2006, p.11),

$$\mathbf{i}_f = \mathbf{h}_f * \mathbf{o} + \mathbf{n}_f \tag{2.2}$$

$$\mathbf{i}_d = \mathbf{h}_d * \mathbf{o} + \mathbf{n}_d, \tag{2.3}$$

where the bold letters means that it is the sampled quantities, the indexes f and d means respectively focused and defocused and **n** regroups the photon and detector noise present on the image. Using the data available, this algorithm approaches the problem at hand with a statistic point of view. They estimate jointly the aberrations and the object (Paxman, Schulz, and Fienup, 1992), which consist to compute the joint maximum *a posteriori* (JMAP) estimator (Mugnier, Blanc, and Idier, 2006, p.17),

$$\begin{aligned}(\hat{\mathbf{o}}, \hat{\boldsymbol{\phi}})_{MAP} &= \arg\max_{\mathbf{o},\boldsymbol{\phi}} p(\mathbf{i}_f, \mathbf{i}_d, \mathbf{o}, \boldsymbol{\phi}; \boldsymbol{\theta}) \\ &= \arg\max_{\mathbf{o},\boldsymbol{\phi}} p(\mathbf{i}_f|\mathbf{o}, \boldsymbol{\phi}; \boldsymbol{\theta_n}) p(\mathbf{i}_d|\mathbf{o}, \boldsymbol{\phi}; \boldsymbol{\theta_n}) p(\mathbf{o}; \boldsymbol{\theta_o}) p(\boldsymbol{\phi}; \boldsymbol{\theta_\phi}),\end{aligned} \tag{2.4}$$

where $p(\mathbf{i}_f, \mathbf{i}_d, \mathbf{o}, \boldsymbol{\phi}; \boldsymbol{\theta})$ is the joint probability density function of the two images $(\mathbf{i}_f, \mathbf{i}_d)$, the object $\mathbf{o}$ and the phase $\boldsymbol{\phi}$. It can also depend on a set of hyperparameters $\boldsymbol{\theta} = (\boldsymbol{\theta_n}, \boldsymbol{\theta_o}, \boldsymbol{\theta_\phi})$. $p(\mathbf{i}_k|\mathbf{o}, \boldsymbol{\phi}; \boldsymbol{\theta_n})$ is the likelihood of the image $\mathbf{i}_k$. $p(\mathbf{o}; \boldsymbol{\theta_o})$ and $p(\boldsymbol{\phi}; \boldsymbol{\theta_\phi})$ are the *a priori* probability density functions of $\mathbf{o}$ and $\boldsymbol{\phi}$.

They assume that the noise is white and stationary with a variance σ^2 on each image. They take Gaussian prior probability distributions for the object and for the phase which they decompose on the Zernike polynomial basis, $\boldsymbol{\phi}(\mathbf{a})$, with $\mathbf{a}$ the vector containing the Zernike coefficients from a_4 to $a_{j_{max}}$, see Mugnier, Blanc, and Idier (2006, p.18-19) for the detailed expressions. Finally, the phase and the object are retrieved by maximizing the joint probability density function $p(\mathbf{i}_f, \mathbf{i}_d, \mathbf{o}, \mathbf{a}; \boldsymbol{\theta})$ or taking the logarithm of the latter they retrieve them by minimizing the following criterion,

$$\begin{aligned}&L_{JMAP}(\mathbf{o}, \mathbf{a}, \boldsymbol{\theta}) \\ &= -\ln p(\mathbf{i}_f, \mathbf{i}_d, \mathbf{o}, \mathbf{a}; \boldsymbol{\theta}) \\ &= N^2 \ln \sigma^2 + \frac{1}{2}\ln\det(R_0) + \frac{1}{2}\ln\det(R_a) \\ &+ \frac{1}{2\sigma^2}(\mathbf{i}_f - H_f\mathbf{o})^t(\mathbf{i}_f - H_f\mathbf{o}) + \frac{1}{2\sigma^2}(\mathbf{i}_d - H_d\mathbf{o})^t(\mathbf{i}_d - H_d\mathbf{o}) \\ &+ \frac{1}{2}(\mathbf{o} - \mathbf{o}_m)^t R_o^{-1}(\mathbf{o} - \mathbf{o}_m) + \frac{1}{2}\mathbf{a}^t R_a^{-1}\mathbf{a} + A,\end{aligned} \tag{2.5}$$

where N^2 is the number of pixels in the image, $\mathbf{o}_m$ and R_o are the mean object and its covariance matrix, R_a is the covariance matrix of the aberrations, H_k is the matrix representing the discrete convolution by the sampled $\mathbf{h}$ and A is a constant.

In order to simplify and fasten the computation, they rewrite the criterion replacing $\mathbf{o}$ by its estimator $\hat{\mathbf{o}}(\mathbf{a}, \boldsymbol{\theta})$ obtained by cancelling the derivative of L_{JMAP} with respect to $\mathbf{o}$, and they move to the Fourier domain, see eqt. (24) of Mugnier, Blanc, and Idier (2006, p.21).

2.2.2 Implementation

2.3 Analytical algorithm

2.3.1 Algorithm description

This algorithm uses an analytical method to retrieve the phase of the wavefront **induced by a known point source object**. We assume the object known, because we

want to correct for the static aberrations present in the optical system to the scientific detector and thus we use a point source to illuminate the optical system, either a star or a laser.

As we have seen in section 1.2, the PSF of an optical system correspond to the image it gives of a point source,

$$PSF(x,y) = \frac{1}{S_p^2}\left|\left[\mathcal{F}\left\{P(\xi,\eta)A(\xi,\eta)e^{-j\phi(\xi,\eta)}\right\}\right](x,y)\right|^2, \tag{2.6}$$

where $P(\xi,\eta)$ is the exit pupil function, $A(\xi,\eta)$ is the amplitude of the wave through the exit pupil, $\phi(\xi,\eta)$ is the phase of the wavefront and S_p is the exit pupil surface. In the following we will omit the coordinates to simplify the notation. The unit of the PSF is directly the Strehl ratio. Under the assumption that we have weak aberrations, we can expand the exponential term,

$$exp(-j\phi) \approx 1 - j\phi - \frac{\phi^2}{2} + O(\phi^3), \tag{2.7}$$

replacing the exponential by its expansion in eqt.(2.6) leads to,

$$S_p^2 PSF \cong |\mathcal{F}\left\{PA(1 - j\phi - \frac{\phi^2}{2})\right\}|^2 \tag{2.8}$$

Developing eqt. (2.8), keeping only the terms up to the second order, assuming that the amplitude through the pupil $A(\xi,\eta)$ is constant and unitary since we have a point source object and using the well known complex relations,

$$\begin{aligned} a + a^* &= 2\Re\{a\} \\ a - a^* &= 2j\Im\{a\}, \end{aligned}$$

we obtain the following relation,

$$S_p^2 PSF \cong |\widetilde{P}|^2 + |\widetilde{P\phi}|^2 + 2\Im\{\widetilde{P^*}\widetilde{P\phi}\} - 2\Re\{\widetilde{P^*}\widetilde{P\phi^2}\} \tag{2.9}$$

Defining ΔPSF as the difference between eqt. (2.9) for an arbitrary optical system and its perfect equivalent, we obtain the following expression,

$$\Delta PSF = S_p^2 PSF - S_p^2 PSF_{perfect} = |\widetilde{P\phi}|^2 + 2\Im\{\widetilde{P^*}\widetilde{P\phi}\} - 2\Re\{\widetilde{P^*}\widetilde{P\phi^2}\}, \tag{2.10}$$

where $S_p^2 PSF_{perfect}$ is equal for a equivalent perfect system with the same pupil to $|\widetilde{P}|^2$. One can decompose ϕ into its even and odd phase, ψ and γ respectively,

$$\phi = \psi + \gamma \tag{2.11}$$

Developing eqt. (2.10) after replacing ϕ by its decomposition and using the properties of the Fourier transform of real and purely even or odd functions, we get the following expression,

$$\Delta PSF = |\widetilde{P\psi}|^2 + |\widetilde{P\gamma}|^2 + 2\Im\{\widetilde{P^*}\widetilde{P\gamma}\} - \Re\{\widetilde{P^*}\widetilde{P\psi^2}\} - \Re\{\widetilde{P^*}\widetilde{P\gamma^2}\} \tag{2.12}$$

We can decompose ΔPSF into its even and odd components,

$$\Delta PSF_{even} = |\widetilde{P\psi}|^2 + |\widetilde{P\gamma}|^2 - \Re\{\widetilde{P^*}\widetilde{P\psi^2}\} - \Re\{\widetilde{P^*}\widetilde{P\gamma^2}\}, \tag{2.13}$$

$$\Delta PSF_{odd} = 2\Im\{\widetilde{P^*}\widetilde{P\gamma}\}, \tag{2.14}$$

This equation system shows that we can retrieve the odd part of the phase easily with eqt. (2.14). But eqt. (2.13) clearly reveals the indetermination of the phase retrieval with only one image, as the sign of the even part of ϕ can not be determine. In order to raise this indetermination, as exposed in section 2.1, we need to introduce a phase diversity $\delta\phi$. We can modify the pupil function P in order to take into account this introduced diversity,

$$P_\delta \equiv Pe^{-j\delta\phi} = P(cos(\delta\phi) - jsin(\delta\phi)) = P(C - iS) \tag{2.15}$$

The expression of $\Delta PSF_{\delta\phi}$, which is the ΔPSF at the defocus plane, is found by replacing P by P_δ in eqt. (2.12), we give directly the expressions of the even and odd components by taking into account that the phase is only define on the pupil ($P\phi = \phi$) to simplify the reading,

$$\begin{aligned}\Delta PSF_{\delta\phi,even} &= |\widetilde{C\psi}|^2 + |\widetilde{C\gamma}|^2 + |\widetilde{S\psi}|^2 + |\widetilde{S\gamma}|^2 - 2\widetilde{PC}^*\widetilde{S\psi} + 2\widetilde{PS}^*\widetilde{C\psi} \\ &\quad - \widetilde{PC}^*\widetilde{C\psi^2} - \widetilde{PC}^*\widetilde{C\gamma^2} - \widetilde{PS}^*\widetilde{S\psi^2} - \widetilde{PS}^*\widetilde{S\gamma^2} \qquad (2.16)\\ \Delta PSF_{\delta\phi,odd} &= 2\widetilde{C\psi}^*\Im\{\widetilde{S\gamma}\} + 2\Im\{\widetilde{C\gamma}^*\}\widetilde{S\psi} + 2\widetilde{PC}^*\Im\{\widetilde{C\gamma}\} + 2\widetilde{PS}^*\Im\{\widetilde{S\gamma}\} \\ &\quad + 2j\widetilde{PC}^*\widetilde{S\psi\gamma} - 2j\widetilde{PS}^*\widetilde{C\psi\gamma}. \qquad (2.17)\end{aligned}$$

Eqt. (2.14), eqt. (2.16) and eqt. (2.17) allow to retrieve the complete phase of the optical system under the assumption of weak aberrations. The retrieval numerical method uses the decomposition of the even and odd part of the phase on the Zernike polynomials,

$$\psi = \sum_{js\ even} a_j Z_j \tag{2.18}$$

$$\gamma = \sum_{js\ odd} a_j Z_j. \tag{2.19}$$

This allows to have a linear system of equations with respect to the Zernike coefficient a_j using eqts. (2.14) and (2.17), but a problem arises as we tried to retrieve the phase of a purely even phase. The equations gave us an odd phase part equals to zero but also an even phase part equal to zero. This comes from the fact that in eqt. (2.17), each term is multiplied by the odd phase component. And we could not use eqts. (2.13) or (2.16), due to the squared modulus of the even and odd phase component, which rendered our system of of equation non-linear.

One way to get around this issue is to add another diversity, which is equal in amplitude to the first one but its inverse. So we have two diversity given by,

$$\delta\phi_+ = \delta\phi \text{ and } \delta\phi_- = -\delta\phi \tag{2.20}$$

Using the new diversity, eqt. (2.14) allows to determine the odd part of the phase as before, but now for the even part of the phase we compute the difference between $\Delta PSF_{\delta\phi_+,even}$ and $\Delta PSF_{\delta\phi_-,even}$. This gives us the following system of equation,

$$\Delta PSF_{odd} = 2\Im\{\widetilde{P^*}\widetilde{P\gamma}\} \tag{2.21}$$

$$\Delta PSF_{\delta\phi_+,even} - \Delta PSF_{\delta\phi_-,even} = -4\widetilde{PC}^*\widetilde{S\psi} + 4\widetilde{PS}^*\widetilde{C\psi}, \tag{2.22}$$

which we can use to determine the complete phase of the wavefront. We can rewrite it using the decomposition of ψ and γ on the Zernike basis,

$$\Delta PSF_{odd} = \sum_{js\ odd} a_j 2\Im\{\widetilde{P^*}\widetilde{PZ_j}\} \tag{2.23}$$

$$\Delta PSF_{\delta\phi_+,even} - \Delta PSF_{\delta\phi_-,even} = \sum_{js\ even} a_j\{-4\widetilde{PC}^*\widetilde{SZ_j} + 4\widetilde{PS}^*\widetilde{CZ_j}\}. \tag{2.24}$$

To solve these two equations and find the a_j's, we use a linear regression method. We can rewrite the equations under a vectorial form to clarify the equations by flattening the images,

$$\overrightarrow{\Delta PSF}_{odd} = \underline{Z}_f \vec{a}_{odd} \tag{2.25}$$

$$\overrightarrow{\Delta PSF}_{\delta\phi_+,even} - \overrightarrow{\Delta PSF}_{\delta\phi_-,even} = \underline{Z}_d \vec{a}_{even}. \tag{2.26}$$

where $\underline{Z}_f$ is the N^2xk_{odd} matrix regrouping all the terms of $2\Im\{\widetilde{P^*}\widetilde{PZ_j}\}$, k_{odd} is the number of odd Zernike polynomials between j_{min} and j_{max}, and $\underline{Z}_d$ is the N^2xk_{even} matrix regrouping all the terms of $\{-4\widetilde{PC}^*\widetilde{SZ_j} + 4\widetilde{PS}^*\widetilde{CZ_j}\}$, k_{even} is the number of even Zernike polynomials between j_{min} and j_{max}.

2.3.2 Implementation

The resolution of the equations as said above is done with a linear regression. The code is structured as following :

The Class phaseDiversity3PSFs is the main class of the program, see Appendix B.1.1. It takes as inputs : the 3 PSFs needed to retrieve the phase as exposed above (inFoc, outFocpos and outFocneg), the displacement of the detector with respect to its focused position (Δz),

$$\Delta z = \frac{4\Delta\phi}{\pi}\lambda\left(\frac{D}{F}\right)^2, \tag{2.27}$$

where $\Delta\phi$ is the Peak To Valley (P2V) dephasing that we want to introduce in the defocused image in the following it will be equal to 2π, D is the pupil diameter and F the focal length of the optical system. It takes also as inputs the wavelength of the incoming light, the pixel size of the detector, the focal length of the optical system, the radius of the pupil and the boundary on $jmin < j < jmax$ the Zernike index.

The instantiation of all the elements of eqts. (2.25) and (2.26) is done by calling the class method `initiateMatrix1()` and `initiateMatrix2()`. These two methods compute the left and right members of the equations by calling methods present in the script `fs.py`.

The script fs is gathering functions needed to retrieve the phase of the wavefront, see Appendix B.1.2. ΔPSF_{odd} and $\Delta PSF_{\delta\phi_+,even} - \Delta PSF_{\delta\phi_-,even}$ are computed by the

two methods `y1(params)` and `y2even(params)`. The matrix elements of $\underline{Z}_f$ are computed by the method `f1j(params)` and the elements of $\underline{Z}_d$ are computed by `f2jeven(params)`. Those two functions needs the Zernike polynomial basis which is coded in `zernike.py`, see Appendix B.1.4.

The linear regression is done after having initiated all the elements of the equations. We use the `numpy.linalg` class, especially its `lstq` function[1]. This functions takes a vector $\vec{b}$ corresponding to the flattened $DeltaPSF_{odd}$ or $\Delta PSF_{\delta\phi_+,even} - \Delta PSF_{\delta\phi_-,even}$ and a matrix $\underline{a}$ corresponding $\underline{Z}_f$ or $\underline{Z}_d$. Then it determines the vector $\vec{x}$ corresponding to the vector $\vec{a}$ that minimizes the euclidean 2-Norm $\|b - ax\|$ by computing the Moore-Penrose pseudo inverse of $\underline{a}$.

[1]Documentation found at https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.lstsq.html#numpy.linalg.lstsq

Chapter 3

Phase Diversity Experiment

Here, we will describe the experiment put in place in the optical laboratory at the HEIG-VD to reconstruct wavefronts with unknown static aberrations introduced using phase screens. At first, we study the behaviour of the phase diversity algorithm put in place by Mugnier, Blanc, and Idier (2006) at ONERA with respect to number of averaging images, in other words noise level, and number of Zernike coefficients retrieved. Then we test the algorithm using a known aberration introduced by a parallel plane plate in the beam comparing the result to Zemax simulation. And finally, we introduce the phase screen to have random aberrations in the pupil and try to compare the phase diversity results with the Shack Hartman wavefront sensor results.

3.1 Experimental Setup

The design of the experiment was already done by Bouxin (2017). The system is built according to her plans and specifications. Figure 3.1 shows the schema of the experimental setup.

The experiment is mounted on a pressurized legs optical table. The assembly contains six main components : a light source, an entrance pupil, an imaging system, a converging lens to focus the beam on the camera, a camera and a wavefront sensor.

3.1.1 Light source

The final application of the phase diversity will be to characterize the optical aberrations induced by the imperfect optical path to a scientific detector of a telescope. For this reason, the light source has to simulate a distant star aberration-free wavefront. A distant star wavefront is considered planar since the object distance, z, is far greater than the telescope size, r, see Fig. 3.2. The source of our experiment must then be characterized by a planar wavefront.

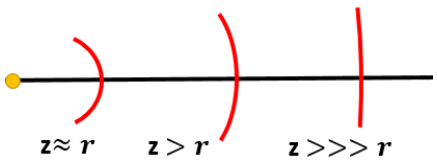


FIGURE 3.2: Wavefront curvature for different point source's distances, z. r represents the characteristic size of the arc of interest.

In order to obtain such a planar wavefront at the entrance pupil, the light source consist of a "pigtailed laser diode", a f=11mm converging lens, a pinhole and a f=200 mm converging lens, see Table 3.1. The pigtailed laser diode emits a Gaussian beam centred at 637.5 nm slightly diverging. The converging lens concentrates the beam at the center of the 10μm pinhole to filter the noise. The second converging lens collimates the beam, obtaining a collimated beam with a planar wavefront, see Fig. 3.3a and 3.3b.

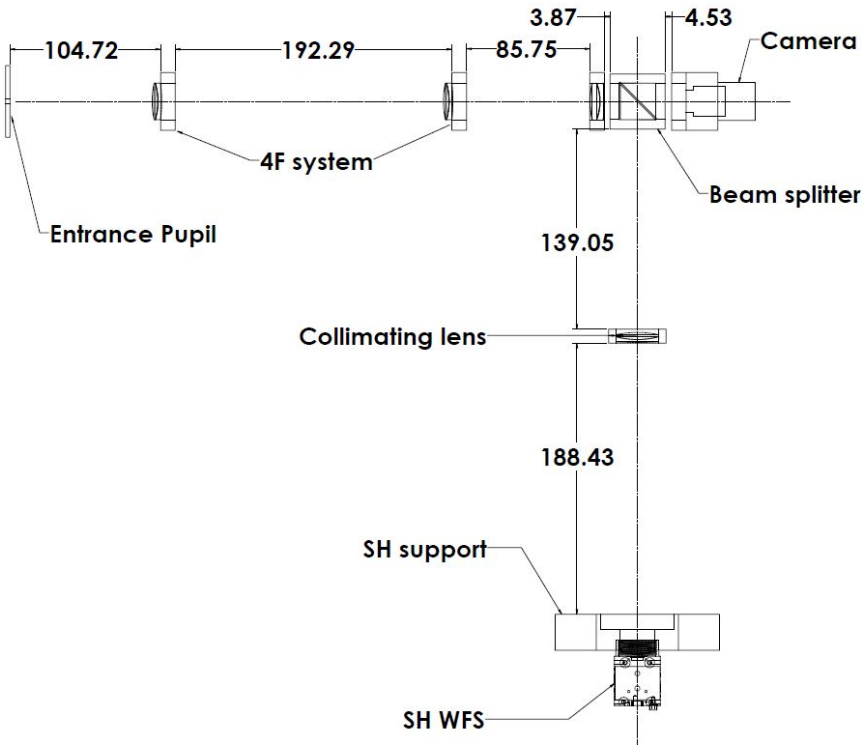
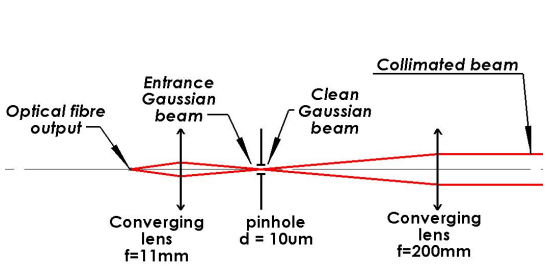
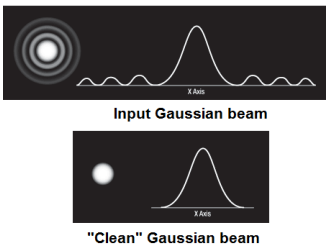


FIGURE 3.1: Experimental setup schema with the relevant distances, (Bouxin, 2017).



(A) Source ray tracing.



(B) Beam view before and after the pinhole, (Thorlabs, 2017).

FIGURE 3.3: Source schema and pinhole effect on the beam.

TABLE 3.1: Optical Components

#	Components	Model	Reference
1	Pigtailed laser diode	Thorlabs, LPS-635-FC	A.1
2	Converging lens, f = 11 mm	Thorlabs, A220TM-A	A.2
3	Pinhole, 10 μm	Thorlabs, P10S	A.3
4	Converging lens, f = 200 mm	Thorlabs, AL100200	A.4
5	3.2 mm Hole milled in metal sheet	...	...
6	Converging lens, f = 100 mm	Thorlabs, AC254-100-A	A.5
7	Converging lens, f = 80 mm		
8	Camera CMOS	Ximea, MQ013MG-E2	A.6
9	Converging lens, f = 100 mm		
10	Shack-Hartman WFS	Thorlabs, WFS150-5C	A.7

3.1.2 Entrance pupil

The entrance pupil of our optical system is a circular aperture of 3.2 mm diameter placed after the collimating lens of the light source. It is milled in a metal plate and centred in his support, to avoid positioning with a XY table. The diameter is chosen in available material to fit in the different detector's surfaces.

3.1.3 Pupil imaging system

The phase diversity technique requires PSFs images as input, which means that the beam as to be focused onto the detector surface. To analyse the aberration in the pupil plane, one needs to focus an image of the beam passing through the entrance pupil. The simplest assembly to achieve this goal is the 4F system, which consist of two converging lenses of focal 100 mm. The two lenses are separated by 200 mm, see Fig. 3.1. This places the image of the entrance pupil 100 mm after the second converging lens.

3.1.4 Detectors

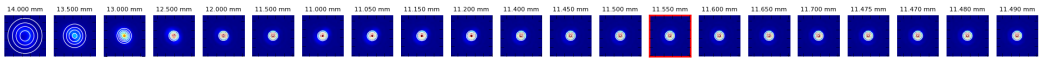
The image of the entrance pupil, obtained with the 4F system, is focused onto a CMOS Ximea camera by a f = 80 mm converging lens to acquire the PSFs for the phase diversity wavefront retrieval. The camera has a surface composed by 1280x1024 pixels of 5.3 μm, see Appendix A.6. It is mounted on sliding support in order to be able to acquire in/out-of-focus images. A beam splitter is placed in the converging beam to separate it in two. The second beam is collimated and a Shack-Hartman WFS is placed on the entrance pupil image plane, to check the results of the phase diversity wavefront retrieval. The Shack-Hartman WFS has a 39 X 31 lenslets grid and a CCD with a resolution of 1280x1024 pixels of 4.65 μm, see Appendix A.7.

3.2 Data Acquisition

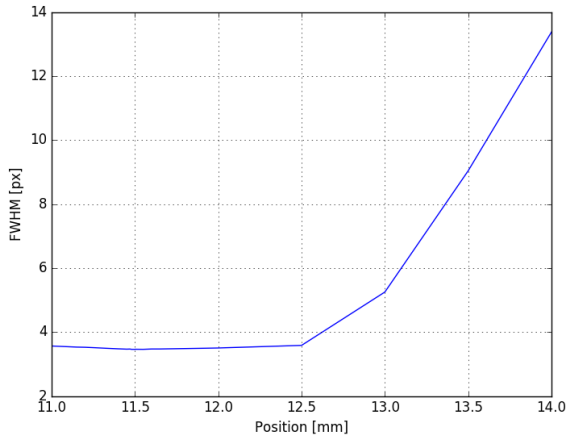
3.2.1 Ximea Camera

The ONERA algorithm takes at least one focused and one defocused PSFs, as described in section 2.2.2. The PSFs are acquired using a python script which uses an open-source library to control the ximea camera, `pyXimea`[1], available on GitHub. The

[1] https://github.com/pupil-labs/pyximea



(A) PSFs taken during the alignment procedure, each image is taken at an other camera position.



(B) FWHM of the PSFs as a function of the camera's position. The minimum is at 11.55 mm

FIGURE 3.4: Example of the results of an alignment procedure

acquisition is done following these steps :

1. The first step in order to acquire PSFs is to determine the position of the camera's focus point using the python script `AlignementScriptXimeaCamera.py`, see Appendix B.2.1. This script let's you acquire consecutively PSFs at different camera's positions and computes their FWHM. It finally returns the minimum FWHM and the camera's position, see Figure 3.4a and 3.4b.

2. Once the focus point position of the camera is determined, the acquisition of the data is possible. The acquisition script is called `AcquisAndSaveXimea.py`, see Appendix B.2.2.

3. The user needs to set the main parameters before acquiring. They are the number of images on which to average, the size of the final PSFs, the position of the focus point on the sliding system and the initial guess to fit the 2D Gaussian on the PSF to find its center. The initial guess can be made using the Ximea camera software, called `xiCamTool`, with which one gets a live feedback of the camera.

4. Then the running program ask the user what he needs to do after having made a sound. The first thing the user needs to do is to place the camera at the focus point and turn on the LED in order to set the optimal exposure time in order to avoid saturation.

5. Finally the acquiring sequence begins, the program ask the user to shut down and turn on the source as the user acquire the different images to get the dark images and the PSF images. The user needs to manually displace the camera to get the defocused PSFs. The program computes the positions to get a 2π P2V defocus dephasing.

Many functions, used in the two scripts described above, are coded in the script `functionsXimea.py`, see Appendix B.2.3.

3.2.2 Shack-Hartmann WFS

The Shack-Hartmann wavefront sensor acquisition is done with the company software. The acquired data are the Zernike coefficients and the reconstructed wavefront computed following the principle described in section 1.4.1. Their is a few parameters that the user can set : the exposure time (there is also an auto set), the number of averaging images (1 up to 300) and the focus points reference of the micro-lenses array.

The data are saved manually in a *.csv* file. I coded an IDL script to read and average the Shack-Hartmann data, in order to be able to analyse them and compare the result with the phase diversity, see Appendix C.1.1 and C.1.2.

3.3 Results

This section presents the results of the phase diversity experiment, with the introduction of different sources of aberration. We will first present the results of the ONERA algorithm test, then we will compare the phase diversity retrieval with a calibrated aberration and finally we will introduce random static aberration with the phase screen and compare the results to the Shack-Hartmann wavefront sensor results.

3.3.1 ONERA Phase Diversity test

The motivation of this test was to better understand the behaviour of the ONERA phase diversity algorithm with respect to the different parameters that could influence its results, such as the noise present in the PSFs, the number of Zernike coefficients returned for the modal mode and the error on the defocused distance of the PSFs.

To study the noise, we acquire PSFs with 25 different numbers of averaging images going from 10 to 5000. The noise level is computed as the ratio between the mean pixel standard deviation and the PSF maximum. The Ximea detector has a noise curve visible in Figure 3.5. The noise curve is computed with the noise level of the focused PSFs. It follows an exponential law with the number of averaging images. The noise level varies from $\sim$ 1e $-$ 3 to $\sim$ 8e $-$ 5, for 10 and 5000 images respectively. Having the noise curve of the detector, we can study empirically how it influences the phase diversity results.

Figures 3.6a and 3.6b presents the results of the 25 different retrievals. As one can see there is some spread due to the different noise levels present in the PSFs given to the algorithm. The biggest standard deviation on a Zernike coefficient is smaller than 6 nm. And there can be up to 20 nm of maximal difference between the a_j's as one can see on Figure 3.6b. The spherical aberration, a_{11}, has the biggest standard deviation and range of values. This shows that the PSFs noise levels have an impact on the retrieval and that not all Zernike coefficient are affected in the same way. One reassuring point is the good correspondence between the modal and zonal retrievals.

Furthermore, looking at the reconstructed wavefronts in Figure 3.7, one can see the effect of noise in the PSFs and how important is the choice of j_{max} for the modal reconstruction. Indeed, for the wavefronts reconstructed using $j_{max} = 30$, the

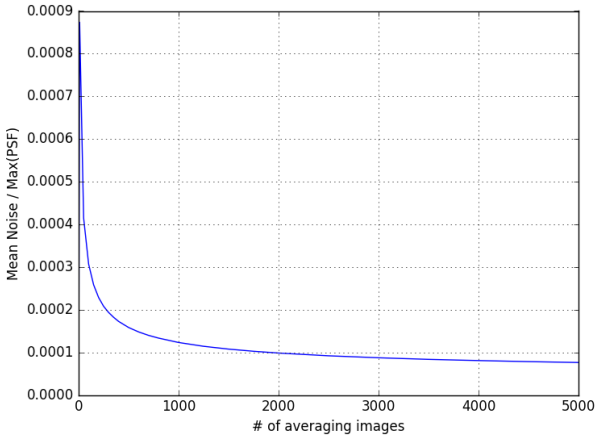
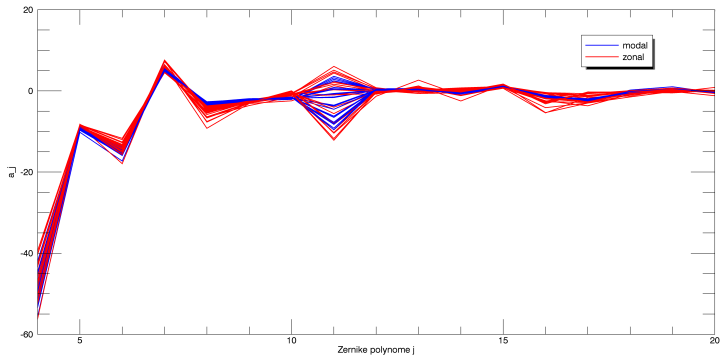
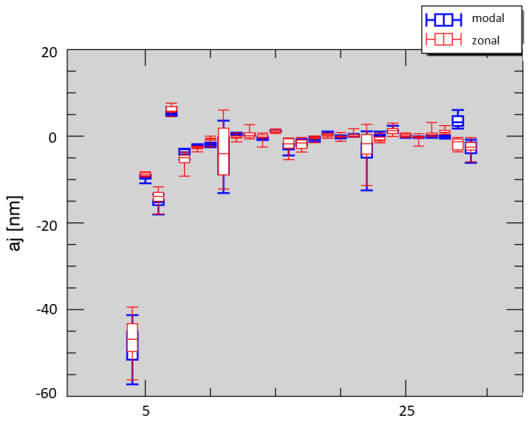


FIGURE 3.5: Noise level as a function of the number of averaging images acquired with $\sim$ 300 μs exposition time. The noise level is computed as the mean of the standard deviation of every pixel divided by the maximum of the focused PSF.

retrieved structures are similar, however for $j_max = 200$ there are more structures in the retrieval done on the PSFs with 10 averaging images. This shows that the noise has a signal at high spatial frequencies. The zonal retrieval shows it clearly, the reconstruction is significantly different between 10 and 3500 averaging images. The averaging over a large number of acquisition smooth out the noise structures that perturbs the reconstruction. Also as said above with the Zernike coefficients is still valid for the wavefront, the two retrieval method gives similar results. But Figure 3.7 shows that the choice of j_{max} is important if one wants to use the reconstructed wavefronts.



(A) Zernike coefficients a_j as a function of j the Zernike index of the 25 modal and zonal phase retrievals, in blue and red respectively.



(B) Boxplot of the 25 Zernike coefficients a_j of the different phase retrievals computed with the different averaging numbers of images as a function of the Zernike index j.

FIGURE 3.6: Results of the phase retrievals computed with the 25 different noise levels. The phase retrievals are done with a $j_{max} = 30$

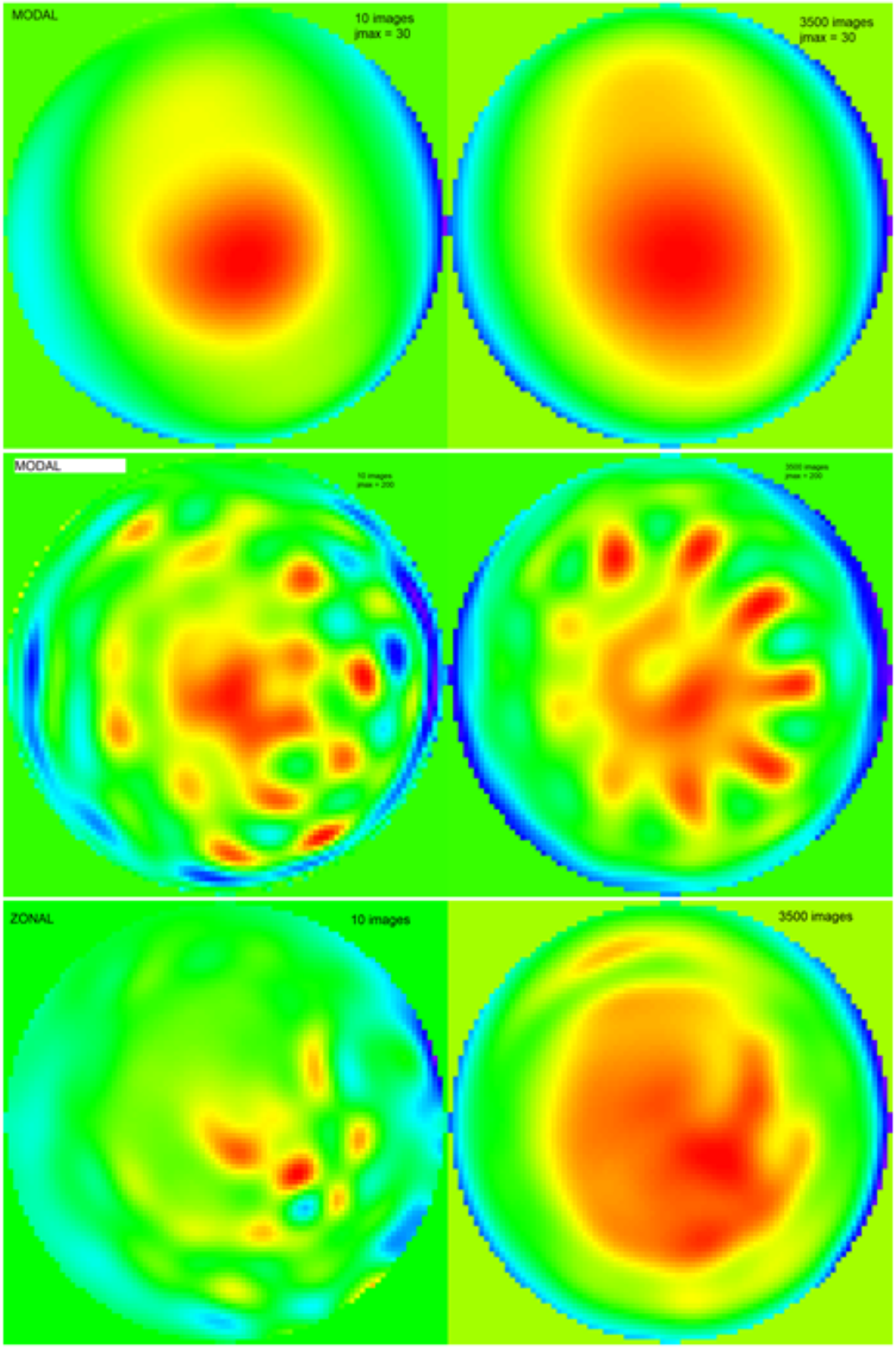


FIGURE 3.7: Reconstructed wavefronts for two different number of averaging images, 10 and 3500, left and right column respectively. The two first line are modal retrieval with $j_{max} = 30$ and $j_{max} = 200$ and the last line is the zonal retrieval.

3.3.2 Parallel plane plate

The first source of aberration studied in this work is a tilted parallel plane plate which is used as a calibrated source of astigmatism.

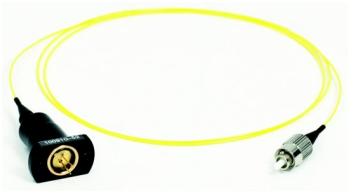
Appendix A

Optical Component Datasheets

A.1 Pigtailed laser diode

THORLABS

Pigtailed Laser Diode, SMF



Description

Thorlabs' Single Mode Pigtailed Laser Diodes are standard TO-packaged diodes that have been pigtailed to a 1 m long single mode fiber with an FC/PC connector. Each unit is tested before shipment. Please refer to the unit-specific test datasheet for optimal operating parameters.

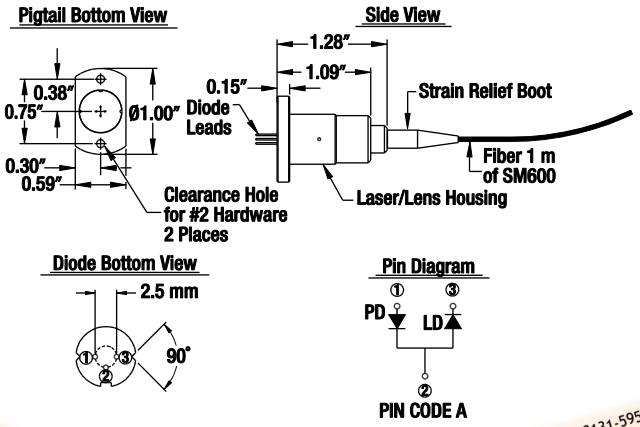
Specifications

LPS-635-FC Specifications		
LD Reverse Voltage (Max)	2 V	
PD Reverse Voltage (Max)	30 V	
Optical Output Power	2.5 mW (Typ.)	3.5 mW (Max)
Operating Temperature	0 to 50 °C	
Storage Temperature	-10 to 65 °C	
Pin Code	9A	
Laser Diode	HL6320G	
Fiber	SM600	
Connector	FC/PC	

LPS-635-FC Specifications			
	Min	Typ.	Max
Wavelength	625 nm	635 nm	640 nm
Threshold Current*	20 mA	50 mA	75 mA
Slope Efficiency*	0.13 mW/mA	0.18 mW/mA	-
Operating Current @ P_O = 2.5 mW*	-	70 mA	95 mA
Operating Voltage @ P_O = 2.5 mW*	-	2.2 V	2.7 V
Monitor Current @ P_O = 2.5 mW*	0.05 mA	0.17 mA	0.3 mA

*Temperature = 25 °C

Drawing



US, Canada, & South America: +1-973-300-3000 | France: +33 (0) 970 444 844 | Europe: +49 (0) 8131-5956-0 | UK & Ireland: +44 (0)1353-654440
Brazil: +55-16-3413 7062 | Scandinavia: +46-31-733-30-00 | Japan & Asia: +81-3-5979-8889 | China: +86 (0)21-60561122

www.thorlabs.com

June 6, 2013
5175-S01, Rev B

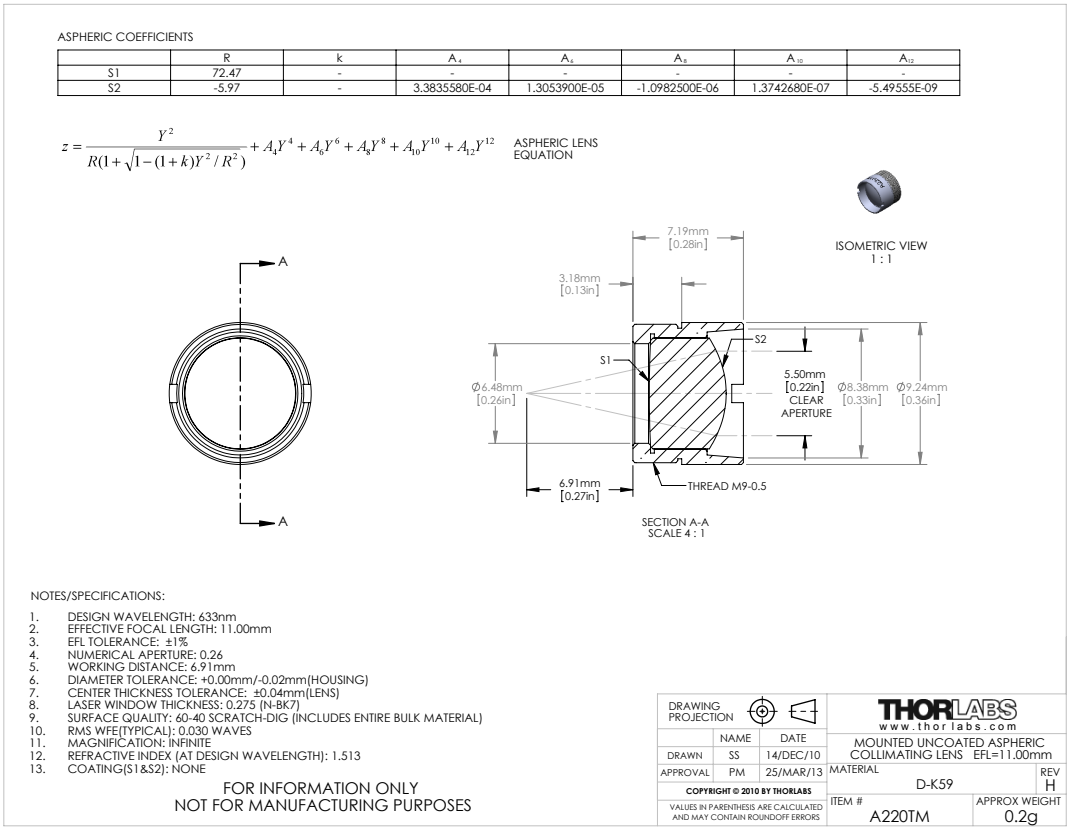
Source : www.thorlabs.com

A.1.1 Power supply modification

The former student, ??? its name ???, mounted a diode driver card to power it. Unfortunately, for the phase diversity experiment, chapter , the power was too high and the Ximea camera was always saturated. So I modified the driver circuit and added two resistances to lower the current so that the detector do not reach the saturation.

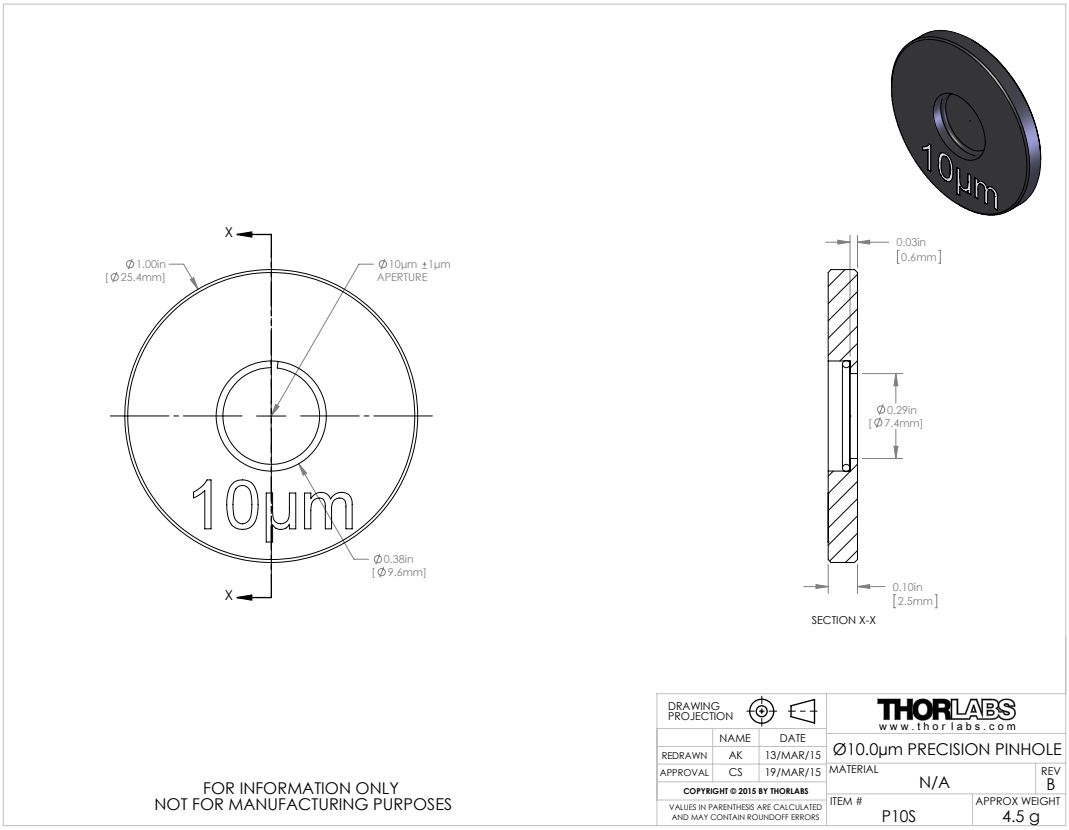
!!! mettre la photo du driver et de la modif !!!

A.2 Converging lens A220TM-A, f = 11 mm



Source : www.thorlabs.com

A.3 Pinhole 10 μm



Source : www.thorlabs.com

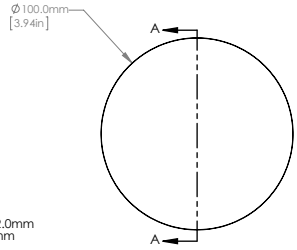
A.4 Converging lens AL100200, f = 200 mm

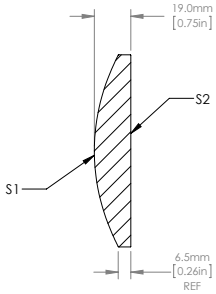
ASPHERIC COEFFICIENTS

	R	k	A_4	A_6	A_8
S1	102.24	-1	4.9646003e-08	7.4017872e-13	9.4141703e-18
S2	INFINITE	-	-	-	-

ASPHERIC LENS EQUATION

$$z = \frac{Y^2}{R\left(1+\sqrt{1-(1+k)Y^2/R^2}\right)} + A_4Y^4 + A_6Y^6 + A_8Y^8$$





NOTES/SPECIFICATIONS:
1. CLEAR APERTURE (COLLIMATION): 92.0mm
2. CLEAR APERTURE (FOCUSING): 89.4mm
3. FOCAL LENGTH: EFL=200mm ±1%
4. Nd: 1.517 ±0.001 Vd: 64.167 ±1.0%
5. DESIGN WAVELENGTH: 780nm
6. NUMERICAL APERTURE: 0.23
7. WORKING DISTANCE (REF): 187.4mm
8. DIAMETER TOLERANCE: +0.00/-0.10mm
9. SURFACE QUALITY: 40-20 SCRATCH-DIG
10. RMS WAVEFRONT ERROR: <0.5µm
11. SAG DEVIATION (POWER)(S1): ±7.5µm
12. RMS IRREGULARITY (S1): <3 FRINGES
13. CENTRATION: <3arcmin
14. MAGNIFICATION: INFINITE

FOR INFORMATION ONLY
NOT FOR MANUFACTURING PURPOSES

DRAWING PROJECTION		
	NAME	DATE
DRAWN	NE	10/OCT/14
APPROVAL	BW	18/NOV/14

VALUES IN PARENTHESIS ARE CALCULATED AND MAY CONTAIN ROUNDOFF ERRORS

THORLABS
www.thorlabs.com

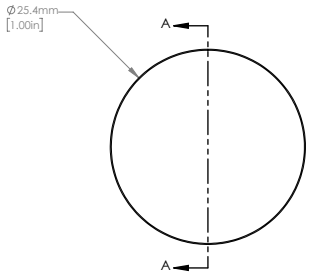
Ø100.0mm ASPHERIC LENS,
NA=0.23, f=200mm

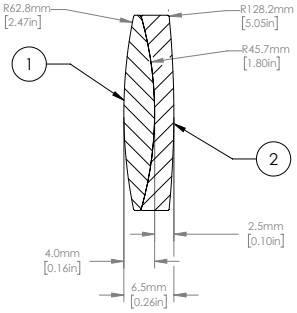
MATERIAL	REV
N-BK7	A
ITEM #	APPROX WEIGHT
AL100200	0.6 kg

Source : www.thorlabs.com

A.5 Converging lens AC254-100-A, f = 100 mm

	MATERIAL
1	N-BK7
2	SF5





NOTES/SPECIFICATIONS:

1. DESIGN WAVELENGTH: 486.1nm, 587.6nm, & 656.3nm
2. FOCAL LENGTH: 100.1mm ±1%
3. BACK FOCAL LENGTH (REF): 97.2mm
4. CLEAR APERTURE: > Ø22.86mm
5. SURFACE QUALITY: 40-20 SCRATCH-DIG
6. CENTRATION: <3arcmin
7. DIAMETER TOLERANCE: +0.0/-0.1mm
8. THICKNESS TOLERANCE: ±0.20mm
9. COATING: BBAR Ravg <0.5% FROM 400-700nm, 0° AOI, ON OUTER OPTICAL SURFACES

FOR INFORMATION ONLY
NOT FOR MANUFACTURING PURPOSES

DRAWING PROJECTION		
	NAME	DATE
DRAWN	KMR	20/SEP/16
APPROVAL	DD	26/SEP/16

VALUES IN PARENTHESIS ARE CALCULATED AND MAY CONTAIN ROUNDOFF ERRORS

THORLABS
www.thorlabs.com

Ø25.4mm, f=100.0mm
VISIBLE ACHROMAT, -A COAT

MATERIAL	REV
SEE TABLE	C
ITEM #	APPROX WEIGHT
AC254-100-A	9.47 g

Source : www.thorlabs.com

A.6 Ximea Camera, MQ013MG-E2

Specifications:	
Resolution:	1.3 MP 1280 × 1024 pixels
Sensor type:	CMOS Matrix B/W
Sensor model:	e2V EV76C560 ABT-EQV
Sensor size:	1/1.8"
Sensor active area:	6.9 × 5.5 mm
Pixel size:	5.3 µm
Bits per pixel:	8, 10
Dynamic range:	60 dB
Frame rates:	60 fps
On-chip binning:	1x1, 2x2
Image data interface:	USB 3.0
Data I/O:	GPIO IN, OUT
Power requirements:	0.9 Watt
Lens mount:	C or CS Mount
Weight:	26 grams
Dimensions WxHxD:	26 x 26 x 26 mm
Operating environment:	50 °C
Customs tariff code:	8525.80 30 (EU) / 8525.80 40 (USA)
ECCN:	EAR99

Source : www.ximea.com/en/products/usb3-vision-cameras-xiq-line/mq013mg-e2

A.7 Shack-Hartmann wavefront sensor, WFS150-5C

8.1 Technical Data

8.1.1 WFS150/300

Item #	WFS150-5C	WFS150-7AR	WFS300-14AR
Microlenses			
Microlens Array	MLA150M-5C	MLA150M-7AR	MLA300M-14AR
Substrate Material	Fused Silica (Quartz)		
Number of Active Lenslets	Software Selectable		
Max. Number of Lenslets	39 x 31		19 x 15
Camera			
Sensor Type	CCD		
Resolution	max. 1280 x 1024 pixels, Software Selectable		
Aperture Size	5.95 mm x 4.76 mm		
Pixel Size	4.65 µm x 4.65 µm		
Shutter	Global		
Exposure Range	79 µs - 65 ms		
Frame Rate	max. 15 Hz		
Image Digitization	8 bit		
Wavefront Measurement			
Wavefront Accuracy [1])	λ/15 rms @ 633 nm		λ/50 rms @ 633 nm
Wavefront Sensitivity [2])	λ/50 rms @ 633 nm		λ/150 rms @ 633 nm
Wavefront Dynamic Range [3])	> 100 λ @ 633 nm		> 50 λ @ 633 nm
Local Wavefront Curvature [4])	> 7.4 mm	> 10.0 mm	> 40.0 mm
External Trigger Input			
Save Static Voltage level	0 to 30 V DC		
LOW Level	0.0 V to 2.0 V		
HIGH Level	5.0 V to 24 V		
Input current	> 10 mA		
Min Pulse Width	100 µs		
Min. Slew Rate	35 V / msec		
Common Specifications			
Optical Input	C-Mount		
Power Supply	<1.5 W, via USB		
Operating Temperature Range [5])	+5 to +35 °C		
Storage Temperature Range	-40 to 70 °C		
Warm-Up Time for Rated Accuracy	15 min		
Dimensions (W x H x D)	32.0 mm x 40.4 mm x 45.5 mm		
Weight	0.1 kg		

[1]) Absolute accuracy using internal reference. Measured for spherical wavefronts of known RoC.
[2]) Typical relative accuracy. Achievable after, and with respect to a user calibration, 10 image averages
[3]) Over entire aperture of wavefront sensor
[4]) Radius of wavefront curvature over single lenslet aperture
[5]) non-condensing

All technical data are valid at 23 ± 5°C and 45 ± 15% rel. humidity (non condensing)

Source : WFS Series Operation Manual, www.thorlabs.com

Appendix B

Python Code

B.1 Phase Diversity analytical algorithm code

B.1.1 phaseDiversity3PSFs.py

```
#Class phaseDiversity object to retrieve the phase in the pupil from two psfs in/out of focus

import numpy as np
import fs
import myExceptions
import PSF as psf

class phaseDiversity3PSFs(object):

    def __init__(self,inFoc,outFocpos,outFocneg,deltaZ,lbda,pxsize,F,pupilRadius,jmin,jmax):
#
#       input:
#       inFoc,outFocpos,outFocneg are the 3 squared PSFs data, (focused, defocused positiv and defocused negative)
#       deltaZ is the displacement of the detector to acquire the two defocused PSFs
#       lbda is the wavelength of the incoming light
#       pxsize is the pixel size of the detector
#       F is the focal length of the imaging system
#       pupilRadius is the radius of the exit pupil
#       jmin and jmax gives the boundary on the js to retrieve

        print 'phaseDiversity ...'

        #PSF
        self.inFoc = inFoc
        self.outFocpos = outFocpos
        self.outFocneg = outFocneg
        shapeinFoc = np.shape(self.inFoc)
        shapeoutFocpos = np.shape(self.outFocpos)
        shapeoutFocneg = np.shape(self.outFocneg)
        if shapeinFoc == shapeoutFocpos and shapeinFoc == shapeoutFocneg:
            self.shape = shapeinFoc
        else:
            raise myExceptions.PSFssizeError('the shape of the in/out PSFs is not the same',[shapeinFoc,shapeoutFocpos])
        if shapeinFoc[0]==shapeinFoc[1] and np.mod(shapeinFoc[0],2)==0:
            self.N = shapeinFoc[0]
```

```
36          else:
37              raise myExceptions.PSFssizeError('Either PSF is not square or
                    mod(N,2) != 0',shapeinFoc)
38
39          # properties
40          self.deltaZ = deltaZ
41          self.lbda = lbda
42          self.pxsize = pxsize
43          self.F = F
44          self.pupilRadius = pupilRadius
45          self.dxp = self.F*self.lbda/(self.N*self.pxsize)
46          self.rad = int(np.ceil(self.pupilRadius/self.dxp))
47          if 2*self.rad > self.N/2.:
48              raise myExceptions.PupilSizeError('Npupil (2*rad) is bigger than
                    N/2 which is not correct for the fft computation',[])
49          self.NyquistCriterion()
50          self.jmin = jmin
51          self.jmax = jmax
52          self.oddjs = fs.getOddJs(self.jmin,self.jmax)
53          self.evenjs = fs.getEvenJs(self.jmin,self.jmax)
54
55          # result computation
56          self.result = self.retrievePhase()
57
58      def NyquistCriterion(self):
59          deltaXfnyq = 0.5*self.lbda/(2*self.pupilRadius)
60          deltaXf = self.pxsize/self.F
61          if deltaXfnyq < deltaXf : raise myExceptions.NyquistError('the
                system properties do not respect the nyquist criterion',[])
62
63      def retrievePhase(self):
64          y1,A1 = self.initiateMatrix1()
65          results1 = np.linalg.lstsq(A1,y1)
66          ajsodd = results1[0]
67          A1tA1inv= np.linalg.inv(np.matmul(np.transpose(A1),A1))
68          n_p = y1.size-1
69          ste1 = np.sqrt(results1[1]/n_p * np.diagonal(A1tA1inv))
70
71          deltaphi =
                fs.deltaPhi(self.N,self.deltaZ,self.F,2*self.pupilRadius,self.lbda,self.dxp)
72          y2,A2 = self.initiateMatrix2(deltaphi)
73          results2 = np.linalg.lstsq(A2,y2)
74          ajseven = results2[0]
75          A2tA2inv= np.linalg.inv(np.matmul(np.transpose(A2),A2))
76          n_p = y2.size-1
77          ste2 = np.sqrt(results2[1]/n_p * np.diagonal(A2tA2inv))
78
79          js = np.append(self.oddjs,self.evenjs)
80          ajs = np.append(ajsodd,ajseven)
81          ajsSte = np.append(ste1,ste2)
82
83          Ixjs = np.argsort(js)
84          result = {'js': js[Ixjs], 'ajs': ajs[Ixjs], 'ajsSte': ajsSte[Ixjs]}
                #,'wavefront':phase}
85          return result
86
87      def initiateMatrix1(self):
```

```
88          deltaPSFinFoc = self.CMPTEdeltaPSF()
89          y1 = fs.y1(deltaPSFinFoc)
90          A1 = np.zeros((self.N**2,len(self.oddjs)))
91          for ij in np.arange(len(self.oddjs)):
92              phiJ = fs.f1j(self.oddjs[ij],self.N,self.dxp,self.pupilRadius)
93              A1[:,ij] = phiJ
94          return y1,A1
95
96      def initiateMatrix2(self,deltaphi):
97          deltaPSFoutFocpos,deltaPSFoutFocneg = self.CMPTEdeltaPSF(self.deltaZ)
98          y2 =
                fs.y2even(fs.getEvenPart(deltaPSFoutFocpos),fs.getEvenPart(deltaPSFoutFocneg))
99          A2 = np.zeros((self.N**2,len(self.evenjs)))
100         for ij in np.arange(len(self.evenjs)):
101             phiJ =
                    fs.f2jeven(self.evenjs[ij],self.N,deltaphi,self.dxp,self.pupilRadius)
102             A2[:,ij] = phiJ
103         return y2,A2
104
105     def CMPTEdeltaPSF(self,deltaZ=[]):
106         if not deltaZ:
107             PSF = psf.PSF([1],[0],self.N,self.dxp,self.pupilRadius)
108             return PSF.Sp**2*self.inFoc - PSF.Sp**2*PSF.PSF
109         else:
110             P2Vdephasing =
                    np.pi*self.deltaZ/self.lbda*(2*self.pupilRadius/self.F)**2/4.
111             a4 = P2Vdephasing/2./np.sqrt(3)
112             PSF = psf.PSF([4],[a4],self.N,self.dxp,self.pupilRadius)
113             return [PSF.Sp**2*self.outFocpos -
                    PSF.Sp**2*PSF.PSF,PSF.Sp**2*self.outFocneg -
                    PSF.Sp**2*PSF.PSF]
```

B.1.2 fs.py

```
1  #functions to compute the matrix element of the system of equations Ax = b
2  import zernike as Z
3  import phasor as ph
4  import numpy as np
5
6  def f1j(j,N,dxp,pupilRadius): # 1: phij's of matrix A to find a_j odd
7      Zj = Z.calc_zern_j(j,N,dxp,pupilRadius)
8      FFTZj = scaledfft2(Zj,dxp)
9
10     Lp = N*dxp
11     xp = np.arange(-Lp/2,Lp/2,dxp)
12     yp = xp
13
14     [Xp,Yp]=np.meshgrid(xp,yp)
15
16     pupil = np.float64(np.sqrt(Xp**2+Yp**2)<=pupilRadius)
17     FFTPupil = scaledfft2(pupil,dxp)
18
19     return np.ravel(2 * np.real(FFTPupil) * np.imag(FFTZj))
20
```

```
21 def f2j(j,N,jsodd,ajsodd,deltaphi,dxp,pupilRadius): # 2: phij's of matrix A
       to find a_j even
22     #Get the jth zernike polynomials values on a circular pupil of radius rad
23     Zj = Z.calc_zern_j(j,N,dxp,pupilRadius)
24
25     #compute the different 2Dfft given in the equations of deltaPSF
26     cosZj = np.cos(deltaphi)*Zj
27     sinZj = np.sin(deltaphi)*Zj
28     FFTcosZj = scaledfft2(cosZj,dxp)
29     FFTsinZj = scaledfft2(sinZj,dxp)
30
31     #odd phase
32     oddPhasor = ph.phasor(jsodd,ajsodd,N,dxp,pupilRadius)
33     oddPhase = oddPhasor.phase
34     pupil = oddPhasor.pupil
35     cosOddPhase = pupil*np.cos(deltaphi)*oddPhase
36     sinOddPhase = pupil*np.sin(deltaphi)*oddPhase
37     FFTcosOddPhase = scaledfft2(cosOddPhase,dxp)
38     FFTsinOddPhase = scaledfft2(sinOddPhase,dxp)
39
40     #2Dfft of pupil function times sin(deltaPhi) and cos(deltaPhi)
41     pupilSin = pupil*np.sin(deltaphi)
42     pupilCos = pupil*np.cos(deltaphi)
43     FFTPupilSin = scaledfft2(pupilSin,dxp)
44     FFTPupilCos = scaledfft2(pupilCos,dxp)
45
46     FFTsinZjOddPhase = scaledfft2(sinZj*oddPhase,dxp)
47     FFTcosZjOddPhase = scaledfft2(cosZj*oddPhase,dxp)
48
49     return np.ravel(2*np.imag(np.conj(FFTcosZj) * FFTsinOddPhase + FFTsinZj
           * np.conj(FFTcosOddPhase)
50             - np.conj(FFTPupilCos) * FFTsinZjOddPhase + np.conj(FFTPupilSin)
                   * FFTcosZjOddPhase))
51
52 def f2jeven(j,N,deltaphi,dxp,pupilRadius): # 2: phij's of matrix A to find
       a_j even
53     #Get the jth zernike polynomials values on a circular pupil of radius rad
54     Zj = Z.calc_zern_j(j,N,dxp,pupilRadius)
55     Phasor = ph.phasor([1],[0],N,dxp,pupilRadius)
56     pupil = Phasor.pupil
57     #compute the different 2Dfft given in the equations of deltaPSF
58     cosZj = pupil*np.cos(deltaphi)*Zj
59     sinZj = pupil*np.sin(deltaphi)*Zj
60     FFTcosZj = scaledfft2(cosZj,dxp)
61     FFTsinZj = scaledfft2(sinZj,dxp)
62     #2Dfft of pupil function times sin(deltaPhi) and cos(deltaPhi)
63     pupilSin = pupil*np.sin(deltaphi)
64     pupilCos = pupil*np.cos(deltaphi)
65     FFTPupilSin = scaledfft2(pupilSin,dxp)
66     FFTPupilCos = scaledfft2(pupilCos,dxp)
67
68     return
           np.ravel(-4*np.real(np.conj(FFTPupilCos)*FFTsinZj-np.conj(FFTPupilSin)*FFTcosZj))
69
70 def y1(deltaPSFinFoc): #1: yi's of y to find a_j odd
71     #compute the odd part of delta PSF
72     oddDeltaPSF = getOddPart(deltaPSFinFoc)
```

```
73      return np.ravel(oddDeltaPSF)
74
75  def y2(deltaPSFoutFoc,N,jsodd,ajsodd,deltaphi,dxp,pupilRadius): # 2: yi's
        of y to find a_j even
76      oddDeltaPSF = getOddPart(deltaPSFoutFoc)
77      oddPhasor = ph.phasor(jsodd,ajsodd,N,dxp,pupilRadius)
78      oddPhase = oddPhasor.phase
79
80      pupilSin = oddPhasor.pupil*np.sin(deltaphi)
81      pupilCos = oddPhasor.pupil*np.cos(deltaphi)
82      FFTPupilSin = scaledfft2(pupilSin,dxp)
83      FFTPupilCos = scaledfft2(pupilCos,dxp)
84      cosOddPhase = oddPhasor.pupil*np.cos(deltaphi)*oddPhase
85      sinOddPhase = oddPhasor.pupil*np.sin(deltaphi)*oddPhase
86      FFTcosOddPhase = scaledfft2(cosOddPhase,dxp)
87      FFTsinOddPhase = scaledfft2(sinOddPhase,dxp)
88
89      return np.ravel(oddDeltaPSF -
            2*np.real(np.conj(FFTPupilCos))*np.imag(FFTcosOddPhase)
90              - 2*np.real(np.conj(FFTPupilSin))*np.imag(FFTsinOddPhase))
91
92  def y2even(deltaPSFoutFocpos,deltaPSFoutfocneg): #3: yi's of y to find a_j
        even
93      return np.ravel(deltaPSFoutFocpos-deltaPSFoutfocneg)
94
95  #Other
        functions----------------------------------------------------------------------
96  def flipMatrix(M):
97      #flip 2D matrix along x and y
98      dimM = (np.shape(M))[0]
99      Mflipped = np.flipud(np.fliplr(M))
100     if np.mod(dimM,2)==0:
101         Mflipped = np.roll(Mflipped,1,axis=0)
102         Mflipped = np.roll(Mflipped,1,axis=1)
103     return Mflipped
104 def getOddPart(F):
105     oddF = (F - flipMatrix(F))/2.
106     return oddF
107 def getEvenPart(F):
108     evenF = (F + flipMatrix(F))/2.
109     return evenF
110 def getOddJs(jmin,jmax):
111     js = []
112     for j in np.arange(jmin,jmax+1):
113         [n,m] = Z.noll_to_zern(j)
114         if np.mod(m,2) == 1:
115             js.append(j)
116         else:
117             continue
118     return np.array(js)
119 def getEvenJs(jmin,jmax):
120     js = []
121     for j in np.arange(jmin,jmax+1):
122         [n,m] = Z.noll_to_zern(j)
123         if np.mod(m,2) == 0:
124             js.append(j)
125         else:
```

```
126             continue
127     return np.array(js)
128 def deltaPhi(N,deltaZ,F,D,wavelength,dxp):
129     Zj = Z.calc_zern_j(4,N,dxp,D/2.)
130     P2Vdephasing = np.pi*deltaZ/wavelength*(D/F)**2/4.
131     a4defocus = P2Vdephasing/2/np.sqrt(3)
132     return a4defocus*Zj
133 def cleanZeros(A,threshold):
134     A[np.abs(A) < threshold] = 0.
135     return A
136 def scaledfft2(f,dxp):
137     return np.fft.ifftshift(np.fft.fft2(np.fft.fftshift(f)))*dxp**2
138 def RMSE(estimator,target):
139     return np.sqrt(np.mean((estimator-target)**2))
140 def BIAS(estimator,target):
141     return np.mean((estimator-target))
142
143 def RMSwavefrontError(js,ajs):
144     if 1 in js:
145         return np.sqrt(np.sum(ajs**2)-ajs[js==1]**2)
146     else:
147         return np.sqrt(np.sum(ajs**2))
```

B.1.3 myExceptions.py

```
1  class PSFssizeError(ValueError):
2     '''Raise when the size of the PSFs are not correct'''
3     def __init__(self, message, foo, *args):
4         self.message = message
5         self.foo = foo
6         super(PSFssizeError, self).__init__(message, foo, *args)
7
8  class NyquistError(ValueError):
9     '''Raise when the PSFs properties do not respect the nyquist criterion'''
10    def __init__(self, message, foo, *args):
11        self.message = message
12        self.foo = foo
13        super(NyquistError, self).__init__(message, foo, *args)
14
15 class PupilSizeError(ValueError):
16     '''Raise when Npupil is bigger than the size of the PSF N'''
17     def __init__(self, message, foo, *args):
18         self.message = message
19         self.foo = foo
20         super(PupilSizeError, self).__init__(message, foo, *args)
```

B.1.4 zernike.py

```
1  import numpy as np
2
3  def calc_zern_j(j, N, dxp, pupilRadius):
4
5      Lp = N*dxp
6
```

```
    if (j <= 0):
        return {'modes':[], 'modesmat':[], 'covmat':0, 'covmat_in':0,
            'mask':[[0]]}
    if (N <= 0):
        raise ValueError("N should be > 0")
    if (dxp <= 0 or dxp >= N):
        raise ValueError("dxp should be > 0 or < N")

    xp = np.arange(-Lp/2,Lp/2,dxp)
    yp = xp
    [Xp,Yp]=np.meshgrid(xp,yp)
    r = np.sqrt(Xp**2+Yp**2)
    r = r*(r<=pupilRadius)/pupilRadius
    pup = np.float64(np.sqrt(Xp**2+Yp**2)<=pupilRadius)
    theta = np.arctan2(Yp,Xp)

    Zj = zernike(j,r,theta)*pup
    return Zj

def zernike(j,r,theta):
    n,m = noll_to_zern(j)
    nc = (2*(n+1)/(1+(m==0)))**0.5
    #nc = (2*(n+1)/(1+(m==0)))**0.5
    if (m > 0): return nc*zernike_rad(m, n, r) * np.cos(m * theta)
    if (m < 0): return nc*zernike_rad(-m, n, r) * np.sin(-m * theta)
    return nc*zernike_rad(0, n, r)

def zernike_rad(m, n, r):
    if (np.mod(n-m, 2) == 1):
        return r*0.0
    wf = r*0.0
    for k in range((n-m)/2+1):
        wf += r**(n-2.0*k) * (-1.0)**k * fac(n-k) / ( fac(k) * fac(
            (n+m)/2.0 - k ) * fac( (n-m)/2.0 - k ) )
    return wf

def noll_to_zern(j):
    j = int(j)
    if (j == 0):
        raise ValueError("Noll indices start at 1, 0 is invalid.")
    n = 0
    j1 = j-1
    while (j1 > n):
        n += 1
        j1 -= n
    m = (-1)**j * ((n % 2) + 2 * int((j1+((n+1)%2)) / 2.0 ))
    return (n, m)
def fac(n):
    if n == 0:
        return 1
    else:
        return n * fac(n-1)
```

B.1.5 phasor.py

```python
#class phasor
import numpy as np
import zernike as Z

class phasor(object):

    def __init__(self,js=[1],ajs=[0],N=800,dxp=1,pupilRadius = 200):
        self.js = js
        self.ajs = ajs
        self.N = N
        self.dxp = dxp
        self.pupilRadius = pupilRadius

        self.pupil = self.constructPupil()

        self.phase = self.constructPhase()
        self.phasor = self.pupil*np.exp(-1j*self.phase)

    def constructPhase(self):
        phase = np.zeros((self.N,self.N))
        for ij, j in enumerate(self.js):
            Zj = Z.calc_zern_j(j,self.N,self.dxp,self.pupilRadius)
            phase += self.ajs[ij]*Zj
        return phase

    def constructPupil(self):
        Lp = self.N*self.dxp

        xp = np.arange(-Lp/2,Lp/2,self.dxp)
        yp = xp

        [Xp,Yp]=np.meshgrid(xp,yp)

        pup = np.float64(np.sqrt(Xp**2+Yp**2)<=self.pupilRadius)
        return pup
```

B.2 Acquisition Code: Ximea Camera

B.2.1 AlignementScriptXimeaCamera.py

```python
##Script to compute the FWHM of the beam on the camera averaging
 #over "nbrImgAveraging" images and see which position minimizes it.

from ximea import xiapi
import numpy as np
from matplotlib import pyplot as plt
import scipy.optimize as opt
import datetime
import functionsXimea as fX
import seaborn as sns
import os
sns.set()
#%% instanciation

dataFolderPath = '...'
```

```
plotFolderPath = '...'
#create the matrix grid of the detector CCD
x = np.linspace(0,1280,1280)
y = np.linspace(0,1024,1024)
x, y = np.meshgrid(x, y)

#initial guess for the fit depending on the position of the beam in the CCD
initial_guess = [250, 481, 706, 3, 3] # [max PSF,y,x,sigmay,sigmax]

#number of image to average
nbrImgAveraging = 10

#%%data acquisition and treatment

#create instance for first connected camera
cam = xiapi.Camera()
#start communication
print('Opening camera...')
cam.open_device()
#settings
cam.set_imgdataformat('XI_MONO8') #XIMEA format 8 bits per pixel
cam.set_gain(0)
#create instance of Image to store image data and metadata
img = xiapi.Image()
#start data acquisition
print('Starting data acquisition...')
if cam.get_acquisition_status() == 'XI_OFF':
        cam.start_acquisition()

cam.set_exposure(fX.determineUnsaturatedExposureTime(cam,img,[60,10000],1))

#instanciation for the while loop
answer ='y'
i=0
relativePos = []
data = []
data_fitted = []
FWHMx = []
FWHMy = []
x0 = []
y0 = []
sigmaX0 = []
sigmaY0 = []

while answer == 'y':

    try:
        relativePos.append(float(raw_input('What is the position on the
            screw [mm] ? ')))
    except ValueError:
        print('Not a float number')

    [tmpdata,stdData] = fX.acquireImg(cam,img,nbrImgAveraging)
    data.append(tmpdata)
    #Fit the img data on the 2D Gaussian to compute the FWHM
    print('Fitting 2D Gaussian...')
```

```
71      popt, pcov = opt.curve_fit(fX.TwoDGaussian, (x,y), data[i].ravel(), p0 =
            initial_guess)
72      print('Fitting done')
73
74      FWHMx.append(2*np.sqrt(2*np.log(2))*popt[3])
75      FWHMy.append(2*np.sqrt(2*np.log(2))*popt[4])
76      x0.append(popt[2])
77      sigmaX0.append(popt[4])
78      y0.append(popt[1])
79      sigmaY0.append(popt[3])
80
81      print 'Fig %d : (x,y) = (%3.2f,%3.2f), FWHM x = %3.2f, FWHM y = %3.2f'
            %(i,x0[i],y0[i],FWHMx[i],FWHMy[i])
82
83      data_fitted.append(fX.TwoDGaussian((x, y),
            popt[0],popt[1],popt[2],popt[3],popt[4]).reshape(1024, 1280))
84
85      #plot the beamspot
86      fig, ax = plt.subplots(1, 1)
87      ax.imshow(data[i], cmap=plt.cm.jet,origin='bottom',
88          extent=(x.min(), x.max(), y.min(), y.max()))
89      ax.contour(x, y, data_fitted[i], 5, colors='w',linewidths=0.8)
90      plt.xlim( (popt[2]-4*popt[4], popt[2]+4*popt[4]) )
91      plt.ylim( (popt[1]-4*popt[3], popt[1]+4*popt[3]) )
92      plt.show()
93
94      #ask if the person wants to acquire a new image to improve the alignement
95      pressedkey = raw_input('Do you want to acquire an other image [y (yes)
            or n (no)]: ')
96      if (pressedkey =='n'):
97          answer = pressedkey
98      #increase i
99      i+=1
100
101 #stop data acquisition
102 print('Stopping acquisition...')
103 cam.stop_acquisition()
104
105 #stop communication
106 cam.close_device()
107
108 #convert list to np.array
109 relativePos = np.array(relativePos)
110 data = np.array(data)
111 FWHMx = np.array(FWHMx)
112 FWHMy = np.array(FWHMy)
113 x0 = np.array(x0)
114 y0 = np.array(y0)
115 sigmaX0 = np.array(sigmaX0)
116 sigmaY0 = np.array(sigmaY0)
117
118 #plot the FWHM vs. relPos
119 fig, ax = plt.subplots(1,1)
120 ind = np.argsort(relativePos)
121 ax.plot(relativePos[ind],(np.sqrt(FWHMx**2+FWHMy**2))[ind])
122 ax.set_xlabel('Position [mm]')
123 ax.set_ylabel('FWHM [px]')
```

```
124 ax.grid()
125 date = datetime.datetime.today()
126 if not os.path.isdir(plotFolderPath):
127     os.makedirs(plotFolderPath)
128 plt.savefig(plotFolderPath+date.strftime('%Y%m%d%H%M%S')+'FWHM_pos.pdf')
129 plt.savefig(plotFolderPath+date.strftime('%Y%m%d%H%M%S')+'FWHM_pos.png')
130
131
132 indOfMinFWHM = np.argmin(np.sqrt(FWHMx**2+FWHMy**2))
133
134 fig, axarr = plt.subplots(1,np.size(data,0))
135 #plot all the images besides each other
136 for iImg in ind:
137     axarr[iImg].imshow(data[iImg], cmap=plt.cm.jet,origin='bottom',
138         extent=(x.min(), x.max(), y.min(), y.max()))
139 #    axarr[iImg].contour(x, y, data_fitted[iImg], 5,
    colors='w',linewidths=0.8)
140     axarr[iImg].set_xlim( (x0[iImg]-12, x0[iImg]+12) )
141     axarr[iImg].set_ylim( (y0[iImg]-12, y0[iImg]+12) )
142     axarr[iImg].set_yticklabels('',visible=False)
143     axarr[iImg].set_xticklabels('',visible=False)
144     axarr[iImg].set_title('%5.3f mm'%relativePos[iImg],fontsize=8)
145     if iImg == indOfMinFWHM:
146         axarr[iImg].set_frame_on(True)
147         for pos in ['top', 'bottom', 'right', 'left']:
148             axarr[iImg].spines[pos].set_edgecolor('r')
149             axarr[iImg].spines[pos].set_linewidth(2)
150     else:
151         axarr[iImg].set_frame_on(False)
152 plt.show()
153 date = datetime.datetime.today()
154
155 plt.savefig(plotFolderPath+date.strftime('%Y%m%d%H%M%S')+'ImgPSF.pdf')
156 plt.savefig(plotFolderPath+date.strftime('%Y%m%d%H%M%S')+'ImgPSF.png')
157
158
159 #save data
160 if not os.path.isdir(dataFolderPath):
161     os.makedirs(dataFolderPath)
162 date = datetime.datetime.today()
163 np.save(dataFolderPath+date.strftime('%Y%m%d%H%M%S')+'data.npy',data)
164 np.save(dataFolderPath+date.strftime('%Y%m%d%H%M%S')+'relativePos.npy',relativePos)
```

B.2.2 AcquisAndSaveXimea.py

```
1 #%% Script to acquire images average over nbrImgAveraging images and save
      them into fits file
2
3 from ximea import xiapi
4 import datetime
5 import functionsXimea as fX
6 import winsound
7 import numpy as np
8
```

```
#%%instanciation
    ------------------------------------------------------------
#number of image to average
nbrImgAveraging = 5000
numberOfFinalImages = 1

#Cropping information
sizeImg = 256

#Parameter of camera and saving
folderPathCropped = 'data//cropped/20/'
darkFolderPathCropped = '.data/dark//cropped/20/'
folderPathFull = 'data/full/'
darkFolderPathFull = '/data/dark//full/'
nameCamera = 'Ximea'
focusPos = 11.63

#Sound
duration = 1000 # millisecond
freq = 2000 # Hz

#initial guess for the fit depending on the position of the beam in the CCD
initial_guess = [250, 468, 954, 3, 3] # [max PSF,y,x,sigmay,sigmax]

#-------------------------------------------------------------------------------
#%% data acquisition
    ------------------------------------------------------------

#Opening the connection to the camera
cam = xiapi.Camera()
cam.open_device()
cam.set_imgdataformat('XI_MONO8') #XIMEA format 8 bits per pixel
cam.set_gain(0)

img = xiapi.Image()
if cam.get_acquisition_status() == 'XI_OFF':
    cam.start_acquisition()
#%% exposition
cond = 1
while bool(cond):
    source = ''
    winsound.Beep(freq, duration)
    source = int(raw_input('Is the source turned on and at focus point
        (usually %5.3f mm) (yes = 1) ? '%focusPos))
    if source == 1:
        cond = 0
    else:
        print 'Please turn on the source and place the camera on the focus
            point (%5.3f mm)'%focusPos

if bool(source):
    #Set exposure time
    cam.set_exposure(fX.determineUnsaturatedExposureTime(cam,img,[1,10000],1))
    #get centroid
    centroid = fX.acquirePSFCentroid(cam,img,initial_guess)
    print 'centroid at (%d, %d)' %(centroid[0],centroid[1])

```

```
62 #%%Acquire images at different camera position
63
64 acquire = 1
65 while bool(acquire):
66     cond = 1
67     while bool(cond):
68         dark = ''
69         winsound.Beep(freq, duration)
70         dark = int(raw_input('Is the source turned off (yes = 1) ? '))
71         if dark == 1:
72             cond = 0
73         else:
74             print 'Please shut down the source.'
75
76     winsound.Beep(freq, duration)
77     pos = float(raw_input('What is the position of the camera in mm focused
           (%5.3f mm) dephase 2Pi (pos+ = %5.3f mm, pos- = %5.3f) ?
           '%(focusPos,focusPos+3.19,focusPos-3.19)))
78
79     if bool(dark):
80         print 'Acquiring dark image...'
81         # Acquire dark images
82         [darkData,stdDarkData] = fX.acquireImg(cam,img,nbrImgAveraging)
83         print 'Cropping'
84         [darkdataCropped,stddarkDataCropped] =
               fX.cropAroundPSF(darkData,stdDarkData,centroid,sizeImg,sizeImg)
85         print 'saving'
86         fX.saveImg2Fits(datetime.datetime.today(),darkFolderPathCropped,nameCamera,darkdataCropped,
87         fX.saveImg2Fits(datetime.datetime.today(),darkFolderPathFull,nameCamera,darkData,stdDarkDat
88
89     #Acquire images -------------------------
90     cond = 1
91     while bool(cond):
92         source = ''
93         winsound.Beep(freq, duration)
94         source = int(raw_input('Is the source turned on (yes = 1) ? '))
95         if source == 1:
96             cond = 0
97         else:
98             print 'Please place turn on the camera'
99
100    if bool(source):
101        print 'Acquiring images...'
102        # Acquire focused images
103        for iImg in range(numberOfFinalImages):
104            imgNumber = iImg+1
105            print 'Acquiring Image %d'%imgNumber
106            [data,stdData] = fX.acquireImg(cam,img,nbrImgAveraging)
107            print 'Cropping'
108            [dataCropped,stdDataCropped] =
                   fX.cropAndCenterPSF(data-darkData,stdData+stdDarkData,sizeImg,initial_guess)
109            print 'Saving'
110            fX.saveImg2Fits(datetime.datetime.today(),folderPathCropped,nameCamera,dataCropped,stdD
111            fX.saveImg2Fits(datetime.datetime.today(),folderPathFull,nameCamera,data-darkData,stdDa
112
113    cond = 1
114    while bool(cond):
```

```
        acquire = ''
        winsound.Beep(freq, duration)
        acquire = int(raw_input('Do you want to acquire at an other camera
            position (yes = 1, no = 0) ? '))
        if acquire == 1:
            cond = 0
        elif acquire == 0:
            cond = 0
        else:
             print 'please answer with 0 or 1 for no or yes, respectively'


##Stop the acquisition
cam.stop_acquisition()
cam.close_device()

print 'Acquisition finished'
```

B.2.3 functionsXimea.py

```
import numpy as np
import pyfits
import os
import scipy.optimize as opt
#%% Functions
    ----------------------------------------------------------------------


#Create and save .fits from numpy array
def saveImg2Fits(date,folderPath,Detector,data,stdData,pos,nbrAveragingImg):

    #date : datetime at which the data where taken
    #folderPath : where to save the data$
    #Detector : name of detector (ex:Ximea)
    #data : np.array containing the image
    #stdData: np.array containing the error on each pixel
    #pos : the position of the camera on the sliding holder in mm

    imgHdu = pyfits.PrimaryHDU(data)
    stdHdu = pyfits.ImageHDU(stdData,name = 'imgStdData')
    hdulist = pyfits.HDUList([imgHdu,stdHdu])

    if not os.path.isdir(folderPath):
        os.makedirs(folderPath)

    hdulist.writeto(folderPath +
        date.strftime('%Y%m%d%H%M%S')+'_'+Detector+'_'+pos+'.fits')

def acquireImg(cam,img,nbrImgAveraging):
    imgData = np.zeros([1024,1280])
    stdData = np.zeros([1024,1280])
    for iImg in range(nbrImgAveraging) :
        #print iImg
        cam.get_image(img)
        imgTmpData = img.get_image_data_numpy()
```

```
34          imgData[:,:] += imgTmpData
35          stdData += imgTmpData*imgTmpData
36
37      stdData =
            np.sqrt((stdData-imgData*imgData/nbrImgAveraging)/(nbrImgAveraging-1))/(np.sqrt(nbrImgAvera
38      imgData = imgData/nbrImgAveraging
39
40      return [imgData,stdData]
41
42
43  def determineUnsaturatedExposureTime(cam,img,exposureLimit,precision):
44       exposureTimes = exposureLimit
45       if cam.get_acquisition_status() == 'XI_OFF':
46           cam.start_acquisition()
47
48       while np.absolute(np.diff(exposureTimes))>precision:
49           expTime2check = int(np.round(np.nanmean(exposureTimes)))
50           print 'Try expTime : %d [us]\n' %expTime2check
51           cam.set_exposure(expTime2check)
52           data = acquireImg(cam,img,10)[0]
53
54           if np.sum(data>250)>1:
55               exposureTimes[1] = int(np.ceil(np.nanmean(exposureTimes)))
56           else:
57               exposureTimes[0] = int(np.floor(np.nanmean(exposureTimes)))
58           print 'exposure time between %d and %d \n'
                %(exposureTimes[0],exposureTimes[1])
59
60       return int(np.floor(np.nanmean(exposureTimes)))
61
62  def TwoDGaussian((x, y), A, yo, xo, sigma_y, sigma_x):
63      g = A*np.exp( - ((x-xo)**2/(2*sigma_x**2) + ((y-yo)**2)/(2*sigma_y**2)))
64      return g.ravel()
65
66  def acquirePSFCentroid(cam,img,initial_guess):
67      #create the matrix grid of the detector CCD
68
69      data = acquireImg(cam,img,200)[0]
70      centroid = getPSFCentroid(data,initial_guess)
71      return centroid
72
73  def getPSFCentroid(data,initial_guess):
74
75      x = np.linspace(0,1280,1280)
76      y = np.linspace(0,1024,1024)
77      x, y = np.meshgrid(x, y)
78      print 'fitting'
79      popt,pcov = opt.curve_fit(TwoDGaussian, (x,y), data.ravel(), p0 =
            initial_guess)
80      print 'fitting done'
81      return [popt[2],popt[1]]
82
83
84  def cropAndCenterPSF(data,stdData,size,initial_guess):
85      Xextent = np.size(data,1)
86      Yextent = np.size(data,0)
87
```

```
88      centroid = getPSFCentroid(data,initial_guess)
89
90      minMarge =
            np.min([centroid[0],centroid[1],Xextent-centroid[0],Yextent-centroid[1]])
91
92      if minMarge>size/2:
93          return cropAroundPSF(data,stdData,centroid,size,size)
94      elif minMarge<size/2:
95          return cropAroundPSF(data,stdData, centroid,2*minMarge,2*minMarge)
96
97
98  def cropAroundPSF(data,stdData,centroid,sizeX,sizeY):
99
100     pxX =
            [int(np.floor(centroid[0])-np.ceil(sizeX/2)),int(np.floor(centroid[0])+np.ceil(sizeX/2
101     pxY =
            [int(np.floor(centroid[1])-np.ceil(sizeY/2)),int(np.floor(centroid[1])+np.ceil(sizeY/2
102
103     dataCropped = data[pxY[0]:pxY[1],pxX[0]:pxX[1]]
104
105     stdDataCropped = stdData[pxY[0]:pxY[1],pxX[0]:pxX[1]]
106
107     return [dataCropped,stdDataCropped]
```

Appendix C

IDL Code

C.1 Shack-Hartmann Acquisition Code

C.1.1 readAndAverageSHdata.pro

```
function readAndAverageSHdata, folderPath

fileExt='*.csv'

files = file_search(folderPath+fileExt)

r = readshwfsdata(files[0])

NFiles = n_elements(files)

for ifile = 1,Nfiles-1 do begin

  rtmp = readshwfsdata(files[ifile])

  r.wavefront = r.wavefront+rtmp.wavefront
  r.zernike[3,*] = r.zernike[3,*]+rtmp.zernike[3,*]

endfor

r.wavefront = r.wavefront / NFiles
r.zernike[3,*] = r.zernike[3,*] / Nfiles

return, r
end
```

C.1.2 readSHWFSdata.pro

```
function readSHWFSdata, filePath

openr, f, filePath, /GET_LUN

iLine = 0
line = ''


coefficient = []
index = []
order = []
```

```
frequency = []
wavefront = []

while ~EOF(f) do begin
  readf, f, line
  iLine += 1

  ;get the zernike coefficient
  if strmatch(line,'* ZERNIKE FIT *') then begin
    subheaderNbrLines = 5
    for isHd =1,subheaderNbrLines do begin
      readf, f, line
      iLine += 1
    endfor

    readf, f, line
    iLine += 1
    sLine = strsplit(line,',',/EXTRACT)

    while stregex(sLine[0],'[0-9]+') ne -1 and ~EOF(f) do begin
      index = [[index],[long(sLine[0])]]
      order = [[order],[long(sLine[1])]]
      frequency = [[frequency],[long(sLine[2])]]
      coefficient = [[coefficient],[double(sLine[3])]]
      readf, f, line
      iLine += 1
      sLine = strsplit(line,',',/EXTRACT)
    endwhile
  endif

  if strmatch(line,'\*\*\* WAVEFRONT \*\*\*') then begin
    subheaderNbrLines = 11
    for isHd =1,subheaderNbrLines do begin
      readf, f, line
      iLine += 1
    endfor
    readf, f, line
    iLine += 1
    sLine = strsplit(line,',',/EXTRACT)
    nel = n_elements(sLine)
    while stregex(sLine[0],'[0-9]+') ne -1 and ~EOF(f) do begin
      wavefront = [[wavefront],[double(sLine[1:nel-1])]]
      readf, f, line
      iLine += 1
      sLine = strsplit(line,',',/EXTRACT)
    endwhile

  endif
endwhile
free_lun, f

zernike = [index,order,frequency,coefficient]

return, {zernike:zernike,wavefront:wavefront}

end
```

Bibliography

Blanc, Amandine (2002). "Identification de réponse impulsionnelle et restauration d'images: apports de la diversité de phase". PhD thesis. Université Paris XI UFR Scientifique d'Orsay.

Bouxin, A. (2017). "Phasor diversity to measure the static aberrations of an optical system". MA thesis. HEIG-VD.

F. Zernike, von (1934). "Beugungstheorie des schneidenver-fahrens und seiner verbesserten form, der phasenkontrastmethode". In: *Physica* 1.7, pp. 689 –704. ISSN: 0031-8914. DOI: https://doi.org/10.1016/S0031-8914(34)80259-5. URL: http://www.sciencedirect.com/science/article/pii/S0031891434802595.

Fontanella, J C (1985). "Wavefront sensing deconvolution and adaptive optics". In: *Journal of Optics* 16.6, p. 257. URL: http://stacks.iop.org/0150-536X/16/i=6/a=002.

Gerchberg, R. W. and W. O. Saxton (1972). "A practical algorithm for the determination of phase from image and diffraction plane pictures". In: *Optik* 35.2, 237–250.

Gonsalves, R. A. (1976). "Phase retrieval from modulus data". In: *J. Opt. Soc. Am.* 66.9, pp. 961–964. DOI: 10.1364/JOSA.66.000961. URL: http://www.osapublishing.org/abstract.cfm?URI=josa-66-9-961.

Gonsalves, Robert A. (1982). "Phase Retrieval And Diversity In Adaptive Optics". In: *Optical Engineering* 21, pp. 21 –21 –4. DOI: 10.1117/12.7972989. URL: http://dx.doi.org/10.1117/12.7972989.

Goodman, Joseph W. (1988). *Introduction to Fourier Optics*. Ed. by L. Cox and John M. Morris. 2nd. McGraw-Hill Companies, Inc.

Hartmann (1900). "Bemerkungen über den Bau und die Justirung von Spektrographen". In: *Z. Instrumentenkd* 20, p. 47.

Kolmogorov, A N (1968). "Local Strucure of Turbulence in an Incompressible Viscous Fluid at Very High Reynolds numbers". In: *Soviet Physics Uspekhi* 10.6, p. 734. URL: http://stacks.iop.org/0038-5670/10/i=6/a=R02.

Mugnier, Laurent M., Amandine Blanc, and Jérôme Idier (2006). "Phase Diversity: A Technique for Wave-Front Sensing and for Diffraction-Limited Imaging". In: ed. by Peter Hawkes. Vol. 141. Advances in Imaging and Electron Physics. Elsevier, pp. 1 –76. DOI: https://doi.org/10.1016/S1076-5670(05)41001-0. URL: http://www.sciencedirect.com/science/article/pii/S1076567005410010.

Noll, Robert J. (1976). "Zernike Polynomials and Atmospheric Turbulence". In: *J. Opt. Soc. Am.* 66.3, pp. 207–211. DOI: 10.1364/JOSA.66.000207. URL: http://www.osapublishing.org/abstract.cfm?URI=josa-66-3-207.

Obukhov, A. M. (1949). "Structure of the temperature field in turbulent flow". In: *Ser. Geograf. Geofiz.* 13.1, pp. 58–69.

Paxman, Richard G., Timothy J. Schulz, and James R. Fienup (1992). "Joint estimation of object and aberrations by using phase diversity". In: *J. Opt. Soc. Am. A* 9.7, pp. 1072–1085. DOI: 10.1364/JOSAA.9.001072. URL: http://josaa.osa.org/abstract.cfm?URI=josaa-9-7-1072.

Ragazzoni, Roberto (1996). "Pupil plane wavefront sensing with an oscillating prism". In: *Journal of modern Optics* 43.2, pp. 289–293.

Roddier, François (1988). "Curvature sensing and compensation: a new concept in adaptive optics". In: *Appl. Opt.* 27, pp. 1223–1225.

Shack, R. V. and B. C. Platt (1971). "Production and use of a lenticular Hartmann screen". In: *Spring Meeting of the Optical Society of America*. Vol. 61, 656–660.

Thorlabs (2017). *Principles of Spatial Filters*. Thorlabs. URL: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=1400.

— (2018). *Shack-Hartmann Wavefront Sensor*. Thorlabs. URL: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=2946.

V.I. Tatarski, Richard A. Silverman (1961). *Wave Propagation in a Turbulent Medium*. Ed. by McGraw-Hill. McGraw-Hill.

Wikipedia (2018). *Zernike polynomials*. Wikipedia. URL: https://en.wikipedia.org/wiki/Zernike_polynomials.