

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

Contents

Declaration of Authorship	iii
Acknowledgements	vii
Introduction	1
1 Theoretical background	3
1.1 Scalar Diffraction Theory	3
1.1.1 Scalar Field and Helmholtz equation	3
1.1.2 Rayleigh-Sommerfeld integral	3
1.1.3 Fresnel approximation	4
1.1.4 Fraunhofer approximation	5
1.1.5 Converging lens introduction	5
1.2 Imaging system	6
1.2.1 Impulse Response (IR)	6
1.2.2 Optical Transfer Function (OTF)	6
1.2.3 From Object to Image	7
1.3 Aberrations	9
1.3.1 Sources of aberration	10
1.3.2 Zernike polynomials	10
1.4 Phase retrieval	12
1.4.1 Shack-Hartmann	12
2 Phase Diversity	15
2.1 Principle	15
2.2 ONERA algorithm	16
2.2.1 Algorithm description	16
2.2.2 Implementation	17
2.3 Analytical algorithm	17
2.3.1 Algorithm description	17
2.3.2 Implementation	20
3 Phase Diversity Experiment	23
3.1 Experimental Setup	23
3.1.1 Light source	23
3.1.2 Entrance pupil	25
3.1.3 Pupil imaging system	25
3.1.4 Detectors	25
3.2 Data Acquisition	25
3.2.1 Ximea Camera	25
3.2.2 Shack-Hartmann WFS	27
3.3 Results	27
3.3.1 ONERA Phase Diversity test	28

3.3.2 Parallel plane plate	33
A Optical Component Datasheets	35
A.1 Pigtailed laser diode	36
A.1.1 Power supply modification	37
A.2 Converging lens A220TM-A, f = 11 mm	37
A.3 Pinhole 10 μm	38
A.4 Converging lens AL100200, f = 200 mm	39
A.5 Converging lens AC254-100-A, f = 100 mm	40
A.6 Ximea Camera, MQ013MG-E2	41
A.7 Shack-Hartmann wavefront sensor, WFS150-5C	42
B Python Code	43
B.1 Phase Diversity analytical algorithm code	43
B.1.1 phaseDiversity3PSFs.py	43
B.1.2 fs.py	45
B.1.3 myExceptions.py	48
B.1.4 zernike.py	48
B.1.5 phasor.py	49
B.2 Acquisition Code: Ximea Camera	50
B.2.1 AlignementScriptXimeaCamera.py	50
B.2.2 AcquisAndSaveXimea.py	53
B.2.3 functionsXimea.py	56
C IDL Code	59
C.1 Shack-Hartmann Acquisition Code	59
C.1.1 readAndAverageSHdata.pro	59
C.1.2 readSHWFSdata.pro	59
Bibliography	61

List of Figures

1.1	Diffraction Schemas	4
1.2	Schema of a imaging instrument, (Goodman, 1988, Chapter 6.1)	6
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.	7
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	8
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	9
1.5	Gaussian reference sphere vs. aberrated Wavefront	9
1.7	Representation of the 21 st Zernike polynomials	11
1.8	Shack-Hartmann principle	13
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.	16
3.2	Wavefront curvature	23
3.1	Experimental Setup Schema	24
3.3	Source schema and pinhole effect on the beam.	24
3.4	Example of the results of an alignment procedure	26
3.5	Shack-Hartmann Software GUI, with the beam view on the top left, the spot field in the middle and the reconstructed wavefront on the top right.	27
3.6	Noise level as a function of the number of averaging images acquired with $\sim 300 \mu\text{s}$ 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.	28
3.7	Results of the phase retrievals computed with the 25 different noise levels. The phase retrievals are done with a $j_{\max} = 30$	29
3.8	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.	30
3.9	Zernike coefficients a_j as a function of j for the 227 retrievals done for 231 different j_{\max} from 4 to 231.	31
3.10	Results of the Δz measurement error test. The plots represent the results of the 125 phase retrievals run with all the permutations of errors, $[-2\sigma, -1\sigma, 0, 1\sigma, 2\sigma]$ with $\sigma = 5\text{e}-3 \text{ mm}$, on the three PSFs position Δz	32

List of Tables

3.1 Optical Components	25
----------------------------------	----

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 :

$$u(P, t) = A(P) \exp [-j2\pi\nu t - j\phi(P)], \quad (1.1)$$

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,

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

must verify the Helmotz equation :

$$(\nabla^2 + k^2)U = 0, \quad (1.3)$$

where k is the wave number given by

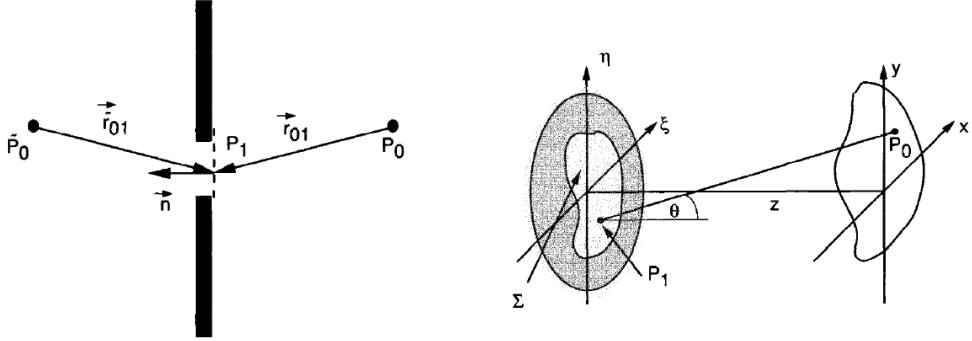
$$k = 2\pi n \frac{\nu}{c} = \frac{2\pi}{\lambda}, \quad (1.4)$$

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 \tilde{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).

$$U(P_0) = \frac{1}{j\lambda} \iint_{\Sigma} U'(P_1) \frac{\exp(jkr_{01})}{r_{01}} \cos(\mathbf{n}, \mathbf{r}_{01}) d\mathbf{s} \quad (1.5)$$



(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 $\mathbf{r}_{01} = \mathbf{P}_0\mathbf{P}_1$ given by

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

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,

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

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 :

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

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,

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

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

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

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

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,

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

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

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

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,

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

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,

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

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,

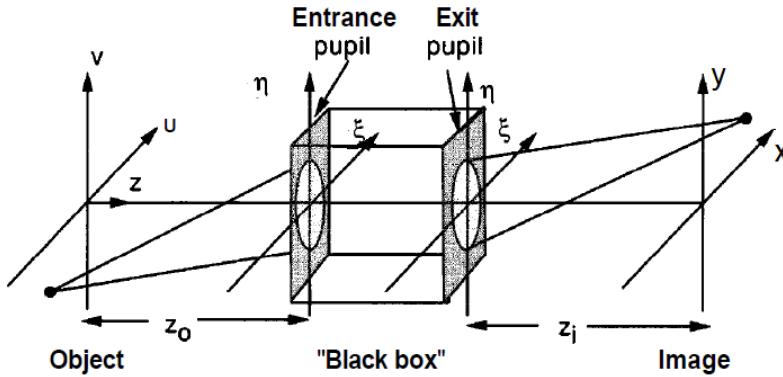


FIGURE 1.2: Schema of an imaging instrument, (Goodman, 1988, Chapter 6.1)

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

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,

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

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

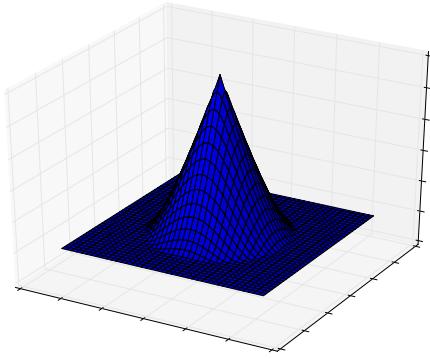


FIGURE 1.3: OTF of a perfect imaging system composed by a 3.6 mm pupil and a focal length of 80 mm at a wavelength of 637.5 nm.

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,

$$\tilde{h}_{optical}(\xi, \eta) = \mathcal{F}\{h_{optical}(x, y)\} = (P \otimes P)(\xi, \eta), \quad (1.17)$$

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,

$$i(x, y) = |U(x, y)|^2 = \left| \iint_{-\infty}^{\infty} h(x, y; u, v) U(u, v) du dv \right|^2. \quad (1.18)$$

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,

$$i(x, y) = \left| \iint_{-\infty}^{\infty} h(x - \tilde{u}, y - \tilde{v}) U(\tilde{u}, \tilde{v}) d\tilde{u} d\tilde{v} \right|^2, \quad (1.19)$$

where ($\tilde{u} = Mu$, $\tilde{v} = Mv$) 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,

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

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

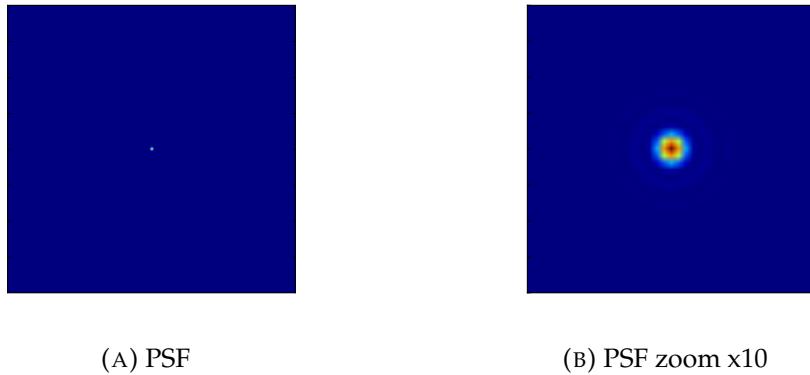


FIGURE 1.4: PSF of a perfect imaging system composed by a 3.6 mm pupil and a focal length of 80 mm at a wavelength of 637.5 nm. The size, N , of the PSF is 400 and the pixel size is $5.3 \mu m$.

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,

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

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,

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

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.

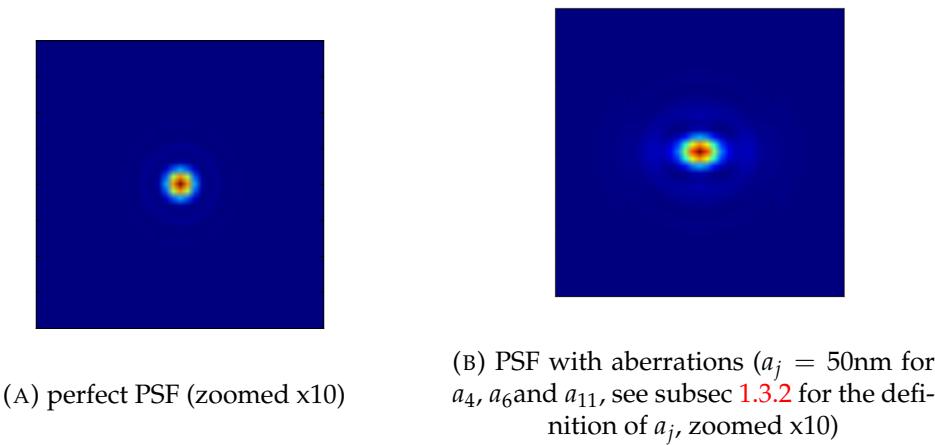


FIGURE 1.6: Comparison of perfect PSF and PSF with aberrations of an imaging system composed by a 3.6 mm pupil and a focal length of 80 mm at a wavelength of 637.5 nm. The size, N , of the PSF is 400 and the pixel size is $5.3 \mu\text{m}$.

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,

$$SR = \frac{\int \tilde{h}_{\text{optical}}(\xi, \eta) d\xi d\eta}{\int \tilde{h}_{\text{perfect}}(\xi, \eta) d\xi d\eta}, \quad (1.23)$$

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

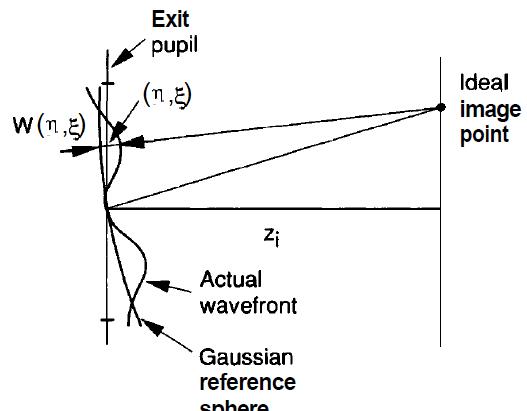


FIGURE 1.5: Gaussian reference sphere vs. aberrated Wavefront, (Goodman, 1988, Chapter 6.4). The dephasing is equal to the wave number multiplied by the aberration function, $\phi_{Ab}(\xi, \eta) = kW(\xi, \eta)$.

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).

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

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 $|m| \leq n$, $n - |m| = \text{even}$.

The trigonometric function is given by,

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

and the radial function is given by,

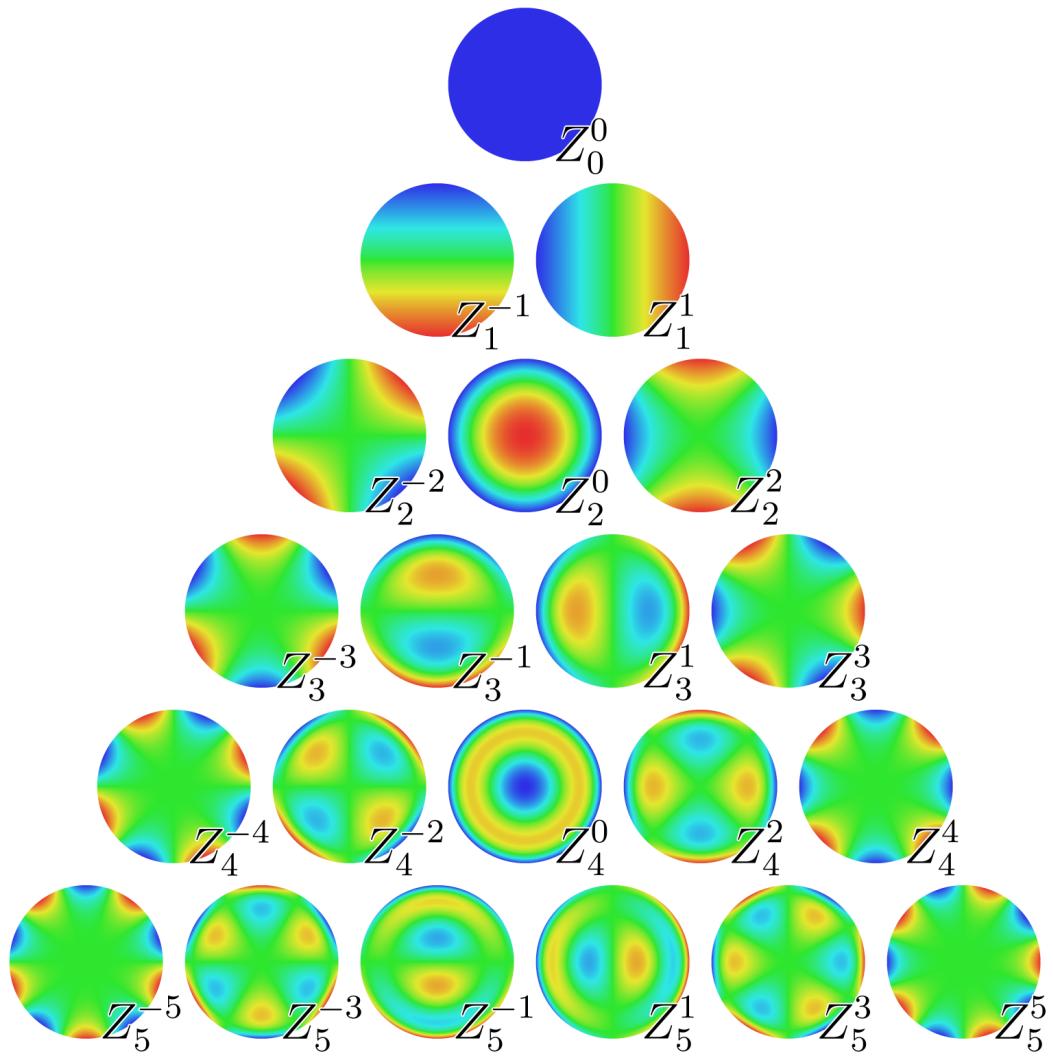


FIGURE 1.7: Representation of the 21st Zernike polynomials (Wikipe-
dia, 2018)

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

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

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

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,

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

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),

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

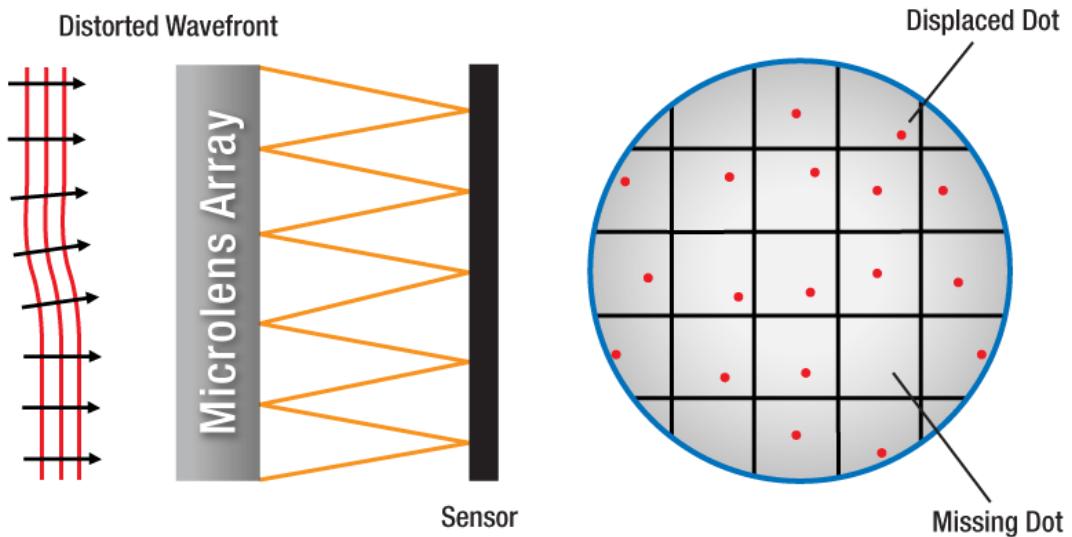


FIGURE 1.8: Shack-Hartmann principle (Thorlabs, 2018)

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 does 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 systems 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 developed by Dr. Laurent Jolissaint and refine by Jordan Voirin.

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,

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

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

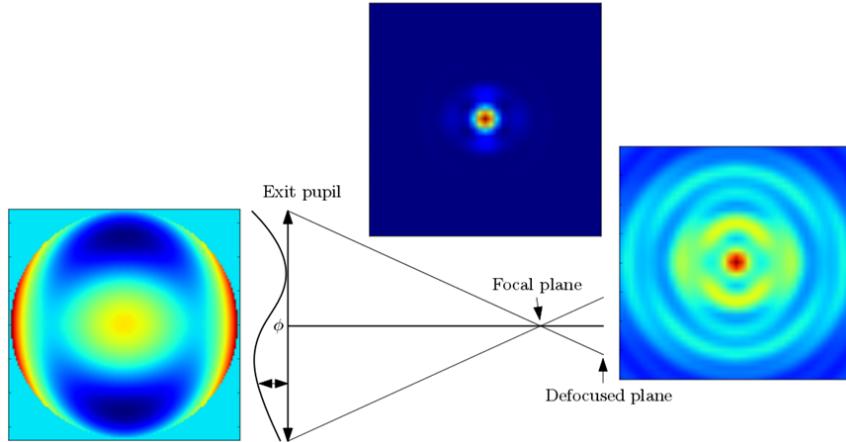


FIGURE 2.1: Schema of the phase diversity principle. The images from left to right are : the phase arriving on the exit pupil, the focused image and the defocused (2π) image.

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),

$$\mathbf{i}_f = \mathbf{h}_f * \mathbf{o} + \mathbf{n}_f \quad (2.2)$$

$$\mathbf{i}_d = \mathbf{h}_d * \mathbf{o} + \mathbf{n}_d, \quad (2.3)$$

where the bold letters means that it is the sampled quantities, the indexes f and d means respectively focused and defocused and \mathbf{o} 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),

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

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,

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

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 approach, developed by Dr. Laurent Jolissaint, 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,

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

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,

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

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

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

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,

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

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

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

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,

$$\phi = \psi + \gamma \quad (2.11)$$

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,

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

We can decompose ΔPSF into its even and odd components,

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

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

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,

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

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,

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

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

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,

$$\psi = \sum_{j \text{ is even}} a_j Z_j \quad (2.18)$$

$$\gamma = \sum_{j \text{ is odd}} a_j Z_j. \quad (2.19)$$

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 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,

$$\delta\phi_+ = \delta\phi \text{ and } \delta\phi_- = -\delta\phi \quad (2.20)$$

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,

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

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

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,

$$\Delta PSF_{odd} = \sum_{j \text{ is odd}} a_j 2\Im\{\widetilde{P}^*\widetilde{P}\widetilde{Z}_j\} \quad (2.23)$$

$$\Delta PSF_{\delta\phi_+,even} - \Delta PSF_{\delta\phi_-,even} = \sum_{j \text{ is even}} a_j \{-4\widetilde{PC}^*\widetilde{S}\widetilde{Z}_j + 4\widetilde{PS}^*\widetilde{C}\widetilde{Z}_j\}. \quad (2.24)$$

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,

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

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

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

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

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 $j_{min} < j < j_{max}$ 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 `lstsq` function¹. This functions takes a vector \vec{b} corresponding to the flattened ΔPSF_{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} .

¹Documentation 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.

In order to obtain such a planar wavefront at the entrance pupil, the light source consist of a "pigtailed laser diode", a $f=11\text{mm}$ converging lens, a pinhole and a $f=200\text{ mm}$ 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 $10\mu\text{m}$ 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.

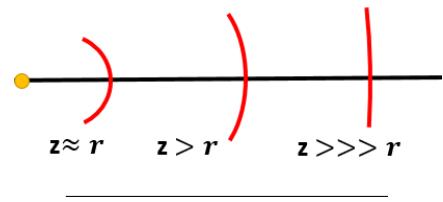


FIGURE 3.2: Wavefront curvature for different point source's distances, z . r represents the characteristic size of the arc of interest.

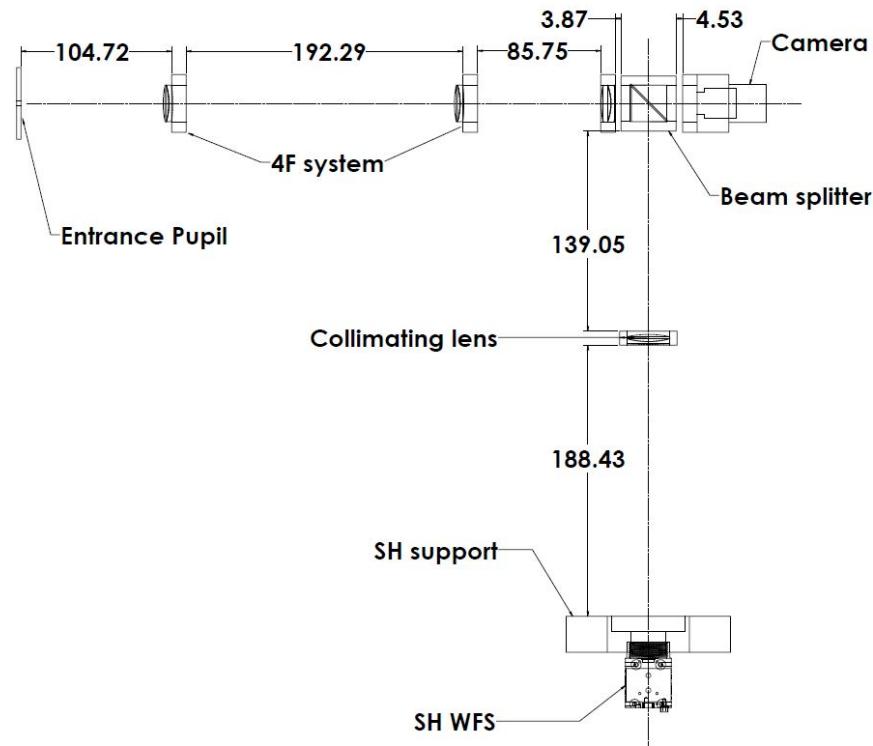


FIGURE 3.1: Experimental setup schema with the relevant distances, (Bouxin, 2017).

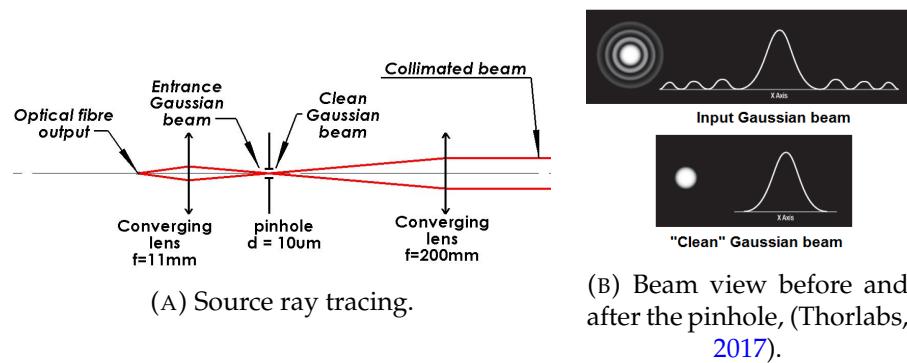


FIGURE 3.3: Source schema and pinhole effect on the beam.

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, $10 \mu\text{m}$	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 has 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 consists 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

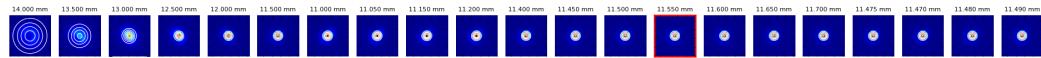
The image of the entrance pupil, obtained with the 4F system, is focused onto a CMOS Ximea camera by a $f = 80$ mm converging lens to acquire the PSFs for the phase diversity wavefront retrieval. The camera has a surface composed by 1280x1024 pixels of $5.3 \mu\text{m}$, see Appendix A.6. It is mounted 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 $4.65 \mu\text{m}$, see Appendix A.7.

3.2 Data Acquisition

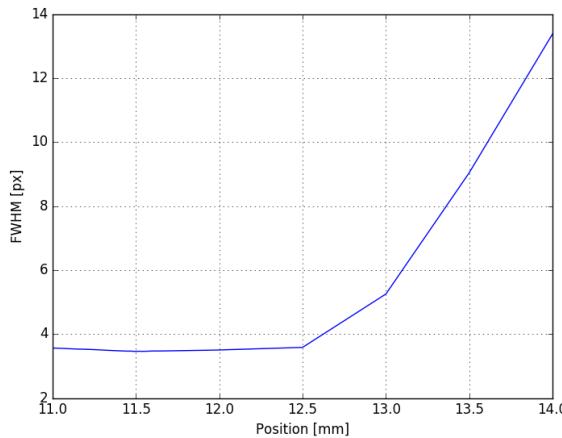
3.2.1 Ximea Camera

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, pyXimea¹, 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 `AlignmentScriptXimeaCamera.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.

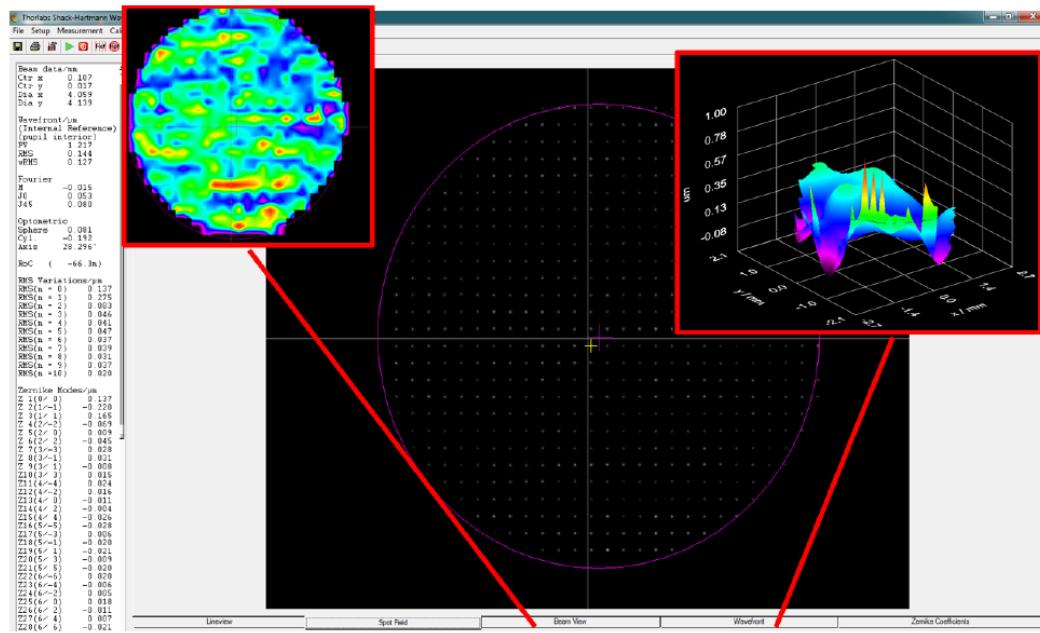


FIGURE 3.5: Shack-Hartmann Software GUI, with the beam view on the top left, the spot field in the middle and the reconstructed wavefront on the top right.

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 is delivered with a software which does the numerical integration to compute the wavefront. The GUI shows the spot field (focal points), the beam view (irradiance of the CCD), the wavefront measured or reconstructed (where you can choose the Zernike coefficients to consider for the integration) and the Zernike coefficients.

The 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

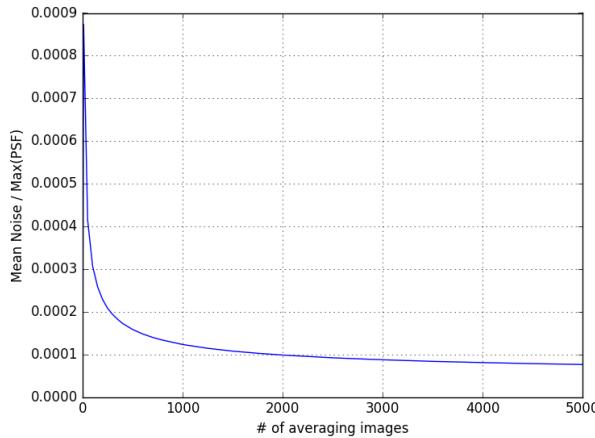


FIGURE 3.6: Noise level as a function of the number of averaging images acquired with $\sim 300 \mu\text{s}$ exposition time. The noise level is computed as the mean of the standard deviation of every pixel divided by the maximum of the focused PSF.

the phase screen and compare the results to the Shack-Hartmann wavefront sensor results.

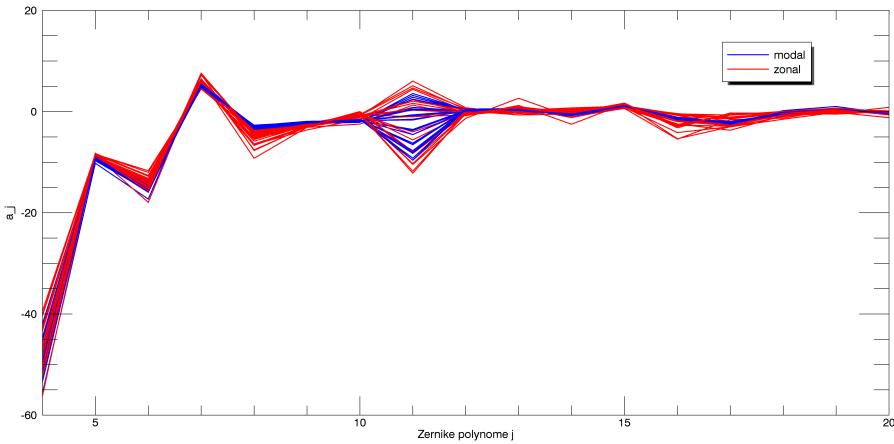
3.3.1 ONERA Phase Diversity test

The motivation of this test is 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 position of the Ximea camera.

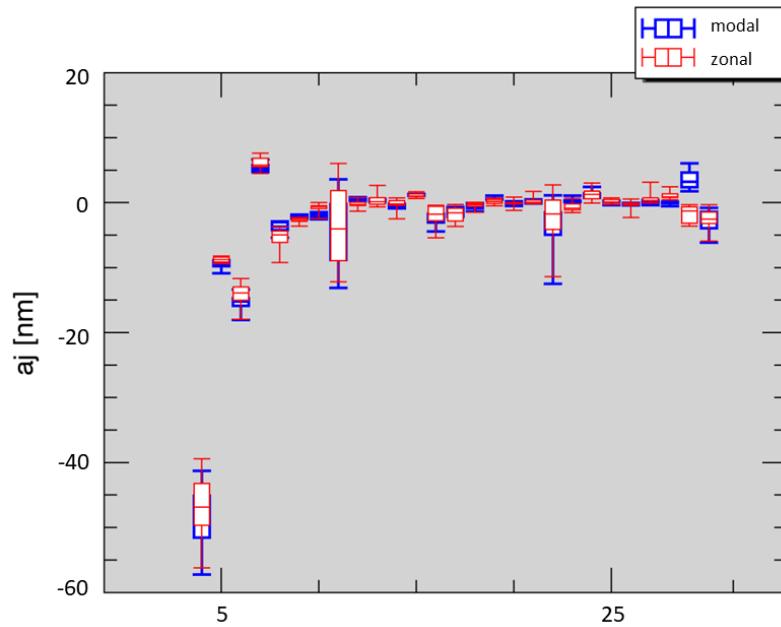
To study the noise, we acquire PSFs with 25 different numbers of averaging images going from 10 to 5000. The noise level is computed as the ratio between the mean pixel standard deviation and the PSF maximum. The Ximea detector has a noise curve visible in Figure 3.6. The noise curve is computed with 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 3.7a and 3.7b 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 3.7b. 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 3.8, 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



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

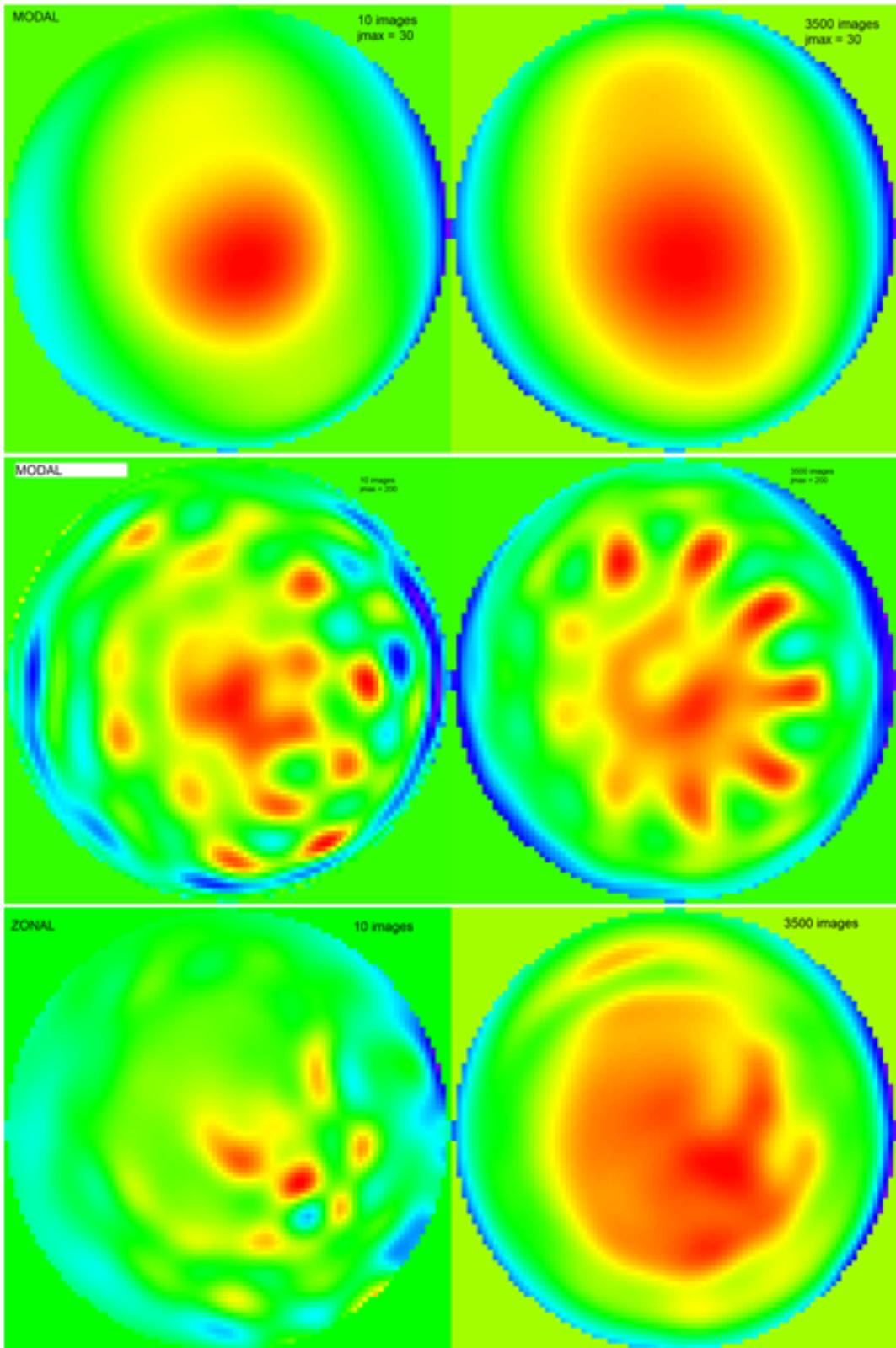


FIGURE 3.8: Reconstructed wavefronts for two different number of averaging images, 10 and 3500, left and right column respectively. The two first line are modal retrieval with $j_{max} = 30$ and $j_{max} = 200$ and the last line is the zonal retrieval.

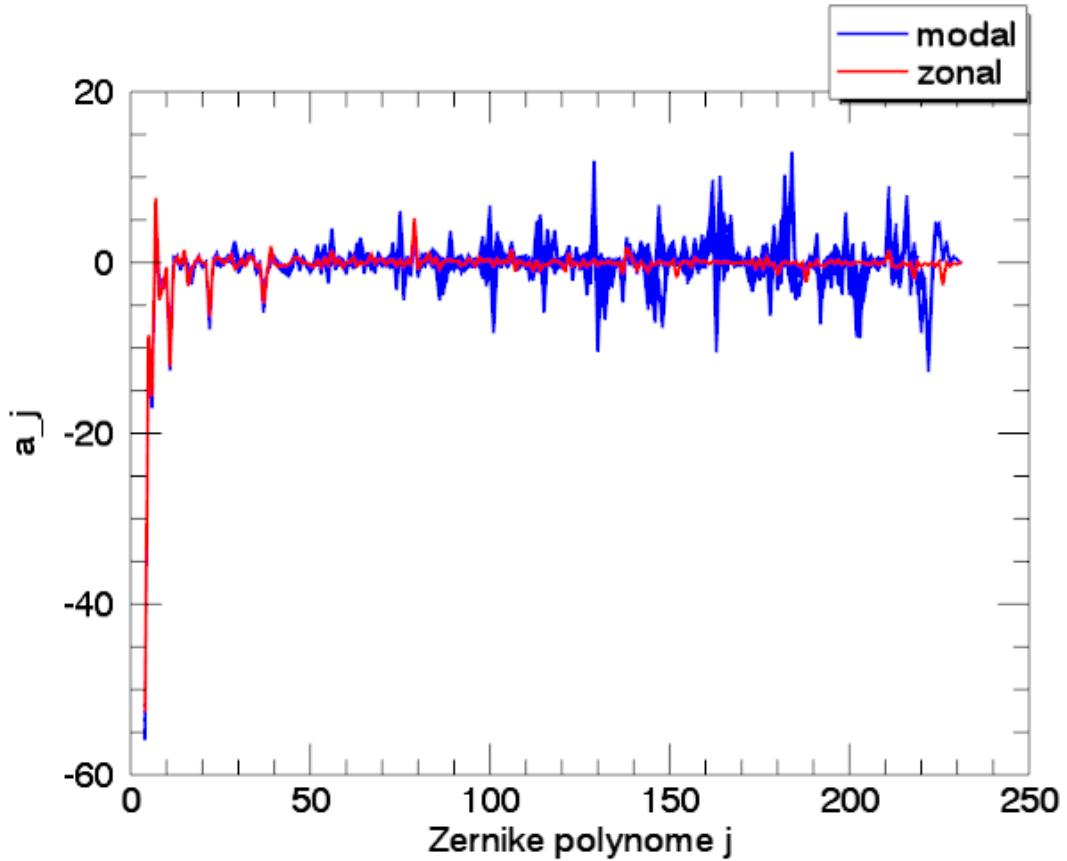


FIGURE 3.9: Zernike coefficients a_j as a function of j for the 227 retrievals done for 231 different j_{max} from 4 to 231.

retrieved structures are similar, however for $j_{max} = 200$ there are more structures in the retrieval done on the PSFs with 10 averaging images than with 3500 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. Looking at the 25 reconstructed wavefront with the zonal method, the noise threshold is at 3500 averaging images to kill the effect of the noise. Also as said above for the Zernike coefficients and still valid for the wavefront, the two retrieval method gives similar results, as one can compare the three retrievals with 3500 images on the left column of Figure 3.8. There is structures present on the wavefront reconstructed on 200 Zernike coefficients, but the footprint of the beam is similar to the two other reconstructions.

In order to be sure that the number of Zernike coefficient retrieved only influenced the wavefront reconstruction and not the a_j 's values, we computed them varying j_{max} from 4 to 231 and found that the spread is negligible, see Figure 3.9. This confirmation is expected since the Zernike polynomials are orthonormal, thus not correlated.

Finally, the last test is to control the impact of a measurement error $\sigma_{\Delta z}$, the error position of the Ximea camera, on the phase retrieval. To do so the latter is run with 125 different permutations of the errors $[-2\sigma, -1\sigma, 0, 1\sigma, 2\sigma]$ on the 3 PSFs position. σ is set to represent the best precision we have on the position of the camera. As

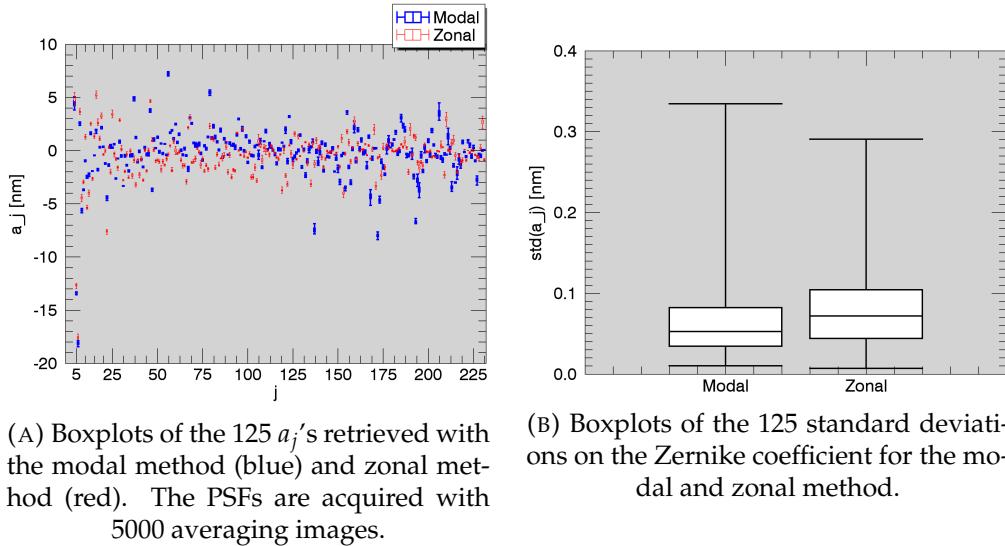


FIGURE 3.10: Results of the Δz measurement error test. The plots represent the results of the 125 phase retrievals run with all the permutations of errors, $[-2\sigma, -1\sigma, 0, 1\sigma, 2\sigma]$ with $\sigma = 5e-3$ mm, on the three PSFs position Δz

explained in section 3.1.4, the camera is mounted on a sliding element moved by a micro metric screw, so $\sigma = 5e-3$ mm. Figures 3.10a and 3.10b present the results. The position error has nearly no effect on the phase retrieval, the median standard deviations are ~ 0.05 nm and ~ 0.07 nm for the modal and zonal method respectively.

In conclusion, the ONERA algorithm tests shows that first the two different methods, to retrieve the phase using the JMAP estimator, modal and zonal give similar results. Then, we have seen that the noise levels present in the PSFs influence the retrieval, see Figures 3.7a, 3.7b and 3.8. We can consider the noise as an added aberration that touches all the Zernike coefficients. The spherical aberration is the most sensitive one. Also looking at the wavefronts, we see that there is a number of averaging images threshold at 3500 images for the zonal retrieval. Above this threshold the noise seems to be smooth and the retrieval loses the high spatial frequency components. The same thing is true for the modal retrieval, but the smoothing is not as effective at 3500 images. For the rest of this work each PSFs will be acquired with 5000 averaging images. Also we showed that the number of Zernike coefficient j_{max} does not alter the modal retrieval. And finally, the error on the position of the camera also have a negligible impact.

3.3.2 Parallel plane plate

Now that we know how the algorithm behaves with respect to noise, j_{max} , etc... We compare its results to a calibrated aberration. The calibration is done with simulation using Zemax.

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		LPS-635-FC Specifications		
		Min	Typ.	Max
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			

*Temperature = 25 °C

Drawing

The drawing includes three views:

- Pigtail Bottom View:** Shows the diode leads with dimensions: 0.38", 0.75", 0.30", 0.59", Ø1.00", and a clearance hole for #2 hardware at two places.
- Side View:** Shows the laser/lens housing with a strain relief boot, a fiber 1 m of SM600, and dimensions 1.28" and 1.09".
- Diode Bottom View:** Shows the diode chip with a 2.5 mm diameter and lead angles of 90°.

 A notice icon and text provide ESD handling instructions: "To avoid equipment damage from static discharge: Wear ESD wriststrap when handling this device."
 The pin diagram shows PD (Photodiode) and LD (Laser Diode) terminals, and PIN CODE A.

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

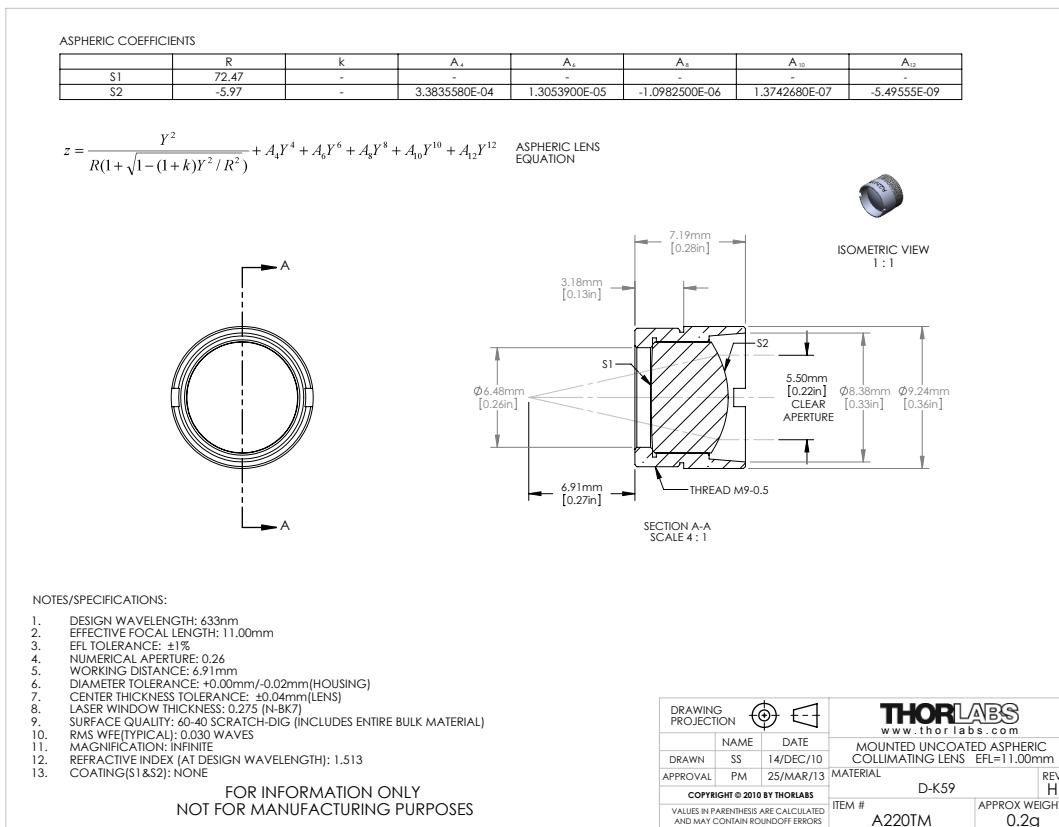
June 6, 2013
 5175-S01, Rev B

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.

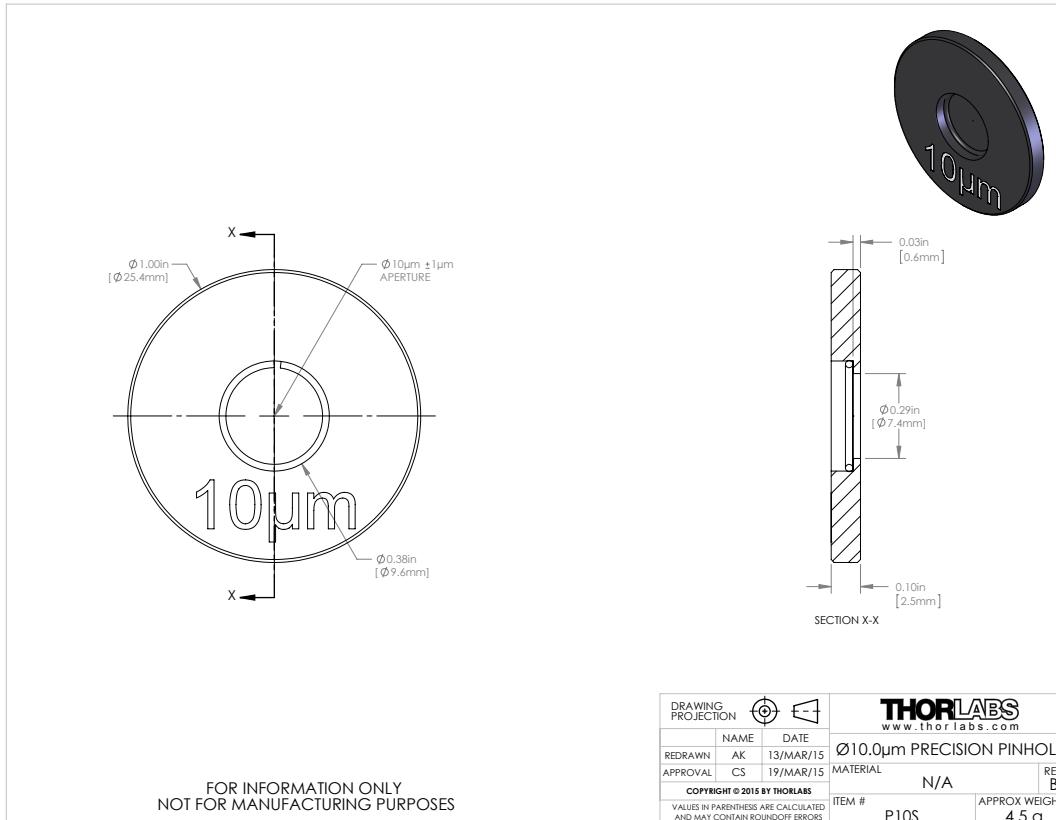
!!! mettre la photo du driver et de la modif !!!

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



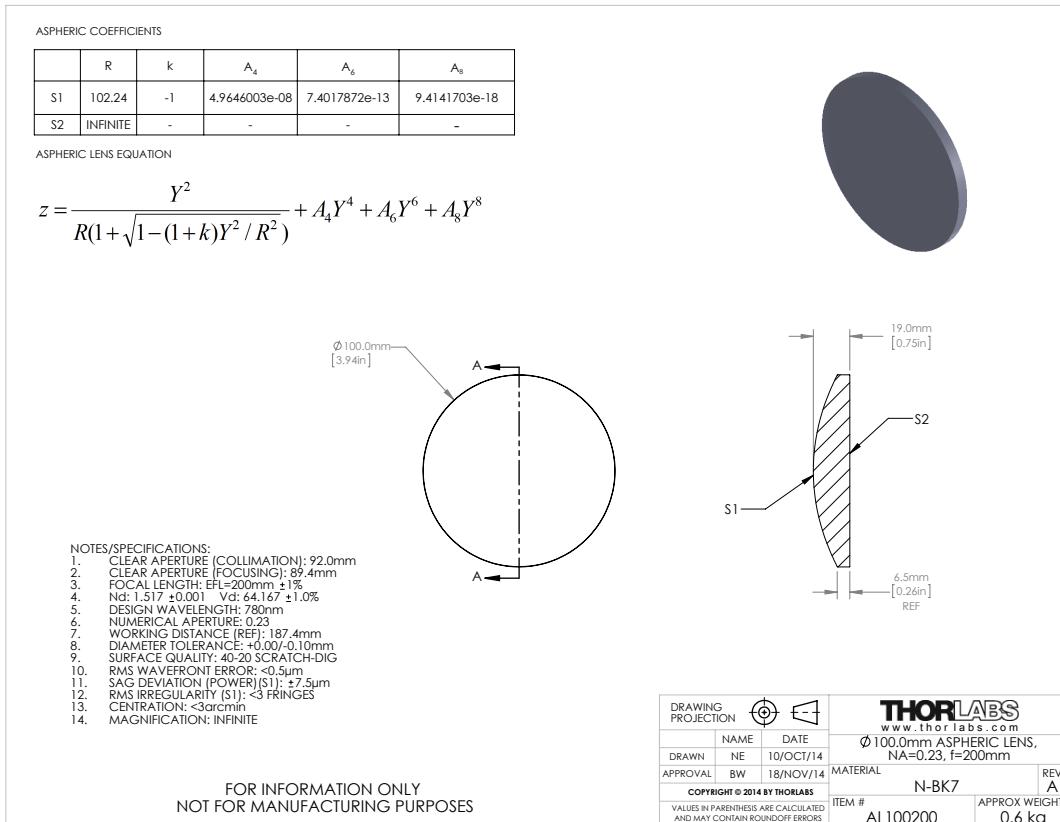
Source : www.thorlabs.com

A.3 Pinhole 10 μm



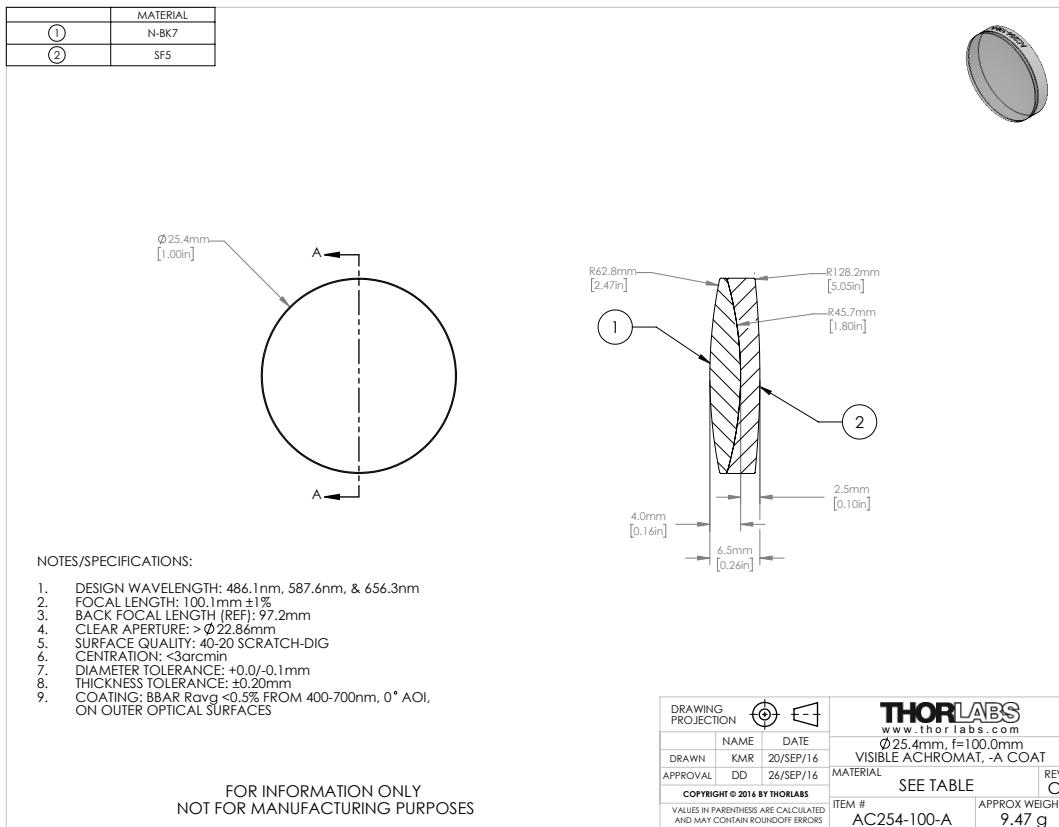
Source : www.thorlabs.com

A.4 Converging lens AL100200, f = 200 mm



Source : www.thorlabs.com

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



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 Appendix

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⁾	$\lambda/15$ rms @ 633 nm		$\lambda/50$ rms @ 633 nm
Wavefront Sensitivity ²⁾	$\lambda/50$ rms @ 633 nm		$\lambda/150$ 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)

Appendix B

Python Code

B.1 Phase Diversity analytical algorithm code

B.1.1 phaseDiversity3PSFs.py

```

1 #Class phaseDiversity object to retrieve the phase in the pupil from two
  psfs in/out of focus
2
3 import numpy as np
4 import fs
5 import myExceptions
6 import PSF as psf
7
8 class phaseDiversity3PSFs(object):
9
10    def
11        __init__(self,inFoc,outFocpos,outFocneg,deltaZ,lbda,pxsize,F,pupilRadius,jmin,jmax):
12    #
13    #      input:
14    #          inFoc,outFocpos,outFocneg are the 3 squared PSFs data, (focused,
15    #          defocused positiv and defocused negative)
16    #          deltaZ is the displacement of the detector to acquire the two
17    #          defocused PSFs
18    #          lbda is the wavelength of the incoming light
19    #          pxsize is the pixel size of the detector
20    #          F is the focal length of the imaging system
21    #          pupilRadius is the radius of the exit pupil
22    #          jmin and jmax gives the boundary on the js to retrieve
23
24        print 'phaseDiversity ...'
25
26    #PSF
27        self.inFoc = inFoc
28        self.outFocpos = outFocpos
29        self.outFocneg = outFocneg
30        shapeinFoc = np.shape(self.inFoc)
31        shapeoutFocpos = np.shape(self.outFocpos)
32        shapeoutFocneg = np.shape(self.outFocneg)
33        if shapeinFoc == shapeoutFocpos and shapeinFoc == shapeoutFocneg:
34            self.shape = shapeinFoc
35        else:
36            raise myExceptions.PSFsizeError('the shape of the in/out PSFs is
37                                         not the same',[shapeinFoc,shapeoutFocpos])
38        if shapeinFoc[0]==shapeinFoc[1] and np.mod(shapeinFoc[0],2)==0:
39            self.N = shapeinFoc[0]

```

```

36     else:
37         raise myExceptions.PSFsizeError('Either PSF is not square or
38                                         mod(N,2) != 0',shapeinFoc)
39
40     # properties
41     self.deltaZ = deltaZ
42     self.lbda = lbda
43     self.pxsize = pxsize
44     self.F = F
45     self.pupilRadius = pupilRadius
46     self.dxp = self.F*self.lbda/(self.N*self.pxsize)
47     self.rad = int(np.ceil(self.pupilRadius/self.dxp))
48     if 2*self.rad > self.N/2.:
49         raise myExceptions.PupilSizeError('Npupil (2*rad) is bigger than
50                                         N/2 which is not correct for the fft computation',[])
51
52     self.NyquistCriterion()
53     self.jmin = jmin
54     self.jmax = jmax
55     self.oddjs = fs.getOddJs(self.jmin,self.jmax)
56     self.evenjs = fs.getEvenJs(self.jmin,self.jmax)
57
58     # result computation
59     self.result = self.retrievePhase()
60
61     def NyquistCriterion(self):
62         deltaXfnyq = 0.5*self.lbda/(2*self.pupilRadius)
63         deltaXf = self.pxsize/self.F
64         if deltaXfnyq < deltaXf : raise myExceptions.NyquistError('the
65                                         system properties do not respect the nyquist criterion',[])
66
67     def retrievePhase(self):
68         y1,A1 = self.initiateMatrix1()
69         results1 = np.linalg.lstsq(A1,y1)
70         ajsodd = results1[0]
71         A1tA1inv= np.linalg.inv(np.matmul(np.transpose(A1),A1))
72         n_p = y1.size-1
73         ste1 = np.sqrt(results1[1]/n_p * np.diagonal(A1tA1inv))
74
75         deltaphi =
76             fs.deltaPhi(self.N,self.deltaZ,self.F,2*self.pupilRadius,self.lbda,self.dxp)
77         y2,A2 = self.initiateMatrix2(deltaphi)
78         results2 = np.linalg.lstsq(A2,y2)
79         ajseven = results2[0]
80         A2tA2inv= np.linalg.inv(np.matmul(np.transpose(A2),A2))
81         n_p = y2.size-1
82         ste2 = np.sqrt(results2[1]/n_p * np.diagonal(A2tA2inv))
83
84         js = np.append(self.oddjs,self.evenjs)
85         ajs = np.append(ajsodd,ajseven)
86         ajsSte = np.append(ste1,ste2)
87
88         Ixjs = np.argsort(js)
89         result = {'js': js[Ixjs], 'ajs': ajs[Ixjs], 'ajsSte': ajsSte[Ixjs]}
90             #,'wavefront':phase}
91         return result
92
93     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 =
99         fs.y2even(fs.getEvenPart(deltaPSFoutFocpos),fs.getEvenPart(deltaPSFoutFocneg))
100    A2 = np.zeros((self.N**2,len(self.evenjs)))
101    for ij in np.arange(len(self.evenjs)):
102        phiJ =
103            fs.f2jeven(self.evenjs[ij],self.N,deltaphi,self.dxp,self.pupilRadius)
104        A2[:,ij] = phiJ
105    return y2,A2
106
107 def CMPTEdeltaPSF(self,deltaZ=[]):
108     if not deltaZ:
109         PSF = psf.PSF([1],[0],self.N,self.dxp,self.pupilRadius)
110         return PSF.Sp**2*self.inFoc - PSF.Sp**2*PSF.PSF
111     else:
112         P2Vdephasing =
113             np.pi*self.deltaZ/self.lbda*(2*self.pupilRadius/self.F)**2/4.
114         a4 = P2Vdephasing/2./np.sqrt(3)
115         PSF = psf.PSF([4],[a4],self.N,self.dxp,self.pupilRadius)
116         return [PSF.Sp**2*self.outFocpos -
117             PSF.Sp**2*PSF.PSF,PSF.Sp**2*self.outFocneg -
118             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
22     # to find a_j even
23     #Get the jth zernike polynomials values on a circular pupil of radius rad
24     Zj = Z.calc_zern_j(j,N,dxp,pupilRadius)
25
26     #compute the different 2Dfft given in the equations of deltaPSF
27     cosZj = np.cos(deltaphi)*Zj
28     sinZj = np.sin(deltaphi)*Zj
29     FFTcosZj = scaledfft2(cosZj,dxp)
30     FFTsinZj = scaledfft2(sinZj,dxp)
31
32     #odd phase
33     oddPhasor = ph.phasor(jsodd,ajsodd,N,dxp,pupilRadius)
34     oddPhase = oddPhasor.phase
35     pupil = oddPhasor.pupil
36     cosOddPhase = pupil*np.cos(deltaphi)*oddPhase
37     sinOddPhase = pupil*np.sin(deltaphi)*oddPhase
38     FFTcosOddPhase = scaledfft2(cosOddPhase,dxp)
39     FFTsinOddPhase = scaledfft2(sinOddPhase,dxp)
40
41     #2Dfft of pupil function times sin(deltaPhi) and cos(deltaPhi)
42     pupilSin = pupil*np.sin(deltaphi)
43     pupilCos = pupil*np.cos(deltaphi)
44     FFTPupilSin = scaledfft2(pupilSin,dxp)
45     FFTPupilCos = scaledfft2(pupilCos,dxp)
46
47     FFTsinZjOddPhase = scaledfft2(sinZj*oddPhase,dxp)
48     FFTcosZjOddPhase = scaledfft2(cosZj*oddPhase,dxp)
49
50     return np.ravel(2*np.imag(np.conj(FFTcosZj) * FFTsinOddPhase + FFTsinZj
51                     * np.conj(FFTcosOddPhase)
52                     - np.conj(FTTPupilCos) * FFTsinZjOddPhase + np.conj(FTTPupilSin)
53                     * FFTcosZjOddPhase))
54
55 def f2jeven(j,N,deltaphi,dxp,pupilRadius): # 2: phij's of matrix A to find
56     # a_j even
57     #Get the jth zernike polynomials values on a circular pupil of radius rad
58     Zj = Z.calc_zern_j(j,N,dxp,pupilRadius)
59     Phasor = ph.phasor([1],[0],N,dxp,pupilRadius)
60     pupil = Phasor.pupil
61
62     #compute the different 2Dfft given in the equations of deltaPSF
63     cosZj = pupil*np.cos(deltaphi)*Zj
64     sinZj = pupil*np.sin(deltaphi)*Zj
65     FFTcosZj = scaledfft2(cosZj,dxp)
66     FFTsinZj = scaledfft2(sinZj,dxp)
67
68     #2Dfft of pupil function times sin(deltaPhi) and cos(deltaPhi)
69     pupilSin = pupil*np.sin(deltaphi)
70     pupilCos = pupil*np.cos(deltaphi)
71     FTTPupilSin = scaledfft2(pupilSin,dxp)
72     FTTPupilCos = scaledfft2(pupilCos,dxp)
73
74     return
75         np.ravel(-4*np.real(np.conj(FTTPupilCos)*FFTsInZj-np.conj(FTTPupilSin)*FFTcosZj))
76
77 def y1(deltaPSFinFoc): #1: yi's of y to find a_j odd
78     #compute the odd part of delta PSF
79     oddDeltaPSF = getOddPart(deltaPSFinFoc)

```

```

73     return np.ravel(oddDeltaPSF)
74
75 def y2(deltaPSFoutFoc,N,jsodd,ajsodd,deltaphi,dxp,pupilRadius): # 2: yi's
76     of y to find a_j even
77     oddDeltaPSF = getOddPart(deltaPSFoutFoc)
78     oddPhasor = ph.phasor(jsodd,ajsodd,N,dxp,pupilRadius)
79     oddPhase = oddPhasor.phase
80
81     pupilSin = oddPhasor.pupil*np.sin(deltaphi)
82     pupilCos = oddPhasor.pupil*np.cos(deltaphi)
83     FFTPupilSin = scaledfft2(pupilSin,dxp)
84     FFTPupilCos = scaledfft2(pupilCos,dxp)
85     cosOddPhase = oddPhasor.pupil*np.cos(deltaphi)*oddPhase
86     sinOddPhase = oddPhasor.pupil*np.sin(deltaphi)*oddPhase
87     FFTcosOddPhase = scaledfft2(cosOddPhase,dxp)
88     FFTsinOddPhase = scaledfft2(sinOddPhase,dxp)
89
90     return np.ravel(oddDeltaPSF -
91                     2*np.real(np.conj(FFTPupilCos))*np.imag(FFTcosOddPhase)
92                     - 2*np.real(np.conj(FFTPupilSin))*np.imag(FTTsInOddPhase))
93
94
95 #Other
96     functions-----
96 def flipMatrix(M):
97     #flip 2D matrix along x and y
98     dimM = (np.shape(M))[0]
99     Mflipped = np.fliplr(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 PSFsizeError(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(PSFsizeError, 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

```

```

7     if (j <= 0):
8         return {'modes':[], 'modesmat':[], 'covmat':0, 'covmat_in':0,
9             'mask':[[0]]}
10    if (N <= 0):
11        raise ValueError("N should be > 0")
12    if (dxp <= 0 or dxp >= N):
13        raise ValueError("dxp should be > 0 or < N")
14
15    xp = np.arange(-Lp/2,Lp/2,dxp)
16    yp = xp
17    [Xp,Yp]=np.meshgrid(xp,yp)
18    r = np.sqrt(Xp**2+Yp**2)
19    r = r*(r<=pupilRadius)/pupilRadius
20    pup = np.float64(np.sqrt(Xp**2+Yp**2)<=pupilRadius)
21    theta = np.arctan2(Yp,Xp)
22
23    Zj = zernike(j,r,theta)*pup
24    return Zj
25
26 def zernike(j,r,theta):
27     n,m = noll_to_zern(j)
28     nc = (2*(n+1)/(1+(m==0)))**0.5
29     #nc = (2*(n+1)/(1+(m==0)))**0.5
30     if (m > 0): return nc*zernike_rad(m, n, r) * np.cos(m * theta)
31     if (m < 0): return nc*zernike_rad(-m, n, r) * np.sin(-m * theta)
32     return nc*zernike_rad(0, n, r)
33
34 def zernike_rad(m, n, r):
35     if (np.mod(n-m, 2) == 1):
36         return r*0.0
37     wf = r*0.0
38     for k in range((n-m)/2+1):
39         wf += r**((n-2.0*k) * (-1.0)**k * fac(n-k) / ( fac(k) * fac(
40             (n+m)/2.0 - k ) * fac( (n-m)/2.0 - k ) ))
41     return wf
42
43 def noll_to_zern(j):
44     j = int(j)
45     if (j == 0):
46         raise ValueError("Noll indices start at 1, 0 is invalid.")
47     n = 0
48     j1 = j-1
49     while (j1 > n):
50         n += 1
51         j1 -= n
52     m = (-1)**j * ((n % 2) + 2 * int((j1+((n+1)%2)) / 2.0 ))
53     return (n, m)
54
55 def fac(n):
56     if n == 0:
57         return 1
58     else:
59         return n * fac(n-1)

```

B.1.5 phasor.py

```

1 #class phasor
2 import numpy as np
3 import zernike as Z
4
5 class phasor(object):
6
7     def __init__(self,js=[1],ajs=[0],N=800,dxp=1,pupilRadius = 200):
8         self.js = js
9         self.ajs = ajs
10        self.N = N
11        self.dxp = dxp
12        self.pupilRadius = pupilRadius
13
14        self.pupil = self.constructPupil()
15
16        self.phase = self.constructPhase()
17        self.phasor = self.pupil*np.exp(-1j*self.phase)
18
19    def constructPhase(self):
20        phase = np.zeros((self.N,self.N))
21        for ij, j in enumerate(self.js):
22            Zj = Z.calc_zern_j(j,self.N,self.dxp,self.pupilRadius)
23            phase += self.ajs[ij]*Zj
24        return phase
25
26    def constructPupil(self):
27        Lp = self.N*self.dxp
28
29        xp = np.arange(-Lp/2,Lp/2,self.dxp)
30        yp = xp
31
32        [Xp,Yp]=np.meshgrid(xp,yp)
33
34        pup = np.float64(np.sqrt(Xp**2+Yp**2)<=self.pupilRadius)
35        return pup

```

B.2 Acquisition Code: Ximea Camera

B.2.1 AlignementScriptXimeaCamera.py

```

1 ##Script to compute the FWHM of the beam on the camera averaging
2 #over "nbrImgAveraging" images and see which position minimizes it.
3
4 from ximea import xiapi
5 import numpy as np
6 from matplotlib import pyplot as plt
7 import scipy.optimize as opt
8 import datetime
9 import functionsXimea as fX
10 import seaborn as sns
11 import os
12 sns.set()
13 #%% instantiation
14
15 dataFolderPath = '...'

```

```
16 plotFolderPath = '...'
17 #create the matrix grid of the detector CCD
18 x = np.linspace(0,1280,1280)
19 y = np.linspace(0,1024,1024)
20 x, y = np.meshgrid(x, y)
21
22 #initial guess for the fit depending on the position of the beam in the CCD
23 initial_guess = [250, 481, 706, 3, 3] # [max PSF,y,x,sigmay,sigmax]
24
25 #number of image to average
26 nbrImgAveraging = 10
27
28 #%%data acquisition and treatment
29
30 #create instance for first connected camera
31 cam = xiapi.Camera()
32 #start communication
33 print('Opening camera...')
34 cam.open_device()
35 #settings
36 cam.set_imgdataformat('XI_MON08') #XIMEA format 8 bits per pixel
37 cam.set_gain(0)
38 #create instance of Image to store image data and metadata
39 img = xiapi.Image()
40 #start data acquisition
41 print('Starting data acquisition...')
42 if cam.get_acquisition_status() == 'XI_OFF':
43     cam.start_acquisition()
44
45 cam.set_exposure(fX.determineUnsaturatedExposureTime(cam,img,[60,10000],1))
46
47 #instanciation for the while loop
48 answer ='y'
49 i=0
50 relativePos = []
51 data = []
52 data_fitted = []
53 FWHMx = []
54 FWHMy = []
55 x0 = []
56 y0 = []
57 sigmaX0 = []
58 sigmaY0 = []
59
60 while answer == 'y':
61
62     try:
63         relativePos.append(float(raw_input('What is the position on the
64             screw [mm] ? ')))
65     except ValueError:
66         print('Not a float number')
67
68     [tmpdata,stdData] = fX.acquireImg(cam,img,nbrImgAveraging)
69     data.append(tmpdata)
70     #Fit the img data on the 2D Gaussian to compute the FWHM
71     print('Fitting 2D Gaussian...')
```

```

71     popt, pcov = opt.curve_fit(fX.TwoDGaussian, (x,y), data[i].ravel(), p0 =
72         initial_guess)
73     print('Fitting done')
74
75     FWHMx.append(2*np.sqrt(2*np.log(2))*popt[3])
76     FWHMy.append(2*np.sqrt(2*np.log(2))*popt[4])
77     x0.append(popt[2])
78     sigmaX0.append(popt[4])
79     y0.append(popt[1])
80     sigmaY0.append(popt[3])
81
81     print 'Fig %d : (x,y) = (%3.2f,%3.2f), FWHM x = %3.2f, FWHM y = %3.2f'
82         %(i,x0[i],y0[i],FWHMx[i],FWHMy[i])
83
83     data_fitted.append(fX.TwoDGaussian((x, y),
84         popt[0],popt[1],popt[2],popt[3],popt[4]).reshape(1024, 1280))
85
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 alignment
95 pressedkey = raw_input('Do you want to acquire an other image [y (yes)
96     or n (no)]: ')
97 if (pressedkey =='n'):
98     answer = pressedkey
99 #increase i
100 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,
140 #                          colors='w',linewidths=0.8)
141     axarr[iImg].set_xlim( (x0[iImg]-12, x0[iImg]+12) )
142     axarr[iImg].set_ylim( (y0[iImg]-12, y0[iImg]+12) )
143     axarr[iImg].set_yticklabels(' ',visible=False)
144     axarr[iImg].set_xticklabels(' ',visible=False)
145     axarr[iImg].set_title('%.3f mm'%relativePos[iImg],fontsize=8)
146     if iImg == indOfMinFWHM:
147         axarr[iImg].set_frame_on(True)
148         for pos in ['top', 'bottom', 'right', 'left']:
149             axarr[iImg].spines[pos].set_edgecolor('r')
150             axarr[iImg].spines[pos].set_linewidth(2)
151     else:
152         axarr[iImg].set_frame_on(False)
153 plt.show()
154 date = datetime.datetime.today()
155
156 plt.savefig(plotFolderPath+date.strftime('%Y%m%d%H%M%S')+'ImgPSF.pdf')
157 plt.savefig(plotFolderPath+date.strftime('%Y%m%d%H%M%S')+'ImgPSF.png')
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
2      them into fits file
3
4 from ximea import xiapi
5 import datetime
6 import functionsXimea as fX
7 import winsound
8 import numpy as np

```

```

9 #%% instantiation
-----
10 #number of image to average
11 nbrImgAveraging = 5000
12 numberOffFinalImages = 1
13
14 #Cropping information
15 sizeImg = 256
16
17 #Parameter of camera and saving
18 folderPathCropped = 'data//cropped/20/'
19 darkFolderPathCropped = '.data/dark//cropped/20/'
20 folderPathFull = 'data/full/'
21 darkFolderPathFull = '/data/dark//full/'
22 nameCamera = 'Ximea'
23 focusPos = 11.63
24
25 #Sound
26 duration = 1000 # millisecond
27 freq = 2000 # Hz
28
29 #initial guess for the fit depending on the position of the beam in the CCD
30 initial_guess = [250, 468, 954, 3, 3] # [max PSF,y,x,sigmay,sigmax]
31
32 #-----
33 #%% data acquisition
-----
34
35 #Opening the connection to the camera
36 cam = xiapi.Camera()
37 cam.open_device()
38 cam.set_imgdataformat('XI_MON08') #XIMEA format 8 bits per pixel
39 cam.set_gain(0)
40
41 img = xiapi.Image()
42 if cam.get_acquisition_status() == 'XI_OFF':
43     cam.start_acquisition()
44 #%% exposition
45 cond = 1
46 while bool(cond):
47     source = ''
48     winsound.Beep(freq, duration)
49     source = int(raw_input('Is the source turned on and at focus point
50         (usually %5.3f mm) (yes = 1) ? '%focusPos))
51     if source == 1:
52         cond = 0
53     else:
54         print 'Please turn on the source and place the camera on the focus
55             point (%5.3f mm)'%focusPos
56
57 if bool(source):
58     #Set exposure time
59     cam.set_exposure(fX.determineUnsaturatedExposureTime(cam,img,[1,10000],1))
60     #get centroid
61     centroid = fX.acquirePSFCentroid(cam,img,initial_guess)
62     print 'centroid at (%d, %d)' %(centroid[0],centroid[1])
63

```

```
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
78             (%5.3f mm) dephase 2Pi (pos+ = %5.3f mm, pos- = %5.3f) ?
79             %(focusPos,focusPos+3.19,focusPos-3.19)))')
80
81         if bool(dark):
82             print 'Acquiring dark image...'
83             # Acquire dark images
84             [darkData, stdDarkData] = fX.acquireImg(cam, img, nbrImgAveraging)
85             print 'Cropping'
86             [darkdataCropped, stddarkDataCropped] =
87                 fX.cropAroundPSF(darkData, stdDarkData, centroid, sizeImg, sizeImg)
88             print 'saving'
89             fX.saveImg2Fits(datetime.datetime.today(), darkFolderPathCropped, nameCamera, darkdataCropped,
90             fX.saveImg2Fits(datetime.datetime.today(), darkFolderPathFull, nameCamera, darkData, stdDarkData)
91
92         #Acquire images -----
93         cond = 1
94         while bool(cond):
95             source = ''
96             winsound.Beep(freq, duration)
97             source = int(raw_input('Is the source turned on (yes = 1) ? '))
98             if source == 1:
99                 cond = 0
100            else:
101                print 'Please place turn on the camera'
102
103            if bool(source):
104                print 'Acquiring images...'
105                # Acquire focused images
106                for iImg in range(numberOfFinalImages):
107                    imgNumber = iImg+1
108                    print 'Acquiring Image %d'%imgNumber
109                    [data, stdData] = fX.acquireImg(cam, img, nbrImgAveraging)
110                    print 'Cropping'
111                    [dataCropped, stdDataCropped] =
112                        fX.cropAndCenterPSF(data-darkData, stdData+stdDarkData, sizeImg, initial_guess)
113                    print 'Saving'
114                    fX.saveImg2Fits(datetime.datetime.today(), folderPathCropped, nameCamera, dataCropped, stdData)
115                    fX.saveImg2Fits(datetime.datetime.today(), folderPathFull, nameCamera, data-darkData, stdDarkData)
116
117            cond = 1
118            while bool(cond):
```

```

115     acquire = ''
116     winsound.Beep(freq, duration)
117     acquire = int(raw_input('Do you want to acquire at an other camera
118                             position (yes = 1, no = 0) ? '))
119     if acquire == 1:
120         cond = 0
121     elif acquire == 0:
122         cond = 0
123     else:
124         print 'please answer with 0 or 1 for no or yes, respectively'
125
126     ##Stop the acquisition
127     cam.stop_acquisition()
128     cam.close_device()
129
130 print 'Acquisition finished'

```

B.2.3 functionsXimea.py

```

1 import numpy as np
2 import pyfits
3 import os
4 import scipy.optimize as opt
5 #%% Functions
-----
6
7
8 #Create and save .fits from numpy array
9 def saveImg2Fits(date,folderPath,Detector,data,stdData,pos,nbrAveragingImg):
10
11     #date : datetime at which the data where taken
12     #folderPath : where to save the data$
13     #Detector : name of detector (ex:Ximea)
14     #data : np.array containing the image
15     #stdData: np.array containing the error on each pixel
16     #pos : the position of the camera on the sliding holder in mm
17
18     imgHdu = pyfits.PrimaryHDU(data)
19     stdHdu = pyfits.ImageHDU(stdData,name = 'imgStdData')
20     hdulist = pyfits.HDUList([imgHdu,stdHdu])
21
22     if not os.path.isdir(folderPath):
23         os.makedirs(folderPath)
24
25     hdulist.writeto(folderPath +
26                      date.strftime('%Y%m%d%H%M%S')+'_'+Detector+'_'+pos+'.fits')
27
28 def acquireImg(cam,img,nbrImgAveraging):
29     imgData = np.zeros([1024,1280])
30     stdData = np.zeros([1024,1280])
31     for iImg in range(nbrImgAveraging) :
32         #print iImg
33         cam.get_image(img)
34         imgTmpData = img.get_image_data_numpy()

```

```
34         imgData[:, :] += imgTmpData
35         stdData += imgTmpData * imgTmpData
36
37     stdData =
38         np.sqrt((stdData - imgData * imgData / nbrImgAveraging) / (nbrImgAveraging - 1)) / (np.sqrt(nbrImgAveraging))
39     imgData = imgData / nbrImgAveraging
40
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' %
59             (exposureTimes[0], exposureTimes[1])
60
61     return int(np.floor(np.nanmean(exposureTimes)))
62
63 def TwoDGaussian((x, y), A, yo, xo, sigma_y, sigma_x):
64     g = A * np.exp(-((x - xo)**2 / (2 * sigma_x**2) + ((y - yo)**2 / (2 * sigma_y**2)))
65     return g.ravel()
66
67 def acquirePSFCentroid(cam, img, initial_guess):
68     #create the matrix grid of the detector CCD
69
70     data = acquireImg(cam, img, 200)[0]
71     centroid = getPSFCentroid(data, initial_guess)
72     return centroid
73
74 def getPSFCentroid(data, initial_guess):
75
76     x = np.linspace(0, 1280, 1280)
77     y = np.linspace(0, 1024, 1024)
78     x, y = np.meshgrid(x, y)
79     print 'fitting'
80     popt, pcov = opt.curve_fit(TwoDGaussian, (x, y), data.ravel(), p0 =
81         initial_guess)
82     print 'fitting done'
83     return [popt[2], popt[1]]
84
85 def cropAndCenterPSF(data, stdData, size, initial_guess):
86     Xextent = np.size(data, 1)
87     Yextent = np.size(data, 0)
```

```
88     centroid = getPSFCentroid(data,initial_guess)
89
90     minMarge =
91         np.min([centroid[0],centroid[1],Xextent-centroid[0],Yextent-centroid[1]])
92
93     if minMarge>size/2:
94         return cropAroundPSF(data,stdData,centroid,size,size)
95     elif minMarge<size/2:
96         return cropAroundPSF(data,stdData, centroid,2*minMarge,2*minMarge)
97
98 def cropAroundPSF(data,stdData,centroid,sizeX,sizeY):
99
100    ppx =
101        [int(np.floor(centroid[0])-np.ceil(sizeX/2)),int(np.floor(centroid[0])+np.ceil(sizeX/2))]
102    ppy =
103        [int(np.floor(centroid[1])-np.ceil(sizeY/2)),int(np.floor(centroid[1])+np.ceil(sizeY/2))]
104
105    dataCropped = data[ppy[0]:ppy[1],ppx[0]:ppx[1]]
106
107    stdDataCropped = stdData[ppy[0]:ppy[1],ppx[0]:ppx[1]]
108
109    return [dataCropped, stdDataCropped]
```

Appendix C

IDL Code

C.1 Shack-Hartmann Acquisition Code

C.1.1 readAndAverageSHdata.pro

```

1 function readAndAverageSHdata, folderPath
2
3 fileExt='*.csv'
4
5 files = file_search(folderPath+fileExt)
6
7 r = readshwfsdata(files[0])
8
9 NFiles = n_elements(files)
10
11 for ifile = 1,Nfiles-1 do begin
12
13   rtmp = readshwfsdata(files[ifile])
14
15   r.wavefront = r.wavefront+rtmp.wavefront
16   r.zernike[3,*] = r.zernike[3,*]+rtmp.zernike[3,*]
17
18 endfor
19
20 r.wavefront = r.wavefront / NFiles
21 r.zernike[3,*] = r.zernike[3,*] / Nfiles
22
23 return, r
24 end

```

C.1.2 readSHWFSdata.pro

```

1 function readSHWFSdata, filePath
2
3 openr, f, filePath, /GET_LUN
4
5 iLine = 0
6 line =
7
8
9 coefficient = []
10 index = []
11 order = []

```

```

12 frequency = []
13 wavefront = []
14
15 while ~EOF(f) do begin
16   readf, f, line
17   iLine += 1
18
19 ;get the zernike coefficient
20 if strmatch(line,'* ZERNIKE FIT *)' then begin
21   subheaderNbrLines = 5
22   for isHd =1,subheaderNbrLines do begin
23     readf, f, line
24     iLine += 1
25   endfor
26
27   readf, f, line
28   iLine += 1
29   sLine = strsplitsplit(line,',',,/EXTRACT)
30
31   while stregex(sLine[0],[0-9]+) ne -1 and ~EOF(f) do begin
32     index = [[index],[long(sLine[0])]]
33     order = [[order],[long(sLine[1])]]
34     frequency = [[frequency],[long(sLine[2])]]
35     coefficient = [[coefficient],[double(sLine[3])]]
36     readf, f, line
37     iLine += 1
38     sLine = strsplitsplit(line,',',,/EXTRACT)
39   endwhile
40   endif
41
42 if strmatch(line,'*/*/* WAVEFRONT /*/*/*') then begin
43   subheaderNbrLines = 11
44   for isHd =1,subheaderNbrLines do begin
45     readf, f, line
46     iLine += 1
47   endfor
48   readf, f, line
49   iLine += 1
50   sLine = strsplitsplit(line,',',,/EXTRACT)
51   nel = n_elements(sLine)
52   while stregex(sLine[0],[0-9]+) ne -1 and ~EOF(f) do begin
53     wavefront = [[wavefront],[double(sLine[1:nel-1])]]
54     readf, f, line
55     iLine += 1
56     sLine = strsplitsplit(line,',',,/EXTRACT)
57   endwhile
58
59   endif
60 endwhile
61 free_lun, f
62
63 zernike = [index,order,frequency,coefficient]
64
65 return, {zernike:zernike,wavefront:wavefront}
66
67 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 schneidenverfahrens 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](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. Instrumentenkde* 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](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/newgroupage9.cfm?objectgroup_id=1400.
- (2018). *Shack-Hartmann Wavefront Sensor*. Thorlabs. URL: https://www.thorlabs.com/newgroupage9.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.