Phase Diversity and Polarization Augmented Techniques for Active Imaging

Peter M. Johnson

Phase Diversity and Polarization Augmented Techniques for Active Imaging

DISSERTATION

Peter M. Johnson, Captain, USAF

AFIT/DS/ENG/07-05

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/DS/ENG/07-05

Phase Diversity and Polarization Augmented Techniques for Active Imaging

DISSERTATION

Presented to the Faculty

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

Peter M. Johnson, B.S.E.E., M.S.E.E.

Captain, USAF

March 2007

AFIT/DS/ENG/07-05

Phase Diversity and Polarization Augmented Techniques for Active Imaging

Peter M. Johnson, B.S.E.E., M.S.E.E.

Captain, USAF

Approved:

Richard K. Martin	01 Mar 07
Dr. Richard K. Martin (Chairman)	Date
[illegible]	1 MAR 07
Dr. Richard G. Cobb (Dean's Representative)	Date
[illegible]	15 FEB 07
LtCol. Matthew E. Goda, PhD (Member)	Date
Mark E. Oxley	1 Mar 07
Dr. Mark E. Oxley (Member)	Date
[illegible]	23 Feb 07
Dr. Victor L. Gamiz (Member)	Date

Accepted:

MU Thomas	9 Mar 07
M. U. Thomas Dean, Graduate School of Engineering and Management	Date

AFIT/DS/ENG/07-05

Abstract

A firm understanding of the space environment is necessary to defend US access to space-based systems. Conventional imaging systems have been developed to gather information on space-based objects, but they are incapable of imaging objects in the earth's shadow. In order close this gap in imaging-system coverage, an active-illumination based approach must be used. To facilitate this, a multi-frame active phase diversity imaging (APDI) algorithm is derived and demonstrated for the statistics of coherent light. In addition to conventional focal-plane and diversity-plane data, a statistical description for the pupil-plane intensity distribution is formed and included in the derivation. The algorithm is implemented and characterized using a Monte Carlo approach. Analysis shows that the algorithm is robust, that the effect of system configuration on optimal algorithm parameters is minimal, that the algorithm is insensitive to detection noise for SNR ≥ 7, and that it performs well for SNR's as low as 2. Furthermore, it's shown that introduction of pupil-plane data on average results in a 60% better image reconstruction from dynamically aberrated data than is obtained using only focal-plane and diversity-plane data.

Both an Expectation-Maximization algorithm and a lensless-APDI approach are presented for generating imagery directly from pupil-plane polarization measurements. Shortfalls of these methods and areas worthy of further consideration are identified. The use of pupil-plane polarization state measurements in place of pupil-plane intensity measurements in the APDI algorithm is explored. A framework for including polarization measurements into the APDI algorithm is demonstrated, and an initial statistical model and results are presented. Under the developed implementation, introduction of the polarization data doesn't result in better performance. Areas that may result in better reconstructions are discussed.

For my family.

Acknowledgements

First and foremost, I owe my love, respect, and deepest gratitude to my wife and children for their support, patience, and long suffering while I was finishing this work. An undertaking such as this impacts the lives of much more than just the person directly involved with it, and my family bore the brunt of the unpleasant side effects. I couldn't have done it without their help.

I am indebted to a great many other people as well, and any attempt at a comprehensive list would fall short. However, I would like to particularly thank LtCol. Goda and Dr. Martin for the personal interest they showed in both me personally and the problems I was working on. Their conversation and suggestions proved invaluable. I am grateful to the Air Force Research Laboratory and Dr. Gamiz specifically for supporting my work and providing the idea that seeded this exploration. The resources that were made available including funding, intellectual support, and industry contacts greatly contributed to my success. Dr. Oxley's questioning and prodding helped keep me on track and make sure that my bases were covered.

The large volume of simulated results I owe to the AFIT high-performance computing center and their liberal usage policies that allowed me to accomplish a great deal of work in short amounts of time by monopolizing a large fraction of system resources. Were these resources not available, it would have taken years to accomplish the same tasks using only my personal workstations.

Peter M. Johnson

Table of Contents

Page

Page

List of Figures

List of Tables

Table Page

List of Symbols

Symbol		Page
z	Propagation direction	4
(ξ, η)	Object-Plane coordinate system, and occasionally coordinates for an intermediate mathematical plane	4
(u, v)	Pupil-Plane/Frequency-Plane coordinate system	4
(x, y)	Data-Plane coordinates, either physical or mathematical	4
Er	Scalar absolute error	7
$\vec{o}$	Parameter vector	7
$\vec{d}$	Observation data vector	7
$\hat{o}$	Parameter estimate	7
$\lVert\cdot\rVert_2$	Euclidean 2-norm	7
C_{mse}	MSE cost function	7
C_u	Uniform cost function	8
$\mathcal{R}$	Bayesian risk	8
$\mathrm{E}\left[\cdot\right]$	Statistical Expectation	8
$\mathcal{L}(\cdot)$	Log-Likelihood	10
$\tilde{\mathcal{D}}$	Space over which "hidden" data is defined in the EM algorithm	11
$\vec{g}$	Gradient vector	13
$\mathbf{G}$	Hessian for optimization	13
$\mathbf{U}$	Unitary matrix of eigenvectors	14
$\mathbf{\Lambda}$	Diagonal matrix of eigenvalues	14
$\mathbf{L}$	Unit lower-triangular matrix	14
$\mathbf{D}$	Diagonal matrix	14
$\vec{\lambda}_l$	Lagrange Multiplier Vector	16
$\vec{E}$	Electric field vector	17
$\vec{B}$	Magnetic flux density	17
$\vec{\mathcal{H}}$	Magnetic field	17

Symbol		Page
$\vec{D}$	Electric displacement	17
J	Free current density	17
ρ	Free charge density	17
$\tilde{n}$	Index of refraction	18
c	Speed of light	18
U	Complex field component	18
$\hat{i}$	Unit vector along x	18
$\hat{j}$	Unit vector along y	18
A_1	Real field amplitude along x	18
A_2	Real field amplitude along y	18
k	Wavenumber $\left(\frac{2\pi}{\lambda}\right)$	18
ω	Angular frequency	18
θ_{12}	Polarization phase	19
Ψ	Polarization ellipse rotation angle	19
$\vec{S}$	Stokes polarization vector	20
$\mathcal{P}$	Degree of polarization	21
θ_i	Incident angle	22
θ_t	Transmitted angle	22
$\chi\left(\cdot\right)$	Obliquity factor	24
τ	Time delay	26
Γ_{12}	Mutual coherence function	26
J_{12}	Mutual intensity	26
σ^2	Variance	28
I_o	Intensity at the object	28
I_p	Far-field/Pupil-plane intensity	28
ψ_p	Quadratic propagation phase	29
$\mathcal{F}\{\cdot\}$	Fourier transform	30
Φ_{PP}	PP speckle PSD	30

Symbol		Page
Γ_{PP}	Speckle autocorrelation	30
μ_c	Complex coherence coefficient	30
$\hat{\Phi}_{nf}$	Normalized near-field PSD	30
$\circledast$	2-D autocorrelation	31
$\mathbf{\Gamma}_U$	Complex field correlation matrix	31
μ	Complex coherence magnitude	32
ψ	Complex coherence phase	32
I_0	Modified Bessel function, first kind, zero order	32
U_o	Object-plane field	33
z_o	Distance between the object and imaging system pupil	33
U_p	Pupil-plane field	33
U_i	Image-plane field	33
z_i	Distance from the pupil-plane to the image-plane	33
f_l	Imaging-system focal length	33
P	Imaging-system pupil-transmission function	33
h	Impulse response	34
M	Magnification	35
I_i	Image-plane intensity	36
s	Point-Spread-Function	36
ϕ	Aberration phase	37
Φ_n	Kolmogorov PSD for index of refraction fluctuations	38
κ	Spatial wave number	38
L_o	Turbulence outer scale	38
l_o	Turbulence inner scale	38
C_n^2	Index of refraction structure function constant	38
L	Turbulent layer thickness	39
$\tilde{n}_0$	Average index of refraction	39
$\tilde{n}_1$	Index of refraction fluctuations caused by turbulence	39

Symbol		Page
Γ_ϕ	Phase correlation function	40
D_ϕ	Phase structure function	40
r_0	Fried parameter	40
$\mathbf{\Gamma}_z$	Zernike-mode covariance for Kolmogorov statistics	41
D	Imaging system pupil-diameter	41
ϕ_D	Known diversity phase	43
α_k	k^{th} aberration Zernike expansion coefficient	44
β_k	k^{th} diveristy-phase Zernike expansion coefficient	44
ψ_k	k^{th} Zernike-mode	44
g_m	Noiseless data model	44
$\mathbf{o}$	Idealized geometrical optics image	44
$\otimes$	Two-dimensional convolution	44
d_m	Noisy data for m^{th} channel	45
$\hat{n}_m$	Additive white Gaussian noise for the m^{th} channel	45
$\mathcal{D}$	Multi-frame or multi-plane data set	45
$\vec{\alpha}$	Vector of aberration expansion coefficients	45
$\mathcal{Im}\{\cdot\}$	Imaginary part	46
$\mathcal{Re}\{\cdot\}$	Real part	50
$\boldsymbol{\alpha}$	Matrix of aberration parameter vectors	50
χ	Arbitrary parameter	61
γ_{tol}	L-BFGS-B stopping tolerance	69
σ_d	Detector read-noise standard deviation	70
p	Average photo-electrons per pixel	70
e^-	photo-electrons	70
θ_s	Unknown Propagation Phase	97
A_s	S-polarization field amplitude	97
A_p	P-polarization field component	97
θ_{12}	Polarization Phase	97

List of Abbreviations

List of Listings

Phase Diversity and Polarization Augmented Techniques for Active Imaging

I. Introduction

Recent military engagements in Iraq and Afghanistan have shown the great impact that space-based systems can have on the battlefield. The strongly asymmetric advantage in space enjoyed by the United States armed forces has been cited as one of the largest factors in the tactical success of Operation Iraqi Freedom and the Global War on Terror in general [16]. With growing dependence on space-based assets, it's becoming more and more important to protect those assets and ensure uninterrupted availability to the war-fighter. Gen. Lance W. Ward, commanding general Air Force Space Command, warned "[US control of space] is not a birthright or a destiny." [27] Recognizing this fact, the United States Air Force has made protection of space-based assets a high priority.

1.1 Motivation

In order to secure access to and use of space-based assets, a clear understanding of the space environment and associated threats known as Space Situational Awareness (SSA) is required. To acquire the desired level of SSA, tools and techniques for identifying, tracking, cataloging, monitoring, characterizing, and assessing objects in orbit have been and continue to be developed and improved. One of the most desirable analytical tools for characterizing or assessing an extra-terrestrial object is the ability to generate high-resolution imagery of that object. High-resolution imagery can provide a wealth of information about the object of interest, and is critical for the identification of unknown objects or monitoring of foreign/friendly systems. Imagery can provide information required to positively identify unknown systems, identify payloads on potentially hostile objects, and assist in the troubleshooting of failed systems.

While techniques exist for imaging satellites (see for example [29, 39, 42]), optimal use of conventional methods cannot cover all of the engagement scenarios critical to SSA. Conventional nighttime imaging systems can only image a satellite during the brief period near terminator when the observation system is in the earth's shadow and the object is illuminated by the sun. However, there are many desirable engagement scenarios that place the satellite beyond terminator and in the earth's shadow. In this case there is no light being reflected from the object with which to form an image, and conventional imaging systems fail.

As an alternative approach, the unique properties of laser light can be used to overcome some of these limitations and provide SSA information that would otherwise be unavailable. The ability of lasers to generate intense focused optical radiation can be used to overcome the requirement for solar illumination by effectively generating a "flash bulb" for the imaging system that shifts it from a passive to an active imaging mode. Unfortunately, the same coherence properties that make it possible to project intense light over long distances also introduce complications unique to coherent light. This work attempts to compensate for and even leverage these coherence properties to generate high-resolution imagery of exo-atmospheric objects.

1.2 Overview

Both active and conventional imaging systems suffer from the effects of atmospheric turbulence. As a result, any system designed to produce high-resolution imagery must be capable of compensating for or otherwise negating the aberration and associated blurring caused by atmospheric turbulence. One technique for compensating for the blurring effects of the atmosphere is the use of phase diversity (PD) image reconstruction [23, 39]. In conventional PD imaging, two images are simultaneously collected where the first image is obtained in the conventional focal plane (FP), and the second is obtained in a diversity plane (DP) that is defocused from the focal plane by some known amount. The data is then post-processed to produce an estimate of both the aberration and the object.

Because of the severity of laser speckle noise, a multi-frame approach is used to derive a maximum-likelihood active phase diversity imaging (APDI) reconstruction algorithm similar to that of Seldin *et al.* [45]. The resulting algorithm is essentially a hybrid between multi-frame blind deconvolution and conventional PD reconstruction techniques. In addition to the modifications required by speckle noise statistics, additional information made available by coherent illumination is included in the derivation to better condition the reconstruction.

Specifically, a significant amount of information about the object being imaged is encoded in the imaging system's pupil-plane (PP) intensity distribution. A statistical description of the PP intensity is developed and incorporated into the derivation of the reconstruction algorithm. The completed APDI algorithm is implemented and characterized using Monte-Carlo style simulations.

Two scenarios are developed. First, a version of the algorithm tailored to static aberrations is developed. This configuration covers situations where the system suffers from an unknown systemic aberration or when the time between frames is much smaller than the atmospheric decorrelation time. The non-time-varying nature of the atmospheric aberration simplifies the problem, making the resulting computations significantly faster. The second scenario considered is the case where atmospheric turbulence dynamically evolves and realizations are completely decorrelated between data frames. This configuration represents the most interesting, practical, computationally expensive, and challenging system.

The effects of key parameters (conditioning bias and convergence tolerance) on algorithm performance are evaluated for both static and dynamic aberrations by varying each parameter and generating 100 random data realizations. The residual mean-squared error (MSE) between the reconstructed image and the known truth object is computed for each realization. The effect of parameter variation on MSE is used to determine optimal parameter values for multiple scenarios. Overall performance in the presence of detection noise is characterized by generating 100 random realizations with signal to noise ratio (SNR) values ranging SNR $= 2$ to SNR $= \infty$ with 10, 20, 30, 40, and 50 frames of data, then processing them with near optimal bias and convergence tolerance. MSE, mean computation time, and average function evaluations are determined for each reconstruction. The effect of including additional terms in the reconstructed turbulence representation is investigated by reconstructing images using 5, 10, 15, 20, 30, 50, and 100 Zernike modes and again comparing MSE, computation time, and number of function evaluations.

To quantify the impact of adding PP data to the algorithm, versions of the algorithm excluding the PP data and alternatively the DP data are compiled and run along with the complete algorithm on 100 data realizations. MSE, convergence time, and function evaluations are computed for each realization and compared between the three algorithm configurations.

As an alternative approach to active imaging, three methods are developed for reconstructing images using PP polarization data. First, a statistical model for the PP polarization state is developed and used to build the framework for an Expectation-Maximization (EM) algorithm for reconstructing the image directly from PP polarization state measurements. Second, the PP polarization data is mathematically shaped to fit within the framework of the APDI algorithm, and the algorithm is modified accordingly. Finally, in an effort to expand the information available to the initial APDI algorithm and produce better results, a statistical model for polarization phase data is developed, and the APDI algorithm expanded to utilize PP polarization state measurements in place of PP intensity measurements.

1.3 Notation and Conventions

To the greatest extent possible, consistent notation is used throughout the text. Where possible, a given symbol retains its interpretation once defined. Most symbols used throughout the text are included in a list of symbols located in the prefatory material. Symbols not defined in the list of symbols either change depending on context, or are only used in a limited section of the text and not referenced elsewhere. A list of abbreviations is also included in the prefatory material for reference.

The general geometry used for all notation in this work is shown in Fig. 1.1. Propagation is taken to be along the z axis. The coordinate set (ξ, η) is used for the object plane (OP), and occasionally for an intermediate mathematical plane. The coordinate set (u, v) is used for pupil-plane (PP) measurements and/or a frequency-domain coordinate system. The coordinate system (x, y) is used for the FP, DP, and all data arrays that directly interface with the APDI algorithm, regardless of whether or not they correspond with a physical plane.

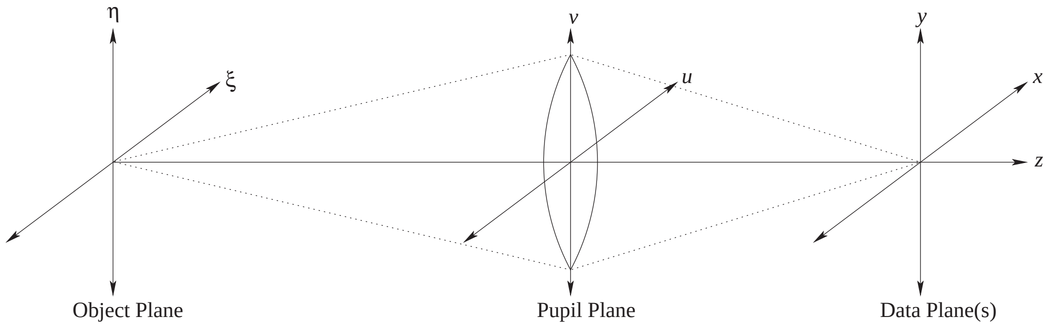


Figure 1.1: Generalized geometry showing notational conventions. (ξ, η) are used for object-plane coordinates, (u, v) for pupil-plane coordinates, and (x, y) for data plane coordinates. z is used for the coordinate along the direction of propagation in all planes.

1.4 Document Organization

The document is organized as follows: Chapter II introduces estimation and optimization theory applicable to the active imaging problem and image reconstruction techniques. Chapter III describes applicable optical phenomena including light propagation, coherence theory, and applicable statistical characteristics of laser speckle patterns. Chapter IV introduces the theory and models used for imaging system performance and atmospheric turbulence. Chapter V contains the development of the APDI algorithm. Chapter VI presents the results of Monte Carlo analysis of the APDI algorithm and a discussion of their significance. Chapter VII builds the framework for use of PP polarization state measurements in image reconstruction and presents the initial results obtained. Chapter VIII summarizes the contributions and conclusions reached as a result of this work and highlights areas for future work. Computer code used to implement the APDI algorithm is included in the appendices.

II. Applicable Estimation and Optimization Theory

At the core of probabilistic image reconstruction techniques is estimation of unknown information based on available data. To do this, a basic understanding of estimation and optimization theory is required. This chapter covers estimation theory and optimization methods applicable to building the statistically based reconstruction algorithms considered.

2.1 Estimation Theory

All information covering estimation in this section is derived or taken directly from [49]. In the most basic form, estimation is simply a well-defined statistical "best guess" at desired data (parameters) based on corrupted or incomplete data (observations) using a set of predefined criteria known as a cost function. The basic estimation problem is composed of four main components shown in Fig. 2.1: the parameter space, probabilistic mapping, observation space, and an estimation rule.

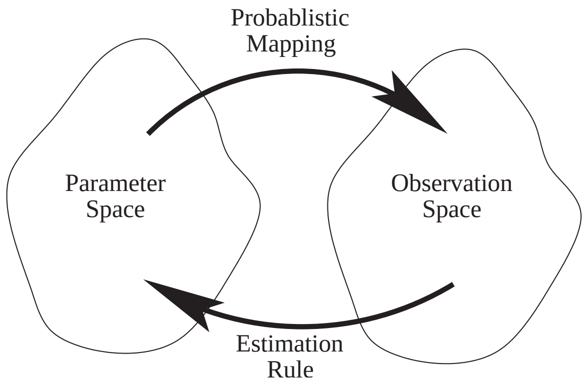


Figure 2.1: Basic elements of an estimator include the parameter space containing the parameters to be estimated, an observation space where data is collected, a probabilistic mapping relating the parameters to observations, and an estimation rule linking observations with parameters.

The parameter space is made up of all possible parameters and contains the data that corresponds to the desired "truth" result. The output of the estimation problem is an element of the parameter space which hopefully corresponds to the "truth." The probabilistic mapping models whatever process corrupts the parameters to be estimated, and is posed in the form of probability distributions describing statistical behavior of the parameters and observations. The observation space is composed of all possible observations, and may or may not intersect the parameter space. The estimation rule is the mapping used to map elements of the observation space back into the parameter space and thereby form an estimate of the desired parameter(s).

The probabilistic mapping is frequently a function of the operating environment, and cannot in general be altered to aid the development of an estimator. Similarly the observation and parameter spaces are not typically customizable for a given problem. The task then becomes development of an adequate estimation rule.

2.1.1 Bayesian Estimation. One of the first considerations when developing an estimation rule is what exactly must be accomplished. For example, one could conceive a situation such as monitoring for a toxic substance in the environment where overestimation of a parameter might be relatively acceptable, but underestimating it could have disastrous consequences. Alternatively, when monitoring something like oxygen concentration in breathable air the converse situation could be true, and overestimation could be deadly. Given these examples, it should be clear that no one estimation rule will work for all applicable problems.

To incorporate requirement variability into the estimation rule, a cost function C is introduced. In general, a cost function is a scalar valued function of both the parameters and the observations whose value is proportional to the "badness" of a given combination of the two. Frequently, the cost function can be written as a function of the scalar absolute error Er, defined as

$$Er(\hat{o},\vec{o}) \equiv \|\hat{o}(\vec{d}) - \vec{o}\|_2 \tag{2.1}$$

where $\vec{o}$ is the parameter vector, $\vec{d}$ is the observation vector, $\hat{o}$ is the estimate of $\vec{o}$ generated by the estimation rule, and $\|\cdot\|_2$ is the standard Euclidean norm.

While the cost function can be tailored to any specific application, the two most commonly used cost functions, shown in Fig. 2.2, are the mean-square error cost function C_{mse}, defined as

$$\begin{aligned} C_{mse} &= [Er\,(\hat{o},\vec{o})]^2 && \text{(2.2a)} \\ &= (\hat{o}-\vec{o})^T\,(\hat{o}-\vec{o}) && \text{(2.2b)} \end{aligned}$$

which increases the cost quadratically as the estimate moves further away from the true parameters; and the uniform cost function C_u defined as

$$C_u = \begin{cases} 0, & Er\left(\hat{o}, \vec{o}\right) \leq \frac{\Delta}{2} \\ 1, & Er\left(\hat{o}, \vec{o}\right) > \frac{\Delta}{2} \end{cases} \tag{2.3}$$

which assigns an equal cost to all estimates outside of a neighborhood of width Δ centered about the true parameter vector, and assigns no cost to estimates within the neighborhood.

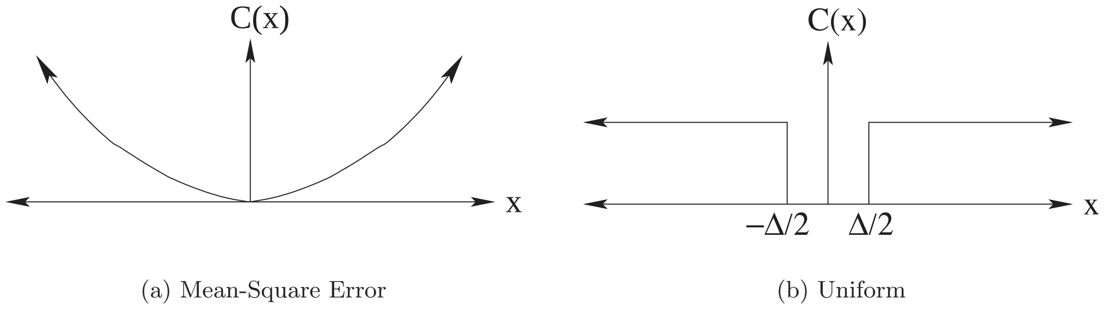


Figure 2.2: One-dimensional mean-square error and uniform cost functions for Bayesian estimation.

Once the cost function has been selected, a Bayesian estimation rule can be formed given that the joint probability of the parameters and observations $p(\vec{o}, \vec{d})$ is known. The Bayesian estimation rule is developed by minimizing the risk $\mathcal{R}$ defined as

$$\begin{aligned} \mathcal{R} &= \mathrm{E}\left[C(\vec{o}, \hat{o})\right] \\ &= \iint_{-\infty}^{\infty} C(\vec{o}, \hat{o}) p(\vec{o}, \vec{d}) \mathrm{d}\vec{d} \mathrm{d}\vec{o} \end{aligned} \tag{2.4}$$

where $\mathrm{E}\left[\cdot\right]$ represents statistical expectation.

As an example, the Bayesian estimation rule for a mean square error cost function can be found by substituting (2.2) into (2.4) and minimizing

$$\mathcal{R}_{mse} = \iint_{-\infty}^{\infty} \left(\hat{o} - \vec{o}\right)^T \left(\hat{o} - \vec{o}\right) p(\vec{o}, \vec{d}) \mathrm{d}\vec{d} \mathrm{d}\vec{o} \tag{2.5}$$

Noting that $p(\vec{o},\vec{d}) = p(\vec{o}|\vec{d})p(\vec{d})$, (2.5) can be written

$$\mathcal{R}_{mse} = \int_{-\infty}^{\infty} p(\vec{d}) \int_{-\infty}^{\infty} (\hat{o} - \vec{o})^T (\hat{o} - \vec{o}) p(\vec{o}|\vec{d}) \mathrm{d}\vec{o} \mathrm{d}\vec{d} \tag{2.6}$$

Because both $p(\vec{d})$ and the inner integral of (2.6) are non-negative, minimizing the inner integral

$$\int_{-\infty}^{\infty} (\hat{o} - \vec{o})^T (\hat{o} - \vec{o}) p(\vec{o}|\vec{d}) \mathrm{d}\vec{o} \tag{2.7}$$

will also minimize $\mathcal{R}_{mse}$. The value of $\hat{o}$ that minimizes this integral is then the minimum-mean-square estimate $\hat{o}_{ms}$.

To minimize the inner integral, (2.7) is differentiated with respect to $\hat{o}$ and the result set equal to zero. Using standard definitions from vector calculus, the derivative of (2.7) is given by

$$\frac{\mathrm{d}}{\mathrm{d}\hat{o}} \int_{-\infty}^{\infty} (\hat{o} - \vec{o})^T (\hat{o} - \vec{o}) p(\vec{o}|\vec{d}) \mathrm{d}\vec{o} = \int_{-\infty}^{\infty} \frac{\mathrm{d}}{\mathrm{d}\hat{o}} (\hat{o} - \vec{o})^T (\hat{o} - \vec{o}) p(\vec{o}|\vec{d}) \mathrm{d}\vec{o} \tag{2.8a}$$

$$= 2\hat{o} \int_{-\infty}^{\infty} p(\vec{o}|\vec{d}) \mathrm{d}\vec{o} - 2 \int_{-\infty}^{\infty} \vec{o} p(\vec{o}|\vec{d}) \mathrm{d}\vec{o} \tag{2.8b}$$

Setting (2.8) equal to zero and noting that $\int_{-\infty}^{\infty} p(\vec{o}|\vec{d}) \mathrm{d}\vec{o} = 1$, the best estimate $\hat{o}_{ms}$ is given by

$$\hat{o}_{ms} = \int_{-\infty}^{\infty} \vec{o} p(\vec{o}|\vec{d}) \mathrm{d}\vec{o} \tag{2.9}$$

which is the expected value of the conditional probability distribution function $p(\vec{o}|\vec{d})$.

2.1.2 Maximum A Posteriori Estimation. Another noteworthy example is the Bayesian estimator that results from the uniform cost function of (2.3) when Δ is made arbitrarily small. This estimation rule is given the special name maximum *a posteriori* (MAP) estimation. The risk for the associated cost function is given by

$$\mathcal{R}_{map} = \int_{-\infty}^{\infty} p(\vec{d}) \mathrm{d}\vec{d} \left[1 - \int_{\hat{o}-\Delta/2}^{\hat{o}+\Delta/2} p(\vec{o}|\vec{d}) \mathrm{d}\vec{o} \right] \tag{2.10}$$

As before, the integral can be minimized by minimizing the inner term, in this case the term in the brackets. For an arbitrarily small Δ, it can be seen that the minimum of the term in brackets is achieved when the *a posteriori* probability distribution function $p(\vec{o}|\vec{d})$ is at a maximum. The problem them becomes determination and maximization of the distribution.

Frequently, the distributions have the parameter of interest in an exponent, and it becomes convenient to work with the logarithm of the distribution. Given that the logarithm is a monotonically increasing function, maximization of the logarithm is equivalent to maximizing the distribution function. Therefore at the MAP estimate value

$$\left.\frac{\partial \ln p(\vec{o}|\vec{d})}{\partial \vec{o}}\right|_{\vec{o}=\hat{o}_{map}} = 0 \tag{2.11}$$

Bayes rule can then be used to separate (2.11) into *a priori* knowledge and observation knowledge.

$$\left.\frac{\partial \mathcal{L}(\vec{o})}{\partial \vec{o}}\right|_{\vec{o}=\hat{o}_{map}} = \frac{\partial}{\partial \vec{o}}\left.\Big(\ln p(\vec{d}|\vec{o}) + \ln p(\vec{o})\Big)\right|_{\vec{o}=\hat{o}_{map}} = 0 \tag{2.12}$$

where constant terms have been dropped and $\mathcal{L}(\cdot)$ is known as the log-likelihood function.

2.1.3 Maximum Likelihood Estimation. Unfortunately, the parameter distribution $p(\vec{o})$ is frequently unknown. Without prior information about the parameter distributions, the best that can be done is to assume that the parameters are uniformly distributed throughout the parameter space. Applying this to the MAP estimation described in the previous section, $p(\vec{o})$ is constant and can be dropped. This results in the maximum likelihood (ML) estimate given by

$$\begin{aligned}\left.\frac{\partial \mathcal{L}(\vec{o})}{\partial \vec{o}}\right|_{\vec{o}=\hat{o}_{map}} &= \left.\frac{\partial \ln p(\vec{d}|\vec{o})}{\partial \vec{o}}\right|_{\vec{o}=\hat{o}_{map}} \\ &=0\end{aligned} \tag{2.13}$$

2.1.4 Expectation Maximization Algorithm. In many cases, a closed form for the maximum-likelihood estimate cannot be found because either the statistical model for the data contains parameters that are unknown, or the likelihood function cannot be easily differentiated. In these cases, an alternative approach must be taken. If something is known about the statistics of the missing data, or the likelihood function can be simplified by adding additional "hidden" data, the Expectation-Maximization (EM) algorithm [5, 13, 15, 36, 42] can be used to form an estimate.

Within the framework of the EM algorithm, the measured data $\vec{d}$ is termed the "incomplete" data (ID), and is generated from a subset of "complete" data (CD) that includes the unknown or "hidden" data $\tilde{d}$. The problem is modeled in terms of the joint probability density function of the complete data conditioned on the parameters $\vec{o}$ that are to be estimated

$$p(\vec{d}, \tilde{d}|\vec{o}) = p(\tilde{d}|\vec{d}, \vec{o})p(\vec{d}|\vec{o}) \tag{2.14}$$

In the event that the "hidden" data is artificially inserted into the problem, the marginal distribution $p(\tilde{d}|\vec{d}, \vec{o})$ is chosen such that it puts the joint density function into a form that is both convenient for calculation and statistically consistent with the measured data. Otherwise, the marginal distribution is determined by the nature of the problem.

Once the joint probability distribution function has been determined, the CD log-likelihood function for $\vec{o}$ can be formed such that

$$\mathcal{L}(\vec{o}|\vec{d}, \tilde{d}) = \ln p(\vec{d}, \tilde{d}, |\vec{o}) \tag{2.15}$$

With the CD log-likelihood function defined, the statistical model for the system is complete and the algorithm can proceed. As the name suggests, the algorithm consists of two components; an expectation component, and a maximization component. In the expectation step, the CD log-likelihood function given the incomplete data and the current estimate of the parameters is expectated over the "hidden" data such that

$$\begin{aligned} Q(\vec{o}, \hat{o}^{(i)}) &= \mathrm{E}\left[\mathcal{L}(\vec{o}|\vec{d}, \tilde{d})|\vec{d}, \hat{o}^{(i)}\right] && (2.16a) \\ &= \mathrm{E}\left[\ln\left\{p(\vec{d}, \tilde{d}|\vec{o})\right\}|\vec{d}, \hat{o}^{(i)}\right] && (2.16b) \\ &= \int_{\tilde{d}\in\tilde{\mathcal{D}}} p(\tilde{d}|\vec{d}, \hat{o}^{(i)}) \ln\left\{p(\vec{d}, \tilde{d}|\vec{o})\right\} \mathrm{d}\tilde{d} && (2.16c) \end{aligned}$$

where $\hat{o}^{(i)}$ is the current estimate of the parameter, $p(\tilde{d}|\vec{d}, \hat{o}^{(i)})$ is the marginal density of the "hidden" data conditioned on the incomplete data and current parameter estimate, and $\tilde{\mathcal{D}}$ is the space over which $\tilde{d}$ is defined. One common problem is that the marginal density of the "hidden" data is unobtainable. In these cases, $p(\tilde{d}|\vec{d}, \hat{o}^{(i)})$ is often replaced by $p(\tilde{d}, \vec{d}|\hat{o}^{(i)}) = p(\tilde{d}|\vec{d}, \hat{o}^{(i)})p(\vec{d}|\hat{o}^{(i)})$ [5].

The second step of the EM algorithm is the maximization step in which the expected value of the CD log-likelihood function $Q(\vec{o}, \hat{o}^{(i)})$ is maximized with respect to $\vec{o}$ using any appropriate optimization technique to generate an updated estimate of the parameter $\vec{o}^{(i+1)}$ such that

$$\hat{o}^{(i+1)} = \underset{\vec{o}}{\operatorname{argmax}}\, Q(\vec{o}, \hat{o}^{(i)}) \tag{2.17}$$

2.2 Numerical Optimization Methods

Frequently the maximization involved in forming the MAP/ML estimates or the EM-iterates cannot be efficiently accomplished analytically due to the size of the problem or complex nature of the objective function being maximized. In these cases, iterative numerical solution methods can be used to generate a sequence of estimates that converge to the optimal solution. The iterative solution techniques in this section are adapted from [21] chapter 4, and are used to minimize a scalar-valued objective function $F(\vec{o})$ such that

$$\hat{o} = \underset{\vec{o}}{\operatorname{argmax}}\, F(\vec{o}) \tag{2.18}$$

To utilize these methods for maximization, the objective function is simply multiplied by -1.

Nearly all numerical optimization techniques are based on the same basic algorithm composed of the following major steps:

1. Develop/define an initial guess for the estimate $\hat{o}_k$ that is consistent with the problem.
2. Check the current estimate $\hat{o}_k$ of the solution $\hat{o}$ for convergence. If the convergence criteria are satisfied, stop the iteration.
3. Compute the "optimal" search direction $\vec{p}_k$ given $\hat{o}_k$ and the objective function $F(\vec{o})$.
4. Compute the "optimal" step size α_k.
5. Update the estimate $\hat{o}_{k+1} = \hat{o}_k + \alpha_k \vec{p}_k$ to produce the next iterate.

6. Repeat, beginning at step 2.

Determination of convergence criteria and the definition of “optimal” are based on the specific method being used.

Additional terms that are frequently used are the gradient vector $\vec{g}$ (analogous to the first derivative for a univariate function)

$$\nabla F \equiv \vec{g}(\vec{o}) = \begin{pmatrix} \frac{\partial F}{\partial o_1} \\ \vdots \\ \frac{\partial F}{\partial o_n} \end{pmatrix} \tag{2.19}$$

and the Hessian $\mathbf{G}$ (analogous to the second derivative for the univariate case)

$$\nabla^2 F \equiv \mathbf{G}(\vec{o}) = \begin{pmatrix} \frac{\partial^2 F}{\partial o_1^2} & \cdots & \frac{\partial^2 F}{\partial o_1 \partial o_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 F}{\partial o_1 \partial o_n} & \cdots & \frac{\partial^2 F}{\partial o_n^2} \end{pmatrix} \tag{2.20}$$

For convenience, $\vec{g}(\hat{o}_k)$ and $\mathbf{G}(\hat{o}_k)$ will be written $\vec{g}_k$ and $\mathbf{G}_k$ respectively.

2.2.1 Newton's Method. For purposes of iterative optimization, it's usually sufficient to approximate an arbitrary objective function $F(\hat{o}_k + \alpha_k \vec{p}_k)$ as a quadratic function through a Taylor-series expansion about $\hat{o}_k$ such that in a small region around $\hat{o}_k$,

$$F(\hat{o}_k + \alpha_k \vec{p}_k) \approx F_k + \alpha_k \vec{g}_k^H \vec{p}_k + \frac{\alpha^2}{2} \vec{p}_k^H \mathbf{G}_k \vec{p}_k \tag{2.21}$$

Given this quadratic form, the minimum will occur when $\vec{p}_k$ is a minimum of the quadratic function

$$\Phi(\vec{p}_k) = \alpha_k \vec{g}_k^H \vec{p}_k + \frac{\alpha_k^2}{2} \vec{p}_k^H \mathbf{G}_k \vec{p}_k \tag{2.22}$$

Differentiating and setting the result to zero, the minimum then satisfies

$$\alpha_k \mathbf{G}_k \vec{p}_k = -\vec{g}_k \tag{2.23}$$

Setting $\alpha_k = 1$ results in Newton's method for a multivariate function. If F is exactly quadratic and $\mathbf{G}_0$ is positive definite, Newton's method converges to the global minimum,

$F(\hat{o}_k + \vec{p}_k) = F(\hat{o})$, in a single step. Provided the approximation of (2.21) is relatively accurate, the initial starting point $\hat{o}_0$ is not too far from the minimum point $\hat{o}$, and $\mathbf{G}_k(\hat{o})$ is positive definite, Newton's method will converge quadratically to the true minimum $F(\hat{o})$.

2.2.2 Quasi-Newton's Methods. In cases where Newton's method fails to converge to a correct solution, alternative solution techniques can be used. For example, if the Hessian $\mathbf{G}_k(\hat{o})$ is not positive definite, Quasi-Newton's methods can be used wherein a positive definite "related Hessian" $\bar{\mathbf{G}}_k$, closely related to $\mathbf{G}_k$, is formed at each iteration. Generally, $\bar{\mathbf{G}}_k$ is formed using some form of factorization on $\mathbf{G}_k$. The search direction $\vec{p}_k$ is then determined by solution of the system

$$\bar{\mathbf{G}}_k \vec{p}_k = \vec{g}_k \tag{2.24}$$

The two most common factorizations used for forming $\bar{\mathbf{G}}_k$ are the spectral decomposition given by

$$\mathbf{G} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^H \tag{2.25}$$

where $\mathbf{U}$ is a unitary matrix of eigenvectors and $\boldsymbol{\Lambda}$ is a diagonal matrix of the eigenvalues of $\mathbf{G}$, and a modification of the Cholesky factorization given by

$$\mathbf{G} = \mathbf{L}\mathbf{D}\mathbf{L}^H \tag{2.26}$$

where $\mathbf{L}$ is a unit lower-triangular matrix and $\mathbf{D}$ is a diagonal matrix. These factorizations both provide clear indications of whether or not a given matrix is positive definite, and what must be done to make it so. The "related Hessian" is then formed by making the "smallest" changes possible to the true Hessian that will ensure positive definiteness. For example, if the modified Cholesky factorization results in a diagonal matrix with negative entries, those negative entries are replaced with an appropriately small positive number. The magnitude of the replacement is selected based on the maximum acceptable condition number. Once the related Hessian has been formed, the iterations proceed as with standard Newton's method solutions except that a new related Hessian is formed at each iteration.

2.2.3 Other Common Methods. In cases where both Newton's and quasi-Newton's methods fail to converge to a solution, alternative solution methods including gradient-descent, conjugate gradient, multi-grid, and other related numerical solution methods can be used. For descriptions and applications of these methods see for example [14, 21].

2.2.4 Constrained Optimization. In many cases physical processes result in constraints on the solution vector $\hat{o}$ that must be satisfied for it to be a valid solution. To accommodate these situations, Lagrange multipliers can be used to incorporate the constraints into the problem [18]. The modified problem can then be passed to one of the previously discussed solution methods to find the optimal solution given the constraints.

As a motivating example, define the objective function as

$$F(\vec{o}) = \vec{o}^H \mathbf{M} \vec{o} \tag{2.27}$$

where H is the hermitian transpose and $\mathbf{M}$ is a hermitian matrix, and constrain the solution such that

$$\vec{o}^H \vec{o} = n \tag{2.28}$$

Without applying the constraints, (2.27) can be minimized by taking the derivative with respect to $\vec{o}$ and setting the result equal to zero, resulting in

$$M\vec{o} = 0 \tag{2.29}$$

If M is non-singular, the minimizing solution is then $\vec{o} = 0$. However, the problem constraints eliminate this solution from consideration, and an alternative approach has to be taken.

To accommodate constraints, the Lagrangian is formed using the objective function $F(\vec{o})$, the constraints, and a vector of Lagrange multipliers such that

$$L(\vec{o}, \vec{\lambda}_l) = F(\vec{o}) + \vec{\lambda}_l^T c(\vec{o}) \tag{2.30}$$

where $F(\vec{o})$ is as defined in (2.27), $\vec{\lambda}_l$ is a vector of Lagrange multipliers, and $c(\vec{o})$ is a vector-valued function that when the constraints are met satisfies

$$c(\vec{o}) = \vec{0} \tag{2.31}$$

where $\vec{0}$ is a vector of zeros. $L(\vec{o}, \vec{\lambda}_l)$ is then minimized using any applicable method for unconstrained problems, and $\vec{\lambda}_l$ is selected to satisfy (2.31) exactly. If (2.31) is satisfied for multiple values of $\vec{\lambda}_l$, the $\vec{\lambda}_l$ that results in the minimum value of the objective function $F(\vec{o})$ is selected.

Considering the objective function of (2.27) with the associated constraint of (2.28), the constraint function $c(x)$ can be written

$$c(\vec{o}) = n - \vec{o}^H \vec{o} \tag{2.32}$$

where n is the number of elements in $\vec{o}$. In this case, $c(\vec{o})$ is a scalar-valued function, reducing the vector of multipliers to a single scalar $\vec{\lambda}_l = \lambda_l$. Applying this to (2.30) results in the expression

$$L(\vec{o}, \lambda_l) = \vec{o}^H \mathbf{M} \vec{o} - \lambda_l \vec{o}^H \vec{o} + n \tag{2.33}$$

Differentiating and setting this equal to zero results in

$$\mathbf{M}\vec{o} = \lambda_l \vec{o} \tag{2.34}$$

which is simply an eigenvalue problem. The eigenvector corresponding to the smallest eigenvalue then minimizes $F(\vec{o})$ subject to the constraints of (2.28).

With a sufficient description of the required statistical estimation methods complete, the next step in building an image reconstruction algorithm is development of a system model. However, before a statistically-based system model can be built, an understanding of the applicable physical processes is required, and will be developed in Chapters III and IV.

III. Optical Phenomena

Active imaging techniques, including those to be considered here, rely heavily on an understanding of certain fundamental properties of light including polarization, propagation and diffraction, and coherence. This chapter provides a brief overview of the optical phenomena applicable to the active imaging problem. The chapter is broken up into four main areas: the first covers polarization and depolarization phenomena, the second covers propagation and diffraction of light, the third introduces the coherence theory that will be used, and the fourth introduces laser speckle statistics that are integral to the active imaging problem.

3.1 Polarization and Associated Effects

As a form of electro-magnetic radiation, the propagation of light is governed by Maxwell's equations, which in differential form are given by [51]

$$\nabla \times \vec{E} + \frac{\partial \vec{B}}{\partial t} = 0 \tag{3.1a}$$

$$\nabla \times \vec{\mathcal{H}} - \frac{\partial \vec{D}}{\partial t} = J \tag{3.1b}$$

$$\nabla \cdot \vec{D} = \rho \tag{3.1c}$$

$$\nabla \cdot \vec{B} = 0 \tag{3.1d}$$

where $\vec{E}$ is the electric field vector, $\vec{B}$ is the magnetic flux density, $\vec{\mathcal{H}}$ is the magnetic field vector, $\vec{D}$ is the electric displacement, J is the free current density in the material, and ρ is the free charge density in the material. Given the vector nature of these equations, it's clear that any electro-magnetic field will have intrinsically vector qualities. Specifically, in free-space the wave vector, electric field vector, and the magnetic field vector are jointly orthogonal and form the basis of a right-handed coordinate system [8]. This characteristic translates into a "directional" property of the field that can be described by assigning polarization directions and writing the field in terms of the components along each of the assigned polarization directions.

3.1.1 Polarization. If the media through which the radiation is passing is homogeneous, linear, isotropic, and non-dispersive, the solution to (3.1) will satisfy the pair of vector Helmholz equations given by

$$\nabla^2 \vec{E} - \frac{\tilde{n}^2}{c^2}\frac{\partial^2 \vec{E}}{\partial t^2} = 0 \tag{3.2a}$$

$$\nabla^2 \vec{\mathcal{H}} - \frac{\tilde{n}^2}{c^2}\frac{\partial^2 \vec{\mathcal{H}}}{\partial t^2} = 0 \tag{3.2b}$$

where $\tilde{n}$ is the index of refraction and c is the speed of light. Because both $\vec{E}$ and $\vec{\mathcal{H}}$ obey the same vector wave equation, an identical scalar wave equation can be written to completely describe each vector component [25]

$$\nabla^2 U - \frac{\tilde{n}^2}{c^2}\frac{\partial^2 U}{\partial t^2} \tag{3.3}$$

where U can represent any of the field components. This implies that as long as the assumptions of a homogeneous, isotropic, linear, non-dispersive media are accurate, the field can be described by considering each component independently using scalar theory.

One consequence of this property is that specification of the electric field in free space can be used to completely describe the magnetic field. Therefore, only the electric field will be considered. Defining the polarization directions of a field propagating in the z direction to be along x and y, the electric field can be written

$$\vec{E} = E_x\hat{\imath} + E_y\hat{\jmath} \tag{3.4}$$

where $\hat{\imath}$ and $\hat{\jmath}$ are unit vectors along the x and y directions respectively. E_x and E_y are the field components projected onto $\hat{\imath}$ and $\hat{\jmath}$ given by

$$\begin{aligned} E_x &= A_1 \cos(\omega t - kz + \theta_1) \\ E_y &= A_2 \cos(\omega t - kz + \theta_2) \end{aligned} \tag{3.5}$$

where A_1 and A_2 are real scalar amplitudes, k is the wavenumber, ω is the angular frequency, and θ_1 and θ_2 are arbitrary phases in the interval $(-\pi, \pi]$.

To suppress the time and z dependence of (3.5), they are written in the form

$$\begin{aligned}\frac{E_x}{A_1} &= \cos(\omega t - kz)\cos(\theta_1) - \sin(\omega t - kz)\sin(\theta_1)\\ \frac{E_y}{A_2} &= \cos(\omega t - kz)\cos(\theta_2) - \sin(\omega t - kz)\sin(\theta_2)\end{aligned} \tag{3.6}$$

leading to

$$\begin{aligned}\frac{E_x}{A_1}\sin(\theta_2) - \frac{E_y}{A_2}\sin(\theta_1) &= \cos(\omega t - kz)\sin(\theta_2 - \theta_1)\\ \frac{E_x}{A_1}\cos(\theta_2) - \frac{E_y}{A_2}\cos(\theta_1) &= \sin(\omega t - kz)\sin(\theta_2 - \theta_1)\end{aligned} \tag{3.7}$$

which are then squared and added to produce

$$\left(\frac{E_x}{A_1}\right)^2 + \left(\frac{E_y}{A_2}\right)^2 - 2\frac{E_x E_y}{A_1 A_2}\cos(\theta_{12}) = \sin^2(\theta_{12}) \tag{3.8}$$

where $\theta_{12}= \theta_2 - \theta_1$. Equation (3.8) describes an ellipse that has been rotated in the plane by an angle Ψ, given by

$$\Psi = \frac{2E_x E_y}{E_x^2 + E_y^2}\cos\theta_{12} \tag{3.9}$$

and is known as the polarization ellipse [8], illustrated in Fig. 3.1.

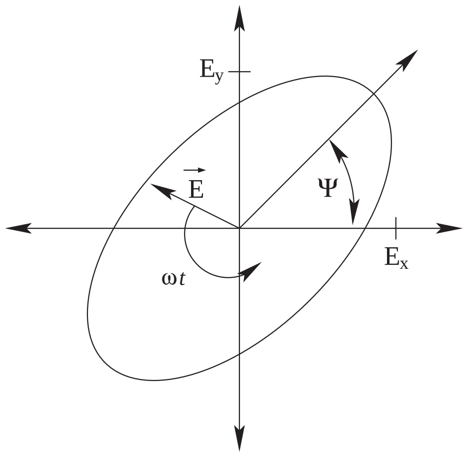


Figure 3.1: The polarization ellipse describes the temporal evolution and polarization state for elliptically polarized radiation. The eccentricity and angle describe the relationship between the $x-$ and $y-$polarized fields. The tip of the electric field vector traces the ellipse with an angular frequency of ω.

If the time-dependent nature of the field is again considered, it can be seen that the tip of the electric field vector traces the ellipse, rotating with an angular frequency of ω and the direction or sense of rotation depending on θ_{12} [8]. For $0 < \theta_{12} < \pi$, the electric field vector would appear to rotate in a clockwise fashion when viewing the (x, y) plane from a point on the positive z axis. By convention in the optical community this sense of rotation is called a right-handed polarization. When $\pi < \theta_{12} < 2\pi$ the electric field vector would appear to rotate in a counter-clockwise manner and be termed a left-handed polarization. It should be noted that these conventions for right- and left-handed polarizations are reversed from those normally used by the RF-microwave community.

One special case of interest is when the polarization state randomly changes on a time scale that is fast compared with the time-resolution of the detection system, or when the field is composed of a superposition of many randomly polarized components. This is referred to as unpolarized radiation, and is characterized by a lack of observable "preferred direction."

3.1.2 Stokes Parameters. With the polarization ellipse and sense of rotation specified, the description of a fully polarized wave of frequency ω traveling through free-space along the z direction is complete. However, the high frequencies and short wavelengths of light make it impossible to directly measure the amplitude and phase of the incident field. Conventional detection schemes rely on detecting the incident intensity/photon flux. Additionally, light that is only partially polarized needs additional information to satisfactorily describe it.

One scheme that adequately deals with these complications is the characterization of the light in terms of the Stokes vector $\vec{S}$, defined by [8]

$$\vec{S} = [S_0, S_1, S_2, S_3] \tag{3.10}$$

where

$$\begin{aligned} S_0 &= A_1^2 + A_2^2 \\ S_1 &= A_1^2 - A_2^2 \\ S_2 &= 2A_1A_2\cos(\theta_{12}) \\ S_3 &= 2A_1A_2\sin(\theta_{12}) \end{aligned} \tag{3.11}$$

Using a small amount of mental gymnastics, algebra, and intuition, it can be seen that S_0 is proportional to the total intensity of the light, S_1 is the predominance of the x polarization over the y polarization, S_2 is the predominance of left-45^o linear over right-45^o linear polarization, and S_3 is the predominance of right-circular over left-circular polarization. With the aid of linear polarizers and a wave plate, all of these quantities can be directly measured using conventional detection methods.

It should be noted that the polarization state of a fully polarized wave is completely described by three parameters, A_1, A_2, and θ_{12}; and that the Stokes parameters provide four. In the event that the wave is fully polarized, $S_0^2 = S_1^2 + S_2^2 + S_3^2$ and one of the parameters is degenerate. However, when the wave is partially polarized, the relationship between S_0 and the remaining parameters is no longer valid. The extra information now available makes it possible to characterize partially polarized light as well.

Partially polarized light can be broken down into the sum of fully polarized and fully unpolarized components. In this case, the stokes parameters can be written as

$$\begin{aligned} \vec{S} &= \vec{S}_u + \vec{S}_p \\ \vec{S}_u &= [(1-\mathcal{P})S_0, 0, 0, 0] \\ \vec{S}_p &= [\mathcal{P}S_0, S_1, S_2, S_3] \end{aligned} \tag{3.12}$$

where the degree of polarization $\mathcal{P}$ is the ratio of the polarized intensity with the total intensity, and can be written

$$\mathcal{P} = \frac{S_1^2 + S_2^2 + S_3^2}{S_0^2} \tag{3.13}$$

3.1.3 Depolarization. Because one of the imaging techniques being considered relies heavily on the changes to the polarization state after reflection from a distant object, an area of particular interest is the variation induced in the polarization state by the reflecting object. The changes in the polarization state stem primarily from the same fundamental process, manifested in three forms: change after reflection from a smooth surface induced by the difference in material reflectivity for the two principle polarization components, depolarization due to multiple reflections, and depolarization due to scattering from rough surfaces [4].

The simplest depolarization mechanism, and the foundation for all the applicable depolarization mechanisms, is caused by the variation between reflection coefficients for the polarization components parallel (TM) and perpendicular (TE) to the plane of incidence. For reflection of a wave from an interface between two materials with (possibly) complex indices of refraction $\tilde{n}_1$ and $\tilde{n}_2$, the amplitude reflection coefficients are given by the Fresnel equations [8]

$$\begin{aligned} r_{TE} &= \frac{\tilde{n}_1 \cos\theta_i - \tilde{n}_2 \cos\theta_t}{\tilde{n}_1 \cos\theta_i + \tilde{n}_2 \cos\theta_t} \\ r_{TM} &= \frac{\tilde{n}_2 \cos\theta_i - \tilde{n}_1 \cos\theta_t}{\tilde{n}_2 \cos\theta_i + \tilde{n}_1 \cos\theta_t} \end{aligned} \tag{3.14}$$

where, θ_i is the angle between the incident propagation vector and the interface, and θ_t is given by the law of refraction such that

$$\tilde{n}_1 \sin\theta_i = \tilde{n}_2 \sin\theta_t \tag{3.15}$$

For the cases of interest, the reflecting interface is between free-space where $\tilde{n}_1 = 1$, and an arbitrary material where $|\tilde{n}_2| > 1$.

As typical examples, the amplitude reflection coefficients and reflection phases for $500nm$ radiation reflected from an air-fused silica interface with $\tilde{n}_1 = 1$ and $\tilde{n}_2 = 1.45$ [26], and an air-unoxidized aluminum interface with $\tilde{n}_1 = 1$ and $\tilde{n}_2 = 0.62(1 + j4.8)$ [38] are shown in Fig. 3.2 and Fig. 3.3. Unless the incident light is completely TM or TE polarized, the incident field will have components containing both TM and TE components. As shown in Fig. 3.2 and Fig. 3.3, the reflection coefficients differ between TM and TE polarizations.

The change in relative amplitudes for each polarization component after reflection result in different major and minor axes for the polarization ellipse, and the change in phase after reflection shifts its rotation angle.

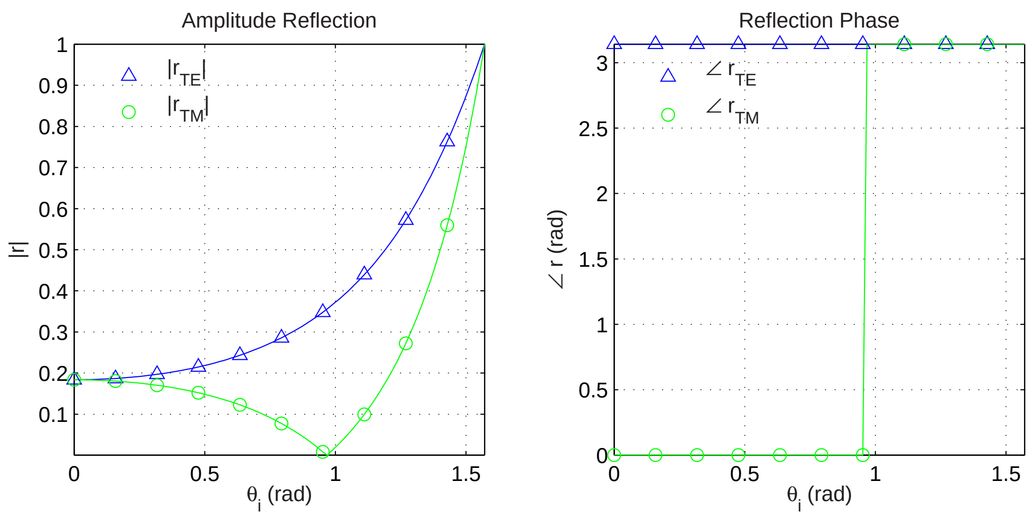


Figure 3.2: Reflection amplitude and phase for 500nm radiation reflected from an air ($\tilde{n}_1 = 1$) fused silica ($\tilde{n}_2 = 1.45$) interface. The difference between polarizations results in a change in the polarization state of the reflected radiation.

If the observed field is made up of contributions from reflections off of several different materials, or of reflections from differently oriented interfaces, the resulting field will be partially depolarized. The degree of depolarization is then dependent on the number of contributions and the degree of variance between the polarization of the contributions. When reflection depolarization is taken to the micro-scale to describe scattering off of a rough surface, the large number of contributions can significantly depolarize the field. For a complete discussion of depolarization via rough surface scattering, see chapter 8 of [4].

3.2 Scalar Diffraction Theory

An additional consequence of the scalar Helmholz equation of (3.3) is that provided the approximations of a linear, homogeneous, isotropic media are valid, diffraction and propagation of the vector fields can be described using scalar theory for each polarization component independently. Using this result, Green's theorem can be used to solve (3.3) for a given component U of the vector field using diverging spherical waves for the Green's

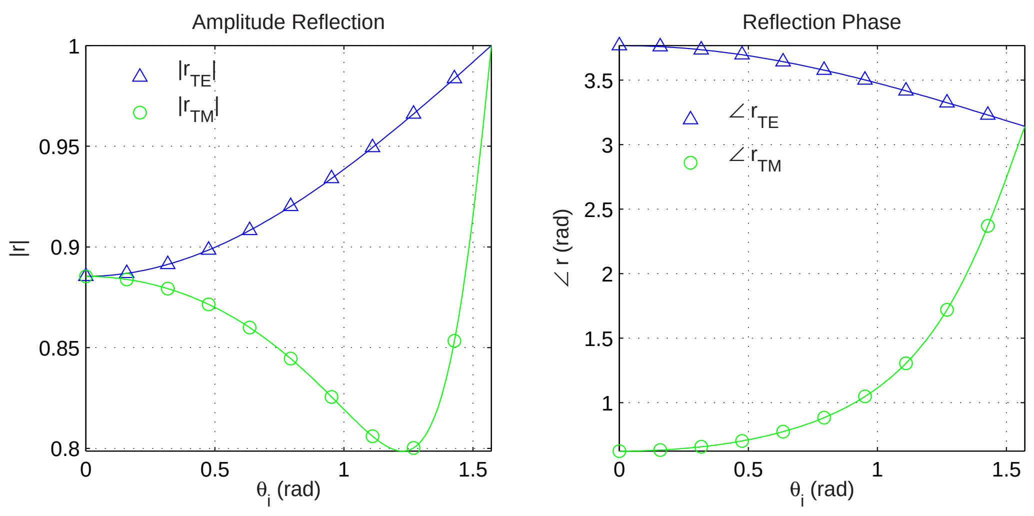


Figure 3.3: Reflection amplitude and phase for 500nm radiation reflected from an air ($\tilde{n}_1 = 1$) aluminum ($\tilde{n}_2 = 0.62(1 + j4.8)$) interface. The difference between polarizations results in a change in the polarization state of the reflected radiation

function [25]. Doing this leads to the most general expression describing scalar diffraction of monochromatic radiation between two planes given by

$$U(x, y) = \frac{1}{j\lambda} \iint_{-\infty}^{\infty} U(\xi, \eta) \frac{\exp(jkr_{01})}{r_{01}} \chi(\theta) \, \mathrm{d}\xi \mathrm{d}\eta \tag{3.16}$$

where $k = \frac{2\pi}{\lambda}$, $\chi(\cdot)$ is an "obliquity" factor (typically $\cos(\theta)$), and θ is the angle between a vector normal to the source wavefront at the point (ξ, η) and the vector of length $r_{01} = \sqrt{z^2 + (x - \xi)^2 + (y - \eta)^2}$ connecting the point (ξ, η) in the source plane with the point (x, y) in the observation plane [25].

Approximating $\chi(\theta) \approx \frac{z}{r_{01}}$, the diffraction equation becomes

$$U(x, y) = \frac{z}{j\lambda} \iint_{-\infty}^{\infty} U(\xi, \eta) \frac{\exp(jkr_{01})}{r_{01}^2} \mathrm{d}\xi \mathrm{d}\eta \tag{3.17}$$

Further simplification comes by using the binomial approximation $\sqrt{1+\epsilon} \approx 1 + \frac{\epsilon}{2}$ to approximate r_{01} in the exponential as

$$r_{01} = z\sqrt{1 + \left(\frac{x-\xi}{z}\right)^2 + \left(\frac{y-\eta}{z}\right)^2} \tag{3.18a}$$

$$\approx z + \frac{(x-\xi)^2 + (y-\eta)^2}{2z} \tag{3.18b}$$

and $r_{01} \approx z$ otherwise. The end result of these simplifications is the Fresnel diffraction integral given by

$$U(x,y) = \frac{e^{jkz}}{j\lambda z} e^{j\frac{k}{2z}(x^2+y^2)} \iint_{-\infty}^{\infty} \left\{ U(\xi,\eta) e^{j\frac{k}{2z}(\xi^2+\eta^2)} \right\} e^{-j\frac{2\pi}{\lambda z}(x\xi+y\eta)} \mathrm{d}\xi \mathrm{d}\eta \tag{3.19}$$

An additional simplification can be made when the distance between the source and observation planes is sufficiently large. Known as the Fraunhoffer diffraction integral, it's valid when the product of the object's maximum dimension and the wavelength is sufficiently small compared with the separation between the object and observation planes, i.e. when

$$z \gg \frac{1}{2} k(\xi_{max}^2 + \eta_{max}^2) \tag{3.20}$$

where ξ_{max} and η_{max} are the maximum linear dimensions of the object along the ξ and η axes respectively [25]. In this case, the quadratic phase term inside the integral is approximately unity over the region of interest and can be safely ignored, resulting in the Fraunhoffer diffraction equation given by

$$U(x,y) = \frac{e^{jkz}}{j\lambda z} e^{j\frac{k}{2z}(x^2+y^2)} \iint_{-\infty}^{\infty} U(\xi,\eta) e^{-j\frac{2\pi}{\lambda z}(x\xi+y\eta)} \mathrm{d}\xi \mathrm{d}\eta \tag{3.21}$$

which is valid for the cases of interest here. Ignoring the scaling terms outside the integral, this relationship can be recognized as a Fourier transform of the near-field distribution when the frequency variables u and v are evaluated at $u = \frac{x}{\lambda z}$ and $v = \frac{y}{\lambda z}$.

3.3 Coherence Theory

The basic premise of coherence theory is the characterization of the statistical properties of radiation. In the most general form, optical coherence is defined as the correlation between the field U at a point P_1 at time t and at a second point P_2 at time $t+\tau$, described by the mutual coherence function Γ_{12} [24], given by

$$\Gamma_{12}(P_1, P_2, \tau) = \mathrm{E}\left[U(P_1, t)U^*(P_2, t+\tau)\right] \tag{3.22}$$

From this general expression, temporal and spatial coherence effects can be described by setting the variables appropriately. For example, temporal coherence effects can be described using (3.22) and co-locating the observation points P_1 and P_2.

3.3.1 Mutual Intensity. The most significant coherence properties for the active imaging problem are the spatial coherence properties obtained when the correlation is evaluated with $\tau = 0$. In this case, the mutual coherence function is given a special designation called the mutual intensity J_{12}, defined by

$$J_{12}(P_1, P_2) \equiv \Gamma_{12}(P_1, P_2, 0) = \mathrm{E}\left[U(P_1, t)U^*(P_2, t)\right] \tag{3.23}$$

If $P_1 = P_2$, the mutual intensity $J_{12}(P_1, P_1)$ equals the average intensity in the observation plane $\mathrm{E}\left[I(P_1)\right]$.

3.3.2 Propagation of the Mutual Intensity. The specific character of the mutual intensity function for light reflected from a distant object is dependent on both the nature of the object being illuminated and the character of the illumination. Typically, the mutual intensity at the object is easily described, and the problem becomes finding the mutual intensity in the observation plane given the mutual intensity at the object.

For convenience, let $P_1 = (\xi_1, \eta_1)$ and $P_2 = (\xi_2, \eta_2)$ be points in the object plane, and let $Q_1 = (u_1, v_1)$ and $Q_2 = (u_2, v_2)$ be points in the observation plane. Equation (3.16) can be used to find the field at points Q_1 and Q_2 which can then be substituted into equation (3.22) to find the mutual intensity. Doing this, interchanging the order of integration and expectation, and expressing τ as a function of the path length difference from the source to points Q_2 and Q_2, results in an expression for the mutual intensity in the observation plane given by

$$J_{12}(Q_1, Q_2) = \iint_{-\infty}^{\infty} \iint_{-\infty}^{\infty} J_{12}(P_1, P_2) \exp\left[-j\frac{2\pi}{\lambda}(r_2 - r_1)\right] \frac{\chi(\theta_1)}{\lambda r_1} \frac{\chi(\theta_2)}{\lambda r_2} d\xi_1 d\eta_1 \mathrm{d}\xi_2 \mathrm{d}\eta_2 \tag{3.24}$$

where r_1 is the distance from the point Q_1 to P_1, r_2 is the distance from the point Q_2 to the point P_2, and θ_1 and θ_2 are the angles between r_1 and r_2 and the surface normal respectively [24].

3.4 Speckle Statistics

The speckled appearance of coherent light scattered from almost any surface is a well known phenomenon. It was recognized early on that speckle was due to interference in the detector plane [12]. When coherent light reflects from a rough surface, the differences in surface profile result in slight differences in propagation distances as shown in Fig. 3.4. These path-length differences are adequately represented mathematically by the addition of a random phase to the wave front. Propagation of the field with the addition of this phase results in the random interference patterns known as speckle. To understand the implications and uses of speckle, its statistical properties need to be understood.

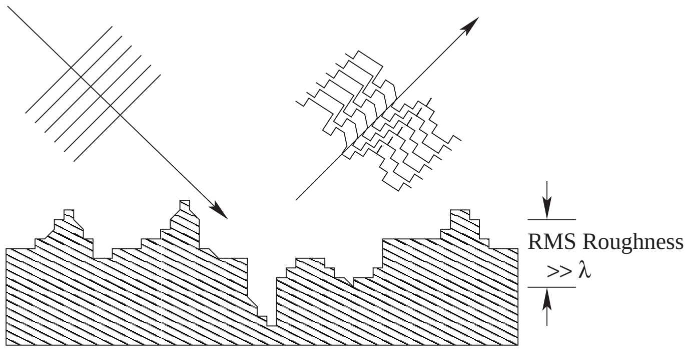


Figure 3.4: When coherent light reflects from a rough surface, the slight differences in path-length result in random phase-front variations that are manifest as speckle.

3.4.1 First-Order Speckle Statistics. The first property of interest for laser speckle is the statistical distribution of the intensity at a point in the observation plane. To do this, a rough object is defined such that the phase induced by surface roughness upon reflection is randomly distributed on the interval $(-\pi, \pi]$. Physically, this corresponds to an object with an RMS surface roughness of one wavelength or more, a condition that is satisfied for all but the most highly polished surfaces at optical wavelengths. When the assumption of a rough object is valid, the field in an observation plane some distance away is given by a superposition of contributions from the randomly phased fields in the object plane. In this case, the problem can be viewed as the sum of complex phasors, which in the limit of many scattering locations simplifies to complex circular Gaussian statistics given by [12]

$$p(A_r, A_i) = \frac{1}{2\pi\sigma^2} \exp\left\{-\frac{A_r^2 + A_i^2}{2\sigma^2}\right\} \tag{3.25}$$

where

$$A_r = \mathcal{Re}\{U\} \tag{3.26a}$$

$$A_i = \mathcal{Im}\{U\} \tag{3.26b}$$

are the real and imaginary parts of the complex field, σ^2 is given by

$$\sigma^2 = \frac{\mathrm{E}\left[I_o\right]}{2} \tag{3.27}$$

and I_o is the intensity reflected from the object.

Applying variable transformations to get the distribution in terms of intensity and phase, and integrating out the phase to get the marginal intensity distribution leads to an exponential distribution in intensity given by

$$p(I_p) = \begin{cases} \frac{1}{\mathrm{E}[I_p]} \exp\left(-\frac{I_p}{\mathrm{E}[I_p]}\right) & I_p \geq 0 \\ 0, & \text{else} \end{cases} \tag{3.28}$$

where I_p is the intensity in the far-field [24]. One implication of this distribution is that the standard deviation of the intensity is equal to the mean $\mathrm{E}\left[I_p\right]$.

3.4.2 Second-Order Speckle Statistics. An understanding of second-order speckle statistics begins with a description of the mutual intensity at the object. Assuming that the correlation length of the surface roughness is small compared with the size of the object, the mutual intensity in the object plane can be described by [24]

$$J_{12}(P_1, P_2) = \mathrm{E}\left[I_o\left(P_1\right)\right]\delta\left(P_1 - P_2\right) \tag{3.29}$$

where $\delta\left(P_1 - P_2\right)$ is the Dirac delta in Cartesian coordinates. If the object is uniformly illuminated, $\mathrm{E}\left[I_o\right]$ is proportional to the reflectivity of the object.

At this point (3.24) can be used to determine the mutual intensity in the observation plane. Given the geometry being considered and the form of the mutual intensity, several simplifications can be made. In the far-field, $\chi(\theta_0)$ and $\chi(\theta_1)$ are approximately 1, and $\frac{1}{r_1 r_2} \approx \frac{1}{z^2}$. Additionally, the standard paraxial approximations can be made for r_1 and r_2 in the exponential,

$$\begin{aligned} r_1 &= \sqrt{z^2 + \left(u_1 - \xi_1\right)^2 + \left(v_1 - \eta_1\right)^2} \\ &\approx z + \frac{\left(u_1 - \xi_1\right)^2 + \left(v_1 - \eta_1\right)^2}{2z^2} \\ r_2 &= \sqrt{z^2 + \left(u_2 - \xi_2\right)^2 + \left(v_2 - \eta_2\right)^2} \\ &\approx z + \frac{\left(u_2 - \xi_2\right)^2 + \left(v_2 - \eta_2\right)^2}{2z^2} \end{aligned} \tag{3.30}$$

Applying these approximations and employing the sifting property of $\delta(P_1 - P_2)$ results in the final expression for the mutual intensity in the observation plane.

$$J_{12}\left(\Delta u, \Delta v\right) = \frac{\gamma e^{-j\psi_p}}{\left(\lambda z\right)^2} \iint_{-\infty}^{\infty} \mathrm{E}\left[I(\xi, \eta)\right] \exp\left\{-j\frac{2\pi}{\lambda z}\left[\left(\Delta u \xi + \Delta v \eta\right)\right]\right\} \mathrm{d}\xi \mathrm{d}\eta \tag{3.31}$$

where $\Delta u = u_2 - u_1$, $\Delta v = v_2 - v_1$, $\psi_p = \frac{\pi}{\lambda z}\left[\left(u_1^2 + v_1^2\right) - \left(u_1^2 + v_1^2\right)\right]$, and γ is a scaling constant. Although derived for coherent illumination scattered from a rough object, (3.31) is functionally identical to the Van Cittert-Zernike theorem for incoherent light [24]. At this point it's also worth noting that the ensemble averaged intensity in the far-field is given by

$$\mathrm{E}\left[I_p(u, v)\right] = J_{12}(\Delta u = 0, \Delta v = 0) \tag{3.32}$$

which is constant across the observation plane.

One characteristic of interest is the pupil-plane Power Spectral Density (PSD) of the speckled intensity. According to the Weiner-Khintchine theorem, the PSD and autocorrelation of the speckle intensity are related through a Fourier transform relationship [46] such that

$$\Phi_{PP}(x,y) = \mathcal{F}\{\Gamma_{PP}(\Delta u, \Delta v)\} \tag{3.33}$$

where $\mathcal{F}\{\cdot\}$ is the Fourier transform, Φ_{PP} is the speckle PSD, and $\Gamma_{PP}(\Delta u, \Delta v)$ is the speckle autocorrelation for point separations Δu and Δv.

Because the underlying fields obey complex Gaussian statistics, the autocorrelation of the field amplitude and the intensity are related through the complex Gaussian moment theorem [24] according to

$$\begin{aligned}\Gamma_{PP} &= \mathrm{E}\left[I_p(u_1,v_1)\right]\mathrm{E}\left[I_p(u_2,v_2)\right] + |J_{12}(u_1,v_1,u_2,v_2)|^2 \\ &= \mathrm{E}\left[I_p\right]^2\left[1 + |\mu_c|^2\right]\end{aligned} \tag{3.34}$$

where the complex coherence coefficient μ_c is given by

$$\mu_c(u_1,v_1,u_2,v_2) = \frac{J_{12}\left(u_1,v_1,u_2,v_2\right)}{\left[J_{12}\left(u_1,v_1,u_1,v_1\right)J_{12}\left(u_2,v_2,u_2,v_2\right)\right]^{1/2}} \tag{3.35}$$

Recognizing from (3.31) and (3.35) that μ_c is a scaled Fourier transform of the object intensity, the normalized near-field intensity PSD $\hat{\Phi}_{nf}$ can be written

$$\hat{\Phi}_{nf}\left(u,v\right) = \left|\mu_c\left(\Delta u, \Delta v\right)\right|^2 \tag{3.36}$$

Substituting (3.36) into (3.34) and applying the Weiner-Khintchine theorem, the far-field intensity (speckle) power spectral density can be written as

$$\Phi_{PP}(x,y) = \mathrm{E}\left[I_p\right]^2\left[\delta\left(x,y\right) + \mathcal{F}\{\hat{\Phi}_{nf}\left(u,v\right)\}\right] \tag{3.37}$$

Because the near field intensity $\mathrm{E}\left[I_p\right]$ is real and non-negative, $\mathcal{F}\{\hat{\Phi}_{nf}\} = \mathcal{F}^{-1}\{\hat{\Phi}_{nf}\}$, which can be used with (3.33) to give

$$\Phi_{PP}(u,v) = \mathrm{E}\left[I_p\right]^2\left[\delta(u,v) + \Gamma_n\right] \tag{3.38}$$

where Γ_n is the normalized autocorrelation of the near-field intensity given by

$$\Gamma_n(x,y) = \gamma \mathbf{o}(x,y) \circledast \mathbf{o}(x,y) \tag{3.39}$$

and where $\circledast$ is a two-dimensional correlation.

Additional statistical properties of the far-field pattern can be found provided the joint probability distribution function for the applicable quantity is known. Of particular interest are the joint distributions for the complex amplitude, intensity, and phase. As stated previously, the field in the detector plane follows joint complex circular Gaussian statistics [12], resulting in a probability distribution function given by

$$P(U) = \frac{1}{(2\pi)^{n/2}\sqrt{|\mathbf{\Gamma}_U|}} \exp\left\{-\frac{1}{2}U^H \mathbf{\Gamma}_U^{-1} U\right\} \tag{3.40}$$

where U is a vector of complex amplitudes, H is the Hermitian transpose, and $\mathbf{\Gamma}_U$ is the associated correlation matrix given by $\mathbf{\Gamma}_U = \mathrm{E}\left[UU^H\right]$ [35]. From this, the joint statistics of the amplitude, phase, and intensity can be derived.

To specify the second-order density function of the amplitude and phase, the complex amplitudes U_1 and U_2 at two distinct points are written in as $U_1 = U_1^{(r)} + iU_1^{(i)}$ and $U_2 = U_2^{(r)} + iU_2^{(i)}$. The joint PDF of U_1 and U_2 is then given by

$$p(U_1^{(r)}, U_1^{(i)}, U_2^{(r)}, U_2^{(i)}) = \frac{\exp\left[-\frac{|U_1|^2+|U_2|^2-\mu_c U_1 U_2^* - \mu_c^* U_1^* U_2}{2\sigma^2\left(1-|\mu_c|^2\right)}\right]}{4\pi^2\sigma^4\left(1-|\mu_c|^2\right)} \tag{3.41}$$

where μ_c is the complex coherence factor from (3.35), and it has been assumed that $\mathrm{E}\left[I_p\left(u_1,v_1\right)\right] = \mathrm{E}\left[I_p\left(u_2,v_2\right)\right] = 2\sigma^2$ [12].

The joint statistics of the intensity and phase are then related to (3.41) through the variable transformations

$$\begin{aligned} U_1^{(r)} &= \sqrt{I_1}\cos(\theta_1) \\ U_2^{(r)} &= \sqrt{I_2}\cos(\theta_2) \\ U_1^{(i)} &= \sqrt{I_1}\sin(\theta_1) \\ U_2^{(i)} &= \sqrt{I_2}\sin(\theta_2) \end{aligned} \tag{3.42}$$

Noting that the Jacobian of this transformation is 1/4, and defining $\mu = |\mu_c|$ and $\psi = \angle\mu_c$, the resulting joint PDF for the intensity and phase is given by

$$p_{I,\theta}(I_1, I_2, \theta_1, \theta_2) = \frac{\exp\left[-\frac{I_1+I_2-2\sqrt{I_1 I_2}\mu\cos(\theta_1-\theta_2+\psi)}{2\sigma^2(1-\mu^2)}\right]}{16\pi^2\sigma^4(1-\mu^2)} \tag{3.43}$$

which can be integrated to find the marginal joint densities for the intensity and phase. From [12], the joint density for the intensity is given by

$$p_I(I_1, I_2) = \frac{\exp\left[-\frac{I_1+I_2}{\mathrm{E}[I](1-\mu^2)}\right]}{\mathrm{E}\left[I\right]^2(1-\mu^2)} I_0\left(\frac{2\mu\sqrt{I_1 I_2}}{\mathrm{E}\left[I\right](1-\mu^2)}\right) \tag{3.44}$$

where I_0 is a modified Bessel function of the first kind, zero order. Also from [12], the joint density of the phase is given by

$$p_\theta(\theta_1, \theta_2) = \frac{1-\mu^2}{4\pi^2(1-\beta^2)^{3/2}}\left(\beta\sin^{-1}\beta + \frac{\pi\beta}{2} + \sqrt{1-\beta^2}\right) \tag{3.45}$$

where $\beta = \mu\cos(\theta_2 - \theta_1 + \psi)$, and θ_1 and θ_2 are in the interval $(-\pi, \pi)$.

This chapter introduced the basic building blocks of polarization, scalar diffraction, and the statistical nature of light. With these tools in hand, the next step is to use these tools to build up a system model for use in algorithm development.

IV. Conventional Imaging and Atmospheric Turbulence

With an understanding of scalar diffraction and other optical phenomenon in hand, the next step required to begin developing a reconstruction algorithm is forming a model for the imaging system that includes the effects of aberrations on image formation. Beginning with the diffraction theory developed in chapter III, a model for image formation using both coherent and incoherent illumination is developed. The effects of system aberrations and the usage of Zernike coefficients for describing them are presented. Because the primary aberrations of interest are caused by atmospheric turbulence, a statistical description of the turbulence is developed and again described in terms of Zernike modes. A method for simulating the effects of turbulence using Zernike modes is presented. Finally, conventional phase diversity techniques for reconstructing images from aberrated data are presented for both Gaussian and Poisson dominated noise statistics.

4.1 Basic System Model

Before diffraction theory can be used to describe the imaging process, a model for the imaging system through which the light will propagate must be developed. For an achromatic imaging system, even complex configurations can be modeled using a single thin lens located in the pupil-plane of the system as shown in Fig. 4.1 [8]. The object is described by the field $U_o(\xi, \eta)$ in a plane a distance z_o from the imaging system pupil, the effective lens and pupil-plane field $U_p(u, v)$ are located in the pupil plane of the imaging system, and the image-plane field $U_i(x, y)$ is in a plane a distance z_i behind the pupil. Diffraction theory as outlined in chapter III is used to translate the field between planes. The effect of the lens can be described by multiplying the incident field with a real pupil function and complex phase such that the field just after the lens is given by

$$U_p'(u, v) = U_p(u, v) P(u, v) e^{-j \frac{\pi}{\lambda f_l}\left(u^2+v^2\right)} \tag{4.1}$$

where f_l is the focal length of the effective lens and $P(u, v)$ is the transmission function of the imaging system's pupil. Frequently, $P(u, v)$ is a clear circular pupil described by

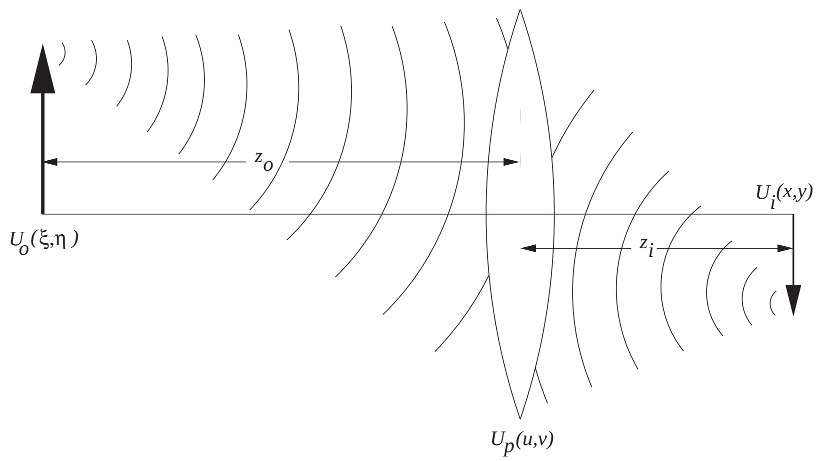


Figure 4.1: Simplified imaging system. Imaging systems can generaly be described by a simple lens with a diameter equal to the imaging system pupil and a focal length equal to the focal length of the imaging system.

$$P(u,v) = \begin{cases} 1, & u^2+v^2 \le 1 \\ 0, & \text{else} \end{cases} \tag{4.2}$$

where the coordinate system has been scaled such that the pupil-diameter is one to simplify later work.

4.1.1 Coherent Imaging. Because diffraction is a linear operation, the effect of diffraction through a linear system can be described in general by the superposition integral

$$U_i(x,y) = \iint_{-\infty}^{\infty} h(x,y;\xi,\eta) U_o(\xi,\eta) \mathrm{d}\xi \mathrm{d}\eta \tag{4.3}$$

where the impulse response, $h(x,y;\xi,\eta)$, is the image-plane field that would result from a point source located at the point (ξ,η) in the object plane [25]. Using a point source object to obtain h, the field in the pupil plane is simply a diverging spherical wave; which under the paraxial approximation is given by

$$U_p(u,v) = \frac{1}{j\lambda z_o} \exp\left\{ j\frac{\pi}{\lambda z_o}\left[(u-\xi)^2 + (v-\eta)^2\right]\right\} \tag{4.4}$$

Applying the lens and using Fresnel diffraction to propagate the pupil plane field to the image plane, the resulting impulse response for the imaging system is given by

$$\begin{aligned}h(x,y;\xi,\eta) = \frac{1}{\lambda^2 z_i z_o} \exp\left[j\frac{\pi}{z_o}\left(\xi^2,\eta^2\right)\right] \exp\left[j\frac{\pi}{z_i}\left(x^2,y^2\right)\right] \\ \iint_{-\infty}^{\infty} P(u,v) \exp\left[j\frac{\pi}{\lambda}\left(\frac{1}{z_o}+\frac{1}{z_i}-\frac{1}{f_l}\right)\left(u^2+v^2\right)\right] \\ \exp\left\{-j\frac{2\pi}{\lambda}\left[\left(\frac{\xi}{z_o}+\frac{x}{z_i}\right)u+\left(\frac{\eta}{z_o}+\frac{y}{z_i}\right)v\right]\right\} \mathrm{d}u\mathrm{d}v \quad (4.5)\end{aligned}$$

Considering the quadratic phase term within the integral, when the Gauss lens law is satisfied, i.e. when

$$\frac{1}{f_l} = \frac{1}{z_o} + \frac{1}{z_i} \tag{4.6}$$

the phase term vanishes, leaving only the phase terms outside the integral to deal with. For the term involving image-plane coordinates x and y, if the intensity is the final quantity of interest this term can be neglected outright. The term involving the object-plane coordinates ξ and η isn't so easily neglected; however, this term can also be approximated as unity under certain assumptions that will be invoked here [25].

To further simplify the impulse response, the system magnification M is defined as

$$M = -\frac{z_i}{z_o} \tag{4.7}$$

and the scale constants outside the integral are dropped, resulting in

$$h\left(x-\xi,y-\eta\right) = \iint_{-\infty}^{\infty} P(u,v) \exp\left\{-j\frac{2\pi}{\lambda z_i}(x-M\xi)\,u+(y-M\eta)\,v\right\} \mathrm{d}u\mathrm{d}v \tag{4.8}$$

which when $M = -1$, is the impulse response function for a linear, shift-invariant system that can be recognized as a scaled Fourier transform of the imaging system pupil-function. Applying this impulse-response function to (4.3) results in the final model for the field in the image plane

$$U_i(x,y) = \iint_{-\infty}^{\infty} U_o\left(\xi,\eta\right) h\left(x-\xi,y-\eta\right) \mathrm{d}\xi\mathrm{d}\eta \tag{4.9}$$

The instantaneous intensity, I_i, in the image plane is then given by

$$I_i(x,y) = \left| \iint_{-\infty}^{\infty} U_o(\xi,\eta)\, h(x-\xi, y-\eta)\, \mathrm{d}\xi \mathrm{d}\eta \right|^2 \tag{4.10}$$

Because all detectors integrate for some period of time, the instantaneous intensity must be averaged [24], resulting in

$$\mathrm{E}\left[I_i(x,y)\right] = \iint_{-\infty}^{\infty} \mathrm{d}\xi_1 \mathrm{d}\eta_1 \iint_{-\infty}^{\infty} \mathrm{d}\xi_2 \mathrm{d}\eta_2 h(x-\xi_1, y-\eta_2) h^*(x-\xi_2, y-\eta_2)\, \mathrm{E}\left[U_o(\xi_1,\eta_1) U_o^*(\xi_2,\eta_2)\right] \tag{4.11}$$

For coherent illumination, the fields are deterministic and the expectation can be dropped. The integrals can then be separated such that the final expression for the coherent image reverts to the instantaneous intensity given by (4.10).

4.1.2 Incoherent Imaging. When the illumination is incoherent, the mutual intensity is adequately described by

$$\mathrm{E}\left[U_o(\xi_1,\eta_1) U_o(\xi_2,\eta_2)\right] = \delta(\xi_1-\xi_2, \eta_1-\eta_2) I_o(\xi_1,\eta_1) \tag{4.12}$$

where $I_o(\xi_1,\eta_1)$ is the intensity in the object plane. Substituting this into (4.11) and employing the sifting property of the δ-function results in an expression for the intensity in the image plane under incoherent illumination given by

$$I_i(x,y) = \iint_{-\infty}^{\infty} s(x-\xi, y-\eta) I_o(\xi,\eta) \mathrm{d}\xi \mathrm{d}\eta \tag{4.13}$$

where the point-spread function s is given by

$$s(x,y) = |h(x,y)|^2 \tag{4.14}$$

4.1.3 Aberrations and Zernike Polynomials. In general, the imaging system isn't completely described by a perfect thin lens, and consequently suffers from some form of aberration which serves to widen the point-spread function and blur the resulting image. The simplest approach to describing the effects of aberrations is to add their contributions to the system model's pupil function. The pupil function is allowed to be complex and defined as

$$P'(u,v) = P(u,v)e^{j\phi(u,v)} \tag{4.15}$$

where any obstructions or amplitude transmission variations are included in $P(u,v)$, and phase aberrations such as defocus, coma, astigmatism, etc... are included in the phase term $\phi(u,v)$.

To simplify analysis, the phase aberration $\phi(u,v)$ is frequently projected against a basis set to parameterize the aberrations. Because they coincide with the classical aberrations and tend to efficiently represent the aberration with relatively few terms, Zernike polynomials (modes) are often used as the basis set. For this work, the Zernike modes are as defined and indexed by Noll [37] such that

$$Z_j(r,\theta) = \begin{cases} \sqrt{n+1}R_n^m(r)\sqrt{2}\cos(m\theta) & m \neq 0 \quad \text{and} \quad j \text{ even} \\ \sqrt{n+1}R_n^m(r)\sqrt{2}\sin(m\theta) & m \neq 0 \quad \text{and} \quad j \text{ odd} \\ \sqrt{n+1}R_n^0(r) & m = 0 \end{cases} \tag{4.16}$$

where

$$R_n^m(r) = \sum_{s=0}^{\frac{n-m}{2}} \frac{(-1)^s(n-s)!}{s!\left[\frac{n+m}{2}-s\right]!\left[\frac{n-m}{2}-s\right]!} r^{n-2s} \tag{4.17}$$

$r = \sqrt{u^2+v^2}$, $\theta = \tan^{-1}(v/u)$, and $0 \leq r \leq 1$, $0 \leq \theta < 2\pi$. The sequence of modes defined by the mode index j is such that incrementing j increments the angular mode number m until it's equal to the radial mode number n, at which point n is incremented and m reset to zero. The progression in mode number for modes 1 through 16 is shown in Table 4.1 along with the corresponding common name (where applicable) for the particular aberration represented by a given mode.

4.2 Atmospheric Turbulence

Quite frequently the limiting system aberration is the blurring caused by atmospheric turbulence. This effect can be easily seen in the "twinkling" of stars and results in much worse resolution than would be the case if the atmosphere were not a contributing factor.

Table 4.1: First 16 Noll-indexed Zernike modes with the associated common aberration names [41]

j	n	m	$Z_j(r,\theta)$	aberration
1	0	0	1	piston
2	1	1	$2r\cos(\theta)$	tilt (x)
3	1	1	$2r\sin(\theta)$	tilt (y)
4	2	0	$\sqrt{3}\left(2r^2-1\right)$	defocus
5	2	2	$\sqrt{6}r^2\sin(2\theta)$	astigmatism (x)
6	2	2	$\sqrt{6}r^2\cos(2\theta)$	astigmatism (y)
7	3	1	$\sqrt{8}\left(3r^3-2r\right)\sin(\theta)$	coma (x)
8	3	1	$\sqrt{8}\left(3r^3-2r\right)\cos(\theta)$	coma (y)
9	3	3	$\sqrt{8}r^3\sin(3\theta)$	
10	3	3	$\sqrt{8}r^3\cos(3\theta)$	
11	4	0	$\sqrt{5}\left(6r^4-6r^2+1\right)$	3^{rd} order spherical
12	4	2	$\sqrt{10}\left(4r^4-3r^2\right)\cos(2\theta)$	
13	4	2	$\sqrt{10}\left(4r^4-3r^2\right)\sin(2\theta)$	
14	4	4	$\sqrt{10}r^4\cos(4\theta)$	
15	4	4	$\sqrt{10}r^4\cos(4\theta)$	
16	5	1	$\sqrt{12}\left(10r^5-12r^3+3r\right)\cos(\theta)$	

4.2.1 Kolmogorov Turbulence Theory. A complete description of turbulence theory would require an entire book, and multiple sources are available for more detailed descriptions (see for example [2,41]). Only a brief overview of the relevant results will be presented here.

Using dimensional analysis and physical insight, Kolmogorov developed a power spectrum for velocity fluctuations in a turbulent medium that was extended to describe the power spectrum for index of refraction variations, Φ_n, given by [2]

$$\Phi_n(\kappa) = 0.033 C_n^2 \kappa^{-\frac{11}{3}}, \quad \frac{1}{L_o} \ll \kappa \ll \frac{1}{l_o} \tag{4.18}$$

where $\kappa = 2\pi/\sqrt{u^2+v^2}$ is the spatial wavenumber, L_o is the turbulence outer scale which corresponds roughly to the size of the largest turbulent area that can be analyzed independently from the parent flow, l_o is the turbulence inner scale which is roughly the size of the smallest intact regions of turbulent flow, and C_n^2 is the index of refraction structure function constant characterizing turbulence strength. In general, C_n^2 will be a function of altitude, but a path-averaged value will be used here.

To transition from index of refraction fluctuations to phase fluctuations, the aberration phase due to propagating a distance L along the z axis through the turbulent flow is found by adding up contributions from the varying optical path lengths caused by the changes in refractive index according to

$$\phi\left(u,v\right)=k\int_0^L \mathrm{d}z\;\tilde{n}_1\left(u,v,z\right) \tag{4.19}$$

where L is the thickness of the turbulent layer, the index of refraction for the flow is defined as

$$\tilde{n}\left(u,v,z\right)=\tilde{n}_0+\tilde{n}_1\left(u,v,z\right) \tag{4.20}$$

$\tilde{n}_0$ is the mean index value, $\tilde{n}_1$ is the small perturbation caused by turbulence, and L is the distance along the propagation path. Assuming phase fluctuations are sufficiently small to approximate their cumulative effect in a single layer, the field resulting from propagating a unit-amplitude incident plane wave through the turbulence would then be given by

$$U_t\left(u,v\right)=\exp\left[j\phi\left(u,v\right)\right] \tag{4.21}$$

Assuming wide-sense stationarity for the turbulence, the spatial correlation function for the resulting field is then given by

$$\begin{aligned}\Gamma_\phi(\Delta u,\Delta v)=&\,\mathrm{E}\left[\exp\left[j\phi(u,v)\right]\exp\left[-j\phi(u-\Delta u,v-\Delta v)\right]\right] && (4.22)\\ =&\,\mathrm{E}\left[\exp\left[jk\left(\int_0^L \mathrm{d}z\left[\tilde{n}_1(u,v,z)-\tilde{n}_1(u-\Delta u,v-\Delta v,z)\right]\right)\right]\right] && (4.23)\end{aligned}$$

Assuming the index fluctuations follow Gaussian statistics, (4.22) can be simplified using

$$\mathrm{E}\left[\exp\left(a+jb\right)\right]=\exp\left[\left(\mu_a+j\mu_b\right)\right]\exp\left[\frac{1}{2}\left(\sigma_a^2-j2\rho\sigma_a\sigma_b-\sigma_b^2\right)\right] \tag{4.24}$$

where

$$\rho=\frac{\mathrm{E}\left[(a-\mu_a)(b-\mu_b)\right]}{\sqrt{\sigma_a^2\sigma_b^2}} \tag{4.25}$$

and where μ_a and μ_b are the mean values of a and b respectively. Applying (4.24) to (4.22) and recognizing that the real part of the exponent, a, is zero everywhere, $\mu_a=\sigma_a=0$ [41].

The imaginary part, b, of (4.24) is seen to be only a function of $\tilde{n}_1$, which from the definition of (4.20) is zero mean. Applying this, $\mu_b = 0$, and the resulting phase correlation function, Γ_ϕ, is given by

$$\Gamma_\phi(\Delta u, \Delta v) = \exp\left[-\frac{1}{2}k^2\,\mathrm{E}\left[\left(\int_0^L \mathrm{d}z\,[\tilde{n}_1(u,v,z) - \tilde{n}_1(u-\Delta u, v-\Delta v, z)]\right)^2\right]\right] \tag{4.26}$$

$$= \exp\left[-\frac{1}{2}D_\phi(\Delta u, \Delta v)\right] \tag{4.27}$$

where the phase structure function $D_\phi(\Delta u, \Delta v)$ is defined

$$D_\phi(\Delta u, \Delta v) = k^2\,\mathrm{E}\left[\left(\int_0^L \mathrm{d}z\,[\tilde{n}_1(u,v,z) - \tilde{n}_1(u-\Delta u, v-\Delta v, z)]\right)^2\right] \tag{4.28}$$

$$= \mathrm{E}\left[(\phi(u,v) - \phi(u-\Delta u, v-\Delta v))^2\right] \tag{4.29}$$

At this point it's clear that describing the structure function is sufficient to completely characterize the statistics of the phase.

Defining

$$r = \sqrt{\Delta u^2 + \Delta v^2} \tag{4.30}$$

and assuming Kolmogorov statistics, the phase structure function is given by [41]

$$D_\phi(\Delta u, \Delta v) = 2.91k^2LC_n^2r^{\frac{5}{3}} \tag{4.31}$$

Defining the Fried parameter r_0 [41]

$$r_0 = 0.185\left[\frac{4\pi^2}{k^2C_n^2L}\right]^{\frac{3}{5}} \tag{4.32}$$

the structure function reduces to

$$D_\phi(\Delta u, v) = 6.88\left(\frac{r}{r_0}\right)^{\frac{5}{3}} \tag{4.33}$$

4.2.2 Representing and Simulating Turbulence with Zernike Polynomials. As with other aberrations, it's frequently convenient to describe turbulence effects in terms of Zernike modes. Using Kolmogorov statistics as given above, Noll wrote the power spectral density for phase fluctuations as [37]

$$\Phi_\phi(\kappa) = 0.023 r_0^{5/3} \kappa^{-11/3} \tag{4.34}$$

Using this spectrum, Noll showed that the Zernike-mode expansion coefficients were Gaussian distributed with zero mean and a covariance matrix, $\mathbf{\Gamma}_z$, defined by

$$(\mathbf{\Gamma}_z)_{j,j'} = \frac{-0.145}{\pi}\left(\frac{D}{r_0}\right)^{\frac{5}{3}} \sqrt{(n+1)(n'+1)}(-\delta_{mm'})^{\frac{n+n'-2m}{2}} \int_0^\infty \mathrm{d}k\, k^{-\frac{5}{3}} \frac{J_{n+1}(2\pi k) J_{n'+1}(2\pi k)}{k^2} \tag{4.35}$$

where D is the diameter of the imaging system pupil, α_j is the coefficient for the j^{th} mode, n and m are the radial and angular mode numbers for the j^{th} mode, and $\delta_{mm'}$ is the Kronecker delta. The integral in (4.35) can be analytically evaluated, and is given by [37]

$$\int_0^\infty \mathrm{d}k\, k^{-\frac{5}{3}} \frac{J_{n+1}(2\pi k) J_{n'+1}(2\pi k)}{k^2} = \frac{\Gamma\left(\frac{14}{3}\right)\Gamma\left[\frac{(n+n')}{2} - \frac{23}{6}\right]}{2^{14/3}\Gamma\left[\frac{(n-n')}{2} + \frac{17}{6}\right]\Gamma\left[\frac{(n+n')}{2} + \frac{23}{6}\right]} \tag{4.36}$$

One notable property of (4.35) is that the only dependence on the atmosphere or imaging system is contained in the D/r_0 term, which can be factored out as a scale constant. The covariance matrix can then be computed for $D/r_0 = 1$ and used for arbitrary turbulence strengths by simply scaling the result by $(D/r_0)^{5/3}$. The normalized covariance matrix, for modes 2-10 is given in Table 4.2.

Table 4.2: Normalized Zernike mode covariance for the first 10 modes excluding piston and assuming Kolmogorov statistics. To obtain the true covariance, multiply these values by $(D/r_0)^{5/3}$.

j	2	3	4	5	6	7	8	9	10
2	0.448	0	0	0	0	0	-0.0141	0	0
3	0	0.448	0	0	0	-0.0141	0	0	0
4	0	0	0.0232	0	0	0	0	0	0
5	0	0	0	0.0232	0	0	0	0	0
6	0	0	0	0	0.0232	0	0	0	0
7	0	-0.0141	0	0	0	0.0062	0	0	0
8	-0.0141	0	0	0	0	0	0.0062	0	0
9	0	0	0	0	0	0	0	0.0062	0
10	0	0	0	0	0	0	0	0	0.0062

One application of these results is the generation of simulated phase screens using Zernike modes. To build a random phase screen with Kolmogorov statistics, a vector of

Gaussian distributed random numbers characterized by the covariance of (4.35) is generated and used to weight the Zernike modes which are then summed.

The correlated Gaussian random numbers can be generated by utilizing the fact that any linear transformation of Gaussian random numbers is also Gaussian distributed [35]. Furthermore, given the joint Gaussian random vector $\vec{x} \sim \mathcal{N}(0, \mathbf{\Sigma})$, the transformation matrix $\mathbf{M}$, and defining $\vec{y} = \mathbf{M}\vec{x}$, the transformed vector $\vec{y}$ will be described by

$$\vec{y} \sim \mathcal{N}(0, \mathbf{M\Sigma M}^H) \tag{4.37}$$

If $\vec{x} \sim \mathcal{N}(0, \mathbf{I})$, as would be the case for computer-generated random numbers, $\mathbf{M\Sigma M}^H = \mathbf{MM}^H$. Because the correlation matrix is symmetric and positive definite, a vector of random numbers with the desired correlation $\mathbf{\Gamma}$ can be generated by factoring the correlation matrix into its Cholesky decomposition such that

$$\mathbf{\Gamma} = \mathbf{C}^H\mathbf{C} \tag{4.38}$$

then generating a vector of uncorrelated random numbers and performing the vector-matrix product

$$\vec{y} = \mathbf{C}^H\vec{x} \tag{4.39}$$

at which point $\vec{y}$ has the desired statistics.

To simplify computation, a reduced set of Zernike modes may be used, sacrificing some of the high-frequency content. While this is an incomplete representation, the results shown in Table 4.2 clearly show that the parameter variance, and thereby the "power" in a given mode, decreases rapidly with increasing order. Because of the ability of the Zernike mode basis set to efficiently pack the majority of the aberration into the lowest order terms, a truncated parameter vector can still accurately represent the aberration phase.

4.3 Phase Diversity Imaging

One successful approach to producing high-resolution images in the presence of atmospheric and systemic aberrations is to use phase diversity (PD) techniques to post-process aberrated data. First proposed by Gonsalves as a means for phase estimation [22,23], phase diversity imaging uses information contained in both a conventional and an intentionally aberrated image of the object, both of which have been degraded by the same unknown aberration. Before modifying this approach to accommodate the active imaging scenario, an understanding of the basic technique is required. Once the basic algorithm is understood, it can be extended to work with active imaging data.

A conventional phase diversity imaging system is composed of a traditional imaging system (in this case a telescope focused at infinity) with the addition of a second detector plane which has been subjected to some known aberration. Because of easy implementation and mathematical simplicity, system developers have traditionally chosen defocus for the known aberration. A notional phase-diversity imaging system is illustrated in Fig. 4.3. The following development of conventional phase diveristy imaging under Gaussian and Poisson noise statistics is derived from that of Paxman *et al.* [39].

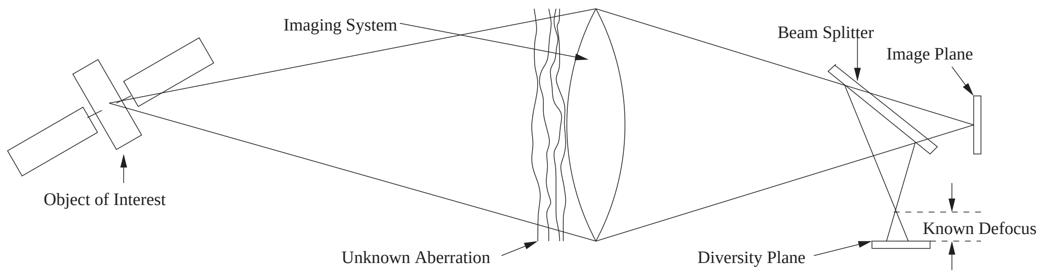


Figure 4.2: A conventional phase diversity imaging system. Data is collected in the focal plane and in a plane with some known amount of defocus added.

4.3.1 System Model. The initial step in developing a phase diversity algorithm is to model the noiseless imaging system. Consistent with section 4.1.2 the system is modeled in terms of its optical transfer functions

$$H_{FP}(u,v) = P(u,v)\exp\left[-j\phi(u,v)\right] \tag{4.40a}$$

$$H_{DP}(u,v) = P(u,v)\exp\left[-j\left(\phi(u,v)+\phi_D(u,v)\right)\right] \tag{4.40b}$$

where $P(u,v)$ is the amplitude of the imaging system's pupil-function, $\phi(u,v)$ is the unknown aberration phase induced by the optical system or the atmosphere, and $\phi_D(u,v)$ is the known diversity phase. To simplify analysis, the aberrating phase and diversity phase are written as an expansion in some suitable basis set such that

$$\phi(u,v) = \sum_k \alpha_k \psi_k(u,v) \tag{4.41}$$

and

$$\phi_D(u,v) = \sum_k \beta_k \psi_k(u,v) \tag{4.42}$$

where α_k and β_k are scalar weights, and ψ_k is the k^{th} basis function.

Because of their ability to describe complex aberrations with relatively few basis functions, the Zernike modes as described in section 4.1.3 are used as the basis set. Selecting defocus for the diversity aberration, the diversity phase is completely described by the fourth Zernike mode such that

$$\phi_D(u,v) = \beta_4 \psi_4(u,v) \tag{4.43}$$

Once the system model is complete, the PSF's are given by

$$s_m(x,y) = |h_m(x,y)|^2 \tag{4.44}$$

where $m \in \{FP, DP\}$ and the impulse response is defined by

$$h_m(x,y) = \mathcal{F}\{H_m(u,v)\} \tag{4.45}$$

The incoherent images, g_m, that would result in the absence of noise are then given by

$$g_m(x,y) = \mathbf{o}\,(x,y) \otimes s_m(x,y) \tag{4.46}$$

where $\mathbf{o}$ is the idealized image that would be formed under an aberration-free geometrical optics approximation and $\otimes$ represents a two-dimensional convolution.

At this point in the development, it's necessary to statistically describe the dominant noise source; then, using the noise statistics, develop an appropriate estimator. In the simplest cases such as when detector thermal noise is the dominant source, the noise is described as an additive white Gaussian process. Alternatively, for low-noise detectors, the dominant noise source would more likely be photon or shot noise characterized by Poisson statistics. Each noise source leads to a different estimator, both of which will be described below.

4.3.2 Additive Gaussian Noise Limited Detection. When the noise is adequately described using additive white Gaussian statistics, the data d_m is modeled by

$$d_m(x,y) = g_m(x,y) + \hat{n}_m(x,y) \tag{4.47a}$$

$$= \mathbf{o}(x,y) \otimes s_m(x,y) + \hat{n}_m(x,y) \tag{4.47b}$$

where $\hat{n}_m(x,y)$ is spatially uncorrelated Gaussian noise for the m^{th} channel. The probability distribution for the data is then given by

$$p_{\mathcal{D}}(\mathcal{D}) = \prod_m \prod_{x,y} \frac{1}{\sqrt{2\pi\sigma_{\hat{n}_m}^2}} \exp\left[-\frac{(d_m(x,y) - g_m(x,y))^2}{2\sigma_{\hat{n}_m}^2}\right] \tag{4.48}$$

where $\mathcal{D}$ is a complete data set containing both focal-plane and diversity plane intensity measurements. From (4.48) the log-likelihood function is then

$$\mathcal{L}(\mathbf{o},\vec{\alpha}) = -\sum_m \sum_{x,y} \left[d_m(x,y) - \mathbf{o}(x,y) \otimes s_m(x,y)\right]^2 \tag{4.49}$$

where $\vec{\alpha} = \{\alpha_1 \ldots \alpha_k\}$ is the vector of Zernike expansion coefficients for the aberration phase. Applying both the convolution theorem and Parseval's theorem, (4.49) can be written

$$\mathcal{L}(\mathbf{o},\vec{\alpha}) = -\sum_m \sum_{u,v} \left[D_m(u,v) - \mathbf{O}(u,v)S_m(u,v)\right]^2 \tag{4.50}$$

where D_m, $\mathbf{O}$, and S_m are Fourier transforms of d_m, $\mathbf{o}$, and s_m respectively. The maximum likelihood estimate for $\vec{\alpha}$ and $\mathbf{o}$ is then found by maximizing (4.50). This is equivalent to minimizing the mean squared error metric used by Gonsalves [23]. Furthermore, Gonsalves showed that the dependence on $\mathbf{o}$ could be removed by forming a closed-form estimate for the object given the data and aberration defined by

$$\mathbf{O}_M(u,v) = \frac{D_{FP}(u,v)S_{FP}^*(u,v) + D_{DP}(u,v)S_{DP}^*(u,v)}{\left|S_{FP}(u,v)\right|^2 + \left|S_{DP}(u,v)\right|^2} \tag{4.51}$$

which is essentially an application of a Weiner filter to the available data.

At this point, (4.51) can be substituted back into (4.50) resulting in a new objective function that only depends on the aberration parameters, greatly reducing the space over which maximization must be done. The reduced objective function is given by

$$\mathcal{L}_M(\vec{\alpha}) = -\sum_{u,v} \frac{\left|D_{FP}(u,v)S_{FP}(u,v) - D_{DP}(u,v)S_{DP}(u,v)\right|^2}{\left|S_{FP}(u,v)\right|^2 + \left|S_{DP}(u,v)\right|^2} \tag{4.52}$$

Paxman *et al.* [39] showed that an equivalent objective function requiring fewer computations could be written

$$\mathcal{L}_M(\vec{\alpha}) = \sum_{(u,v)\in\zeta_1} \frac{\left|\sum_m D_m(u,v)S_m^*(u,v)\right|^2}{\sum_m \left|S_m(u,v)\right|^2} - \sum_{(u,v)\in\zeta_2} \sum_m \left|D_m(u,v)\right|^2 \tag{4.53}$$

where the space ζ containing (u,v) has been partitioned into $\zeta = \{\zeta_1 \cup \zeta_2\}$ where ζ_2 contains all points where $S_m(u,v) = 0$ for all values of $m \in \{FP, DP\}$. Either (4.52) or (4.53) is then maximized over the space of aberration parameters to determine the unknown aberration. The aberration is then used to generate the PSFs which are then substituted into (4.51) to generate a reconstructed image.

The maximization of (4.52) or (4.53) can be done using any appropriate optimization routine. However, most efficient nonlinear optimization methods, such as quasi-Newton's methods, require the computation or estimation of gradients. Paxman *et al.* [39] showed that the required gradients were given by

$$\frac{\partial \mathcal{L}_M}{\partial \alpha_k} = -4\sum_{u,v} \phi_k(u,v)\,\mathcal{I}m\left\{\sum_m H_m(u,v)\left(Z_m \otimes H_m^*\right)(u,v)\right\} \tag{4.54}$$

where $\mathcal{I}m\{\cdot\}$ is the imaginary part of the argument and

$$Z_m(u,v) = \begin{cases} \frac{\sum_l |S_l|^2 \left(\sum_j D_j S_j^*\right) D_m^* - \left|\sum_j D_j S_j^*\right|^2 S_m^*}{\sum_l |S_l|^2}, & (u,v)\in\zeta_1 \\ 0, & (u,v)\in\zeta_0 \end{cases} \tag{4.55}$$

The dependence on (u,v) in (4.55) is implied, but for brevity has not been written out. Because tractable analytic expressions for the gradients are available, standard optimization methods such as quasi-Newton's methods can be used to determine the maximum likelihood estimates for $\vec{\alpha}$ and the associated reconstructed image $\mathbf{o}$.

4.3.3 Poisson Noise Limited Detection. While additive Gaussian noise sources lead to tractable estimates, in several operating regimes of interest the dominant noise source is photon noise characterized by Poisson statistics. Assuming that the noise process is spatially independent, the probability distribution for a data set is given by

$$p_{\mathcal{D}}(\mathcal{D}) = \prod_m \prod_{x,y} \frac{g_m(x,y)^{d_m(x,y)} \exp\left[-g_m(x,y)\right]}{d_m(x,y)!} \tag{4.56}$$

where as with the Gaussian noise case, $\mathcal{D}$ is the data set containing both FP and DP data. Using (4.56), the log-likelihood function for the object and aberration is then given by

$$\mathcal{L}(\mathbf{o}, \vec{\alpha}) = \sum_m \sum_{x,y} \left[d_m(x,y) \ln g_m(x,y) - g_m(x,y)\right] \tag{4.57}$$

where terms that don't depend on $\mathbf{o}$ or $\vec{\alpha}$ have been dropped for convenience. Considering the second term in (4.57)

$$\begin{aligned}
\sum_m \sum_{x,y} g_m(x,y) &= \sum_m \sum_{x,y} \sum_{\xi,\eta} \mathbf{o}(\xi,\eta) s_m(x-\xi, y-\eta) && \text{(4.58a)}\\
&= \sum_m \sum_{\xi,\eta} \mathbf{o}\left(\xi,\eta\right) \sum_{x,y} s_m(x-\xi, y-\eta) && \text{(4.58b)}\\
&= \sum_m \sum_{\xi,\eta} \mathbf{o}(\xi,\eta) \sum_{u,v} \left|H_m(u,v) \exp\left\{-j2\pi\left[(x-\xi)u + (y-\eta)v\right]\right\}\right|^2 && \text{(4.58c)}\\
&= \sum_{\xi,\eta} \mathbf{o}\left(\xi,\eta\right) \sum_m \sum_{u,v} \left|H_m(u,v)\right|^2 && \text{(4.58d)}\\
&= \sum_{\xi,\eta} \mathbf{o}(\xi,\eta) \sum_m \sum_{u,v} P^2(u,v) && \text{(4.58e)}
\end{aligned}$$

where Parseval's theorem and the Fourier shift theorem have been used. Note that the double sum over the pupil function $P(u,v)$ is neither a function of the object $\mathbf{o}$ nor the aberration $\vec{\alpha}$. Letting

$$\gamma \equiv \sum_m \sum_{u,v} P^2(u,v) \tag{4.59}$$

(4.58) becomes

$$\sum_{m}\sum_{x,y} g_m(x,y) = \gamma \sum_{x,y} \mathbf{o}\left(x,y\right) \tag{4.60}$$

and (4.57) simplifies to

$$\mathcal{L}(\mathbf{o},\vec{\alpha}) = \sum_{m}\sum_{x,y} \left[d_m(x,y)\ln g_m(x,y)\right] - \gamma \sum_{x,y} \mathbf{o}\left(x,y\right) \tag{4.61}$$

Removing explicit dependence on $\mathbf{o}$ from (4.61), as was done for additive Gaussian noise, requires that the object that maximizes (4.61) for a given aberration $\vec{\alpha}$ be found and substituted back into (4.61). However, a closed form solution to that problem has not been found. The approach then is to attempt to maximize the objective over the complete space of pixel intensities and aberration parameters. With the expanded parameter space over which to maximize, utilization of an efficient large-scale optimization routine becomes even more important. To this end, computation of the gradients with respect to both the aberrations and the object pixels is the next logical step.

Differentiating (4.61) with respect to $\mathbf{o}(x_o, y_o)$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{o}\left(x_o,y_o\right)} = \sum_{m}\sum_{x,y} d_m(x,y) \frac{\partial \ln\left[g_m(x,y)\right]}{\partial \mathbf{o}\left(x_o,y_o\right)} - \gamma \sum_{x,y} \frac{\partial \mathbf{o}\left(x,y\right)}{\partial \mathbf{o}\left(x_o,y_o\right)} \tag{4.62a}$$

$$= \sum_{m}\sum_{x,y} \frac{d_m(x,y)}{g_m(x,y)} \frac{\partial g_m(x,y)}{\partial \mathbf{o}\left(x_o,y_o\right)} - \gamma \tag{4.62b}$$

To compute the partial derivative of g_m, (4.13) and the definition of discrete convolution are applied to give

$$\frac{\partial g_m(x,y)}{\partial \mathbf{o}\left(x_o,y_o\right)} = \frac{\partial}{\partial \mathbf{o}\left(x_o,y_o\right)} \sum_{u,v} \mathbf{o}\left(u,v\right) s(x-u, y-v) \tag{4.63a}$$

$$= s_m(x-x_o, y-y_o) \tag{4.63b}$$

Substituting this into (4.62) gives the final form of the partial derivative of $\mathcal{L}$ with respect to $\mathbf{o}\left(x_o,y_o\right)$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{o}\left(x_o,y_o\right)} = \sum_{m}\sum_{x,y} \frac{d_m(x,y) s_m(x-x_o, y-y_o)}{g_m(x,y)} - \gamma \tag{4.64}$$

Differentiating (4.61) with respect to aberration parameter α_k results in an expression similar to (4.62) with the exceptions that the constant term γ is dropped, and $\frac{\partial}{\partial \mathbf{o}(x_o,y_o)}$ is replaced by $\frac{\partial}{\partial \alpha_k}$

$$\frac{\partial \mathcal{L}}{\partial \alpha_k} = \sum_m \sum_{x,y} \frac{d_m(x,y)}{g_m(x,y)} \frac{\partial g_m(x,y)}{\partial \alpha_k} \tag{4.65}$$

The partial derivative of $g_m(x,y)$ with respect to α_k is then found by using (4.13) and the definition of discrete convolution, resulting in

$$\frac{\partial g_m(x,y)}{\partial \alpha_k} = \sum_{u,v} \mathbf{o}\,(u,v)\, \frac{\partial s_m(x-u,y-v)}{\partial \alpha_k} \tag{4.66}$$

Using the definition of s_m from (4.44) the partial derivative of $s_m(x,y)$ with respect to α_k can be written

$$\frac{\partial s_m(x,y)}{\partial \alpha_k} = h_m^*(x,y) \frac{\partial h_m(x,y)}{\partial \alpha_k} + h_m(x,y) \frac{\partial h_m^*(x,y)}{\partial \alpha_k} \tag{4.67}$$

Applying the definitions from (4.40) and (4.45) to the partial derivative of $h_m(x,y)$ and interchanging the order of integration and differentiation results in

$$\frac{\partial h_m(x,y)}{\partial \alpha_k} = \mathcal{F}\Big\{ j\psi_k(u,v) P(u,v) \exp\Big[j\Big(\beta_m \psi_4(u,v) + \sum_n \alpha_n \psi_n(u,v) \Big)\Big]\Big\} \tag{4.68}$$

where $\beta_{FP} = 0$ and β_{DP} defines the defocus diversity phase. The partial derivative of s_m with respect to α_k is then

$$\frac{\partial s_m(x,y)}{\partial \alpha_k} = 2\mathcal{Re}\Big\{ h_m^*(x,y) \mathcal{F}\Big\{ j\psi_k(u,v) P(u,v) \exp\Big[j\Big(\beta_m \psi_4(u,v) + \sum_n \alpha_n \psi_n(u,v) \Big)\Big]\Big\} \tag{4.69a}$$

$$= s_m'(x,y) \tag{4.69b}$$

where $\mathscr{Re}\{\cdot\}$ is the real part of the argument. Substituting (4.69b) back into (4.66) and (4.65) results in the final form for the derivative of $\mathcal{L}$ with respect to α_k

$$\frac{\partial \mathcal{L}}{\partial \alpha_k} = \sum_{m,x,y} \frac{d_m(x,y)}{g_m(x,y)} \sum_{\xi,\eta} \mathbf{o}(\xi,\eta) s'(x-\xi, y-\eta) \tag{4.70a}$$

$$= \sum_{m,x,y} \frac{d_m(x,y)}{g_m(x,y)} \left[\mathbf{o}(x,y) \otimes s'(x,y)\right] \tag{4.70b}$$

At this point the objective function and gradients can be used with any applicable nonlinear optimization routine to simultaneously develop an estimate of the object and aberration parameters.

4.3.4 Multi-Frame Phase Diversity Reconstruction. Seldin and Paxman expanded phase-diversity image reconstruction for Poisson statistics to allow for multiple data frames where each frame has been corrupted by a different atmospheric aberration [45]. This resulted in a hybrid between multi-frame blind deconvolution and conventional phase diversity reconstruction that allowed for better handling of noisy data such as that encountered at low light levels.

Defining the aberration set $\boldsymbol{\alpha}$ as

$$\boldsymbol{\alpha} \equiv \left\{\vec{\alpha}^1, \vec{\alpha}^2, \ldots, \vec{\alpha}^F\right\} \tag{4.71}$$

where $\vec{\alpha}^f$ is a vector describing the aberration for the f^{th} frame, the point spread functions are given by

$$s_m^f(x,y) = \left|\mathcal{F}\left\{P(u,v)\exp\left[-j\left(\beta_m \psi_4(u,v) + \sum_k \alpha_k^f \psi_k(u,v)\right)\right]\right\}\right|^2 \tag{4.72}$$

resulting in

$$g_m^f(x,y) = \mathbf{o}(x,y) \otimes s_m^f(x,y) \tag{4.73}$$

where $m \in \{FP, DP\}$. Assuming statistical independence between data frames, the joint probability distribution for a set of F frames of phase diversity data is then given by

$$p_{\mathcal{D}}(\mathcal{D}) = \prod_f \prod_m \prod_{x,y} \frac{g_m^f(x,y)^{d_m^f(x,y)} \exp\left[-g_m^f(x,y)\right]}{d_m^f(x,y)!} \tag{4.74}$$

where $f \in \{1 \dots F\}$ and the data set $\mathcal{D}$ contains F sets of phase-diversity data. Other than expanding the search space to include additional aberration parameters, the further development of this formulation is identical to that presented in section 4.3.3 resulting in an objective function given by

$$\mathcal{L}(\mathbf{o}, \vec{\alpha}) = \sum_f \sum_m \sum_{x,y} \left[d_m^f(x,y) \ln g_m^f(x,y) - g_m^f(x,y)\right] \tag{4.75}$$

where the f superscript indexes the data frames. The model images $g_m^f(x,y)$ are determined as before, using the aberration parameters for the current data frame. The gradients are identical to those of (4.64) and (4.70) for any given value of f.

Seldin and Paxman [45] observed that the algorithm tended to converge to local maxima that did not represent the known object. In an effort to better condition the problem, they incorporated a "sieve" to enforce some degree of smoothness on the reconstruction. In its basic form, the sieve is a convolution kernel used to help eliminate local maxima in the objective function. As implemented, the object was defined by

$$\mathbf{o}\,(x,y) = \sum_{\xi,\eta} \mathbf{o}'(\xi,\eta) v(x-\xi, y-\eta) \tag{4.76}$$

where $v(x,y)$ was an otherwise unspecified two-dimensional Gaussian convolution kernel. At this point, $g_m^f(x,y)$ is then defined as

$$g_m^f(x,y) = \sum_{u,v} \sum_{\xi,\eta} \mathbf{o}'(\xi,\eta) v(x-\xi, y-\eta) s_m^f(u-\xi, v-\eta) \tag{4.77}$$

The underlying $\mathbf{o}'$ is estimated using the algorithm, then used in (4.76) to reconstruct the image. Similar techniques have also been suggested to better condition single-frame phase diversity [22].

When the unknown aberration is due to atmospheric turbulence, the known statistics of the atmosphere can be introduced into the problem to help further condition the problem. Assuming Kolmogorov statistics for the atmospheric aberration and a Zernike-mode expansion, the covariance of the Zernike-mode turbulence coefficients is known [37] and can be added as a regularization term to the objective function of (4.75).

Thelen *et al.* [48] proposed a framework for incorporating the prior knowledge about the aberration into the objective function. Noting that the aberration parameters are correlated Gaussian random variates, the joint probability distribution function for a given set of N aberrations is given by

$$p_{\vec{\alpha}^f}(\vec{\alpha}^f) = \frac{1}{\sqrt{(2\pi)^N |\mathbf{\Gamma}_z|}} \exp\left(-\frac{1}{2}\left(\vec{\alpha}^f\right)^T \mathbf{\Gamma}_z^{-1}\vec{\alpha}^f\right) \tag{4.78}$$

where T is a transpose, $\vec{\alpha}^f$ is a vector composed of the Noll-ordered aberration parameters for the f^{th} frame, and $|\mathbf{\Gamma}_z|$ is the determinant of the covariance matrix $\mathbf{\Gamma}_z$. This probability distribution may then be converted into a log-likelihood function given by

$$\mathcal{L}_2(\alpha) = -\frac{1}{2}\sum_f \left(\vec{\alpha}^f\right)^T \mathbf{\Gamma}_z^{-1}\vec{\alpha}^f \tag{4.79}$$

which can then be incorporated into (4.75) resulting in a modified objective function for the multi-frame case given by

$$\mathcal{L}'(\mathbf{o},\alpha) = \sum_f \sum_m \sum_{x,y} \left[d_m^f(x,y)\ln g_m^f(x,y) - g_m^f(x,y)\right] - \frac{1}{2}\sum_f \left(\vec{\alpha}^f\right)^T \mathbf{\Gamma}_z^{-1}\vec{\alpha}^f \tag{4.80}$$

The only additional information now required before passing (4.80) to a gradient-type optimization routine is the gradient of (4.79). Given that (4.79) does not depend on the object $\mathbf{o}$, that portion of the gradient is clearly zero. For the partial derivatives with respect to to individual aberration parameter α_n^f, Thelen *et al.* showed that

$$\frac{\partial}{\partial \alpha_n^f}\mathcal{L}_2 = -\left(\mathbf{\Gamma}_z^{-1}\vec{\alpha}^f\right)_n \tag{4.81}$$

i.e. the n^{th} element of the vector that results from multiplication of the correlation matrix inverse with the f^{th} aberration vector.

At this point, the tools required for building a system model are complete, and all of the pieces are in place to begin development of a statistical image reconstruction algorithm. The only remaining step is to combine the estimation and optimization methods given in Chapter II with the system models developed in Chapters III and IV.

V. Active Phase Diversity Imaging

One critical failure of conventional imaging techniques is their reliance on natural illumination. There exist many scenarios of interest where the object to be imaged is in the earth's shadow and cannot be imaged using conventional methods. An approach to overcoming this shortcoming is to use lasers to actively illuminate the target, in effect implementing a "flash bulb" for the imaging system. However, the unique properties of laser light introduce both complications and possibilities that are not present with natural or incoherent light. With the understanding of estimation methods, diffraction, imaging, and statistical properties of coherent light developed in chapters II through IV, it's possible to begin development of an image reconstruction algorithm appropriate for an active imaging system.

Several techniques have been proposed over the years to utilize coherent illumination to reconstruct an image including imaging correlography [17, 44, 50], sheared-beam interferometry [50], root reconstructors [31], reference-wave heterodyne detection [6, 7], analytic methods [28], iterative statistical estimation/deconvolution methods [30, 43], optimization methods [34], and Fourier-telescopy [3, 29, 32, 50]. The configuration being considered is an active PD imaging (APDI) system as shown in Fig. 5.1 and consists of a conventional PD imaging system with a laser illuminator and an additional sensor to record PP-intensity. The following sections detail the algorithmic modifications required to expand PD imaging as described in Chapter IV to the case of active illumination.

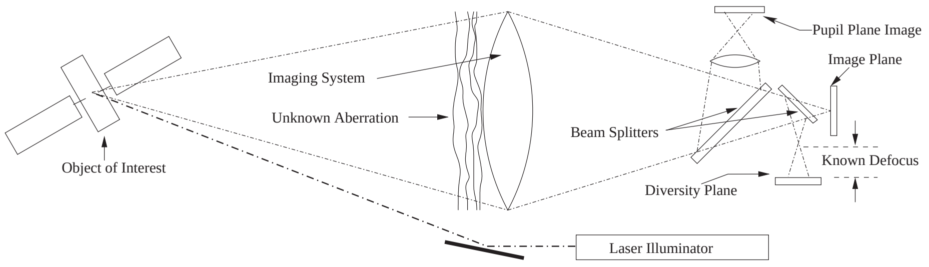


Figure 5.1: An active PD imaging system consists of a conventional PD system with an additional sensor to record PP intensity and a laser illuminator

5.1 Noise Model

The first, and possibly most significant difficulty in working with coherent illumination is the introduction of speckle noise as described in chapter III. Because nearly all surfaces are rough on the scale of an optical wavelength, the wave reflected from the object will have a spatially random phase, that when propagated will cause the various field components to interfere and produce the grainy speckle patterns characteristic of reflected laser light. As shown previously, speckle noise follows exponential statistics such that for a given pixel

$$p_I(I) = \frac{1}{\mathrm{E}\left[I\right]} \exp\left[-\frac{I}{\mathrm{E}\left[I\right]}\right] \tag{5.1}$$

where it has been assumed that $I \geq 0$.

One of the biggest consequences of laser speckle is its extremely noisy nature as demonstrated by the fact that the mean and standard deviation are equal. Defining the signal-to-noise ratio (SNR) as the ratio of the mean signal with the noise standard deviation, speckle noise maintains a SNR of 1 regardless of the signal strength. Unlike photon or detector noise sources, SNR doesn't improve with higher signal levels. As a result of the noisy nature of active illumination, a multi-frame approach will be required to help mitigate noise effects.

While any one pixel in the image plane follows the statistics of (5.1), the joint statistics of the detected intensity are required to develop a reconstruction algorithm. Given that the complex field is Gaussian distributed in all planes, one could ideally perform variable transformations on the joint-Gaussian distribution as was done for (3.40), and integrate out all phase dependence. However, beyond the second-order distribution derived in Chapter II the applicable integrations and transformations become intractable. As a result, an approximate form for the joint-distribution must be developed.

The simplest approximation is to assume statistical independence between pixels, and form the joint distribution by multiplying the marginals. For this simplification to be valid, the correlation between data points must be negligibly small. In the active imaging case, the required correlation can be found by using the mutual intensity in accordance with the complex Gaussian moment theorem of (3.34). As shown in chapter III, the mutual intensity in the PP of the imaging system can be described by a scaled Fourier transform of the object. This mutual intensity can then be propagated to the FP and DP of the imaging system using the more general (3.24). However, the required integrals become prohibitively difficult. Fortunately, it was shown by Zernike that in nearly all cases of interest, the light in the PP of a coherent imaging system could be approximated as a constant intensity incoherent source [24]. Under this assumption, the Van Cittert-Zernike theorem can be used to determine the FP and DP mutual intensities.

Substituting the modulus squared of the imaging system's pupil-function in for the intensity in (3.31) and normalizing

$$\mu_c(\Delta u, \Delta v) = \iint_{-\infty}^{\infty} |P(x,y)|^2 \exp\left\{-j\frac{2\pi}{\lambda f}\left(\Delta u x + \Delta v y\right)\right\} \mathrm{d}x\mathrm{d}y \tag{5.2}$$

where $\mu_c(\Delta u, \Delta v)$ is the complex correlation coefficient for points separated by $(\Delta u, \Delta v)$. For a clear circular aperture, this correlation is given by the scaled point-spread function of the un-aberrated imaging system where the spatial variables (x, y) have been replaced by $(\Delta x, \Delta y)$. Assuming a clear circular pupil, the modulus of the mutual intensity falls off rapidly with increasing point separation as shown in Fig. 5.2. Assuming that the pixel spacing of the detector is small compared with the width of the PSF, measured pixel intensities are hereafter assumed statistically independent.

One major failure of the assumptions used to generate Fig. 5.2 is that they provide no insight into the average detected intensities. An alternative approach for finding these quantities stems from the description of the mutual intensity in the object plane given by (3.29). This description is identical to that of an incoherent source, and the effect of averaging across many speckle realizations will be equivalent to that obtained for incoherent illumination. Because of this, the modeled noiseless images representing the mean of the statistical distributions are described by the incoherent image formation model of (4.13).

An additional consideration is the temporal correlation between speckle realizations. In the absence of any relative motion between the object, illumination source, and imaging system; the phase relationships that give rise to speckle will remain constant, and speckle observed in the observation planes will be stationary. However, even small relative changes on the order of a wavelength are sufficient to significantly change the speckle pattern. For this work it's assumed that the time over which speckle realizations become de-correlated

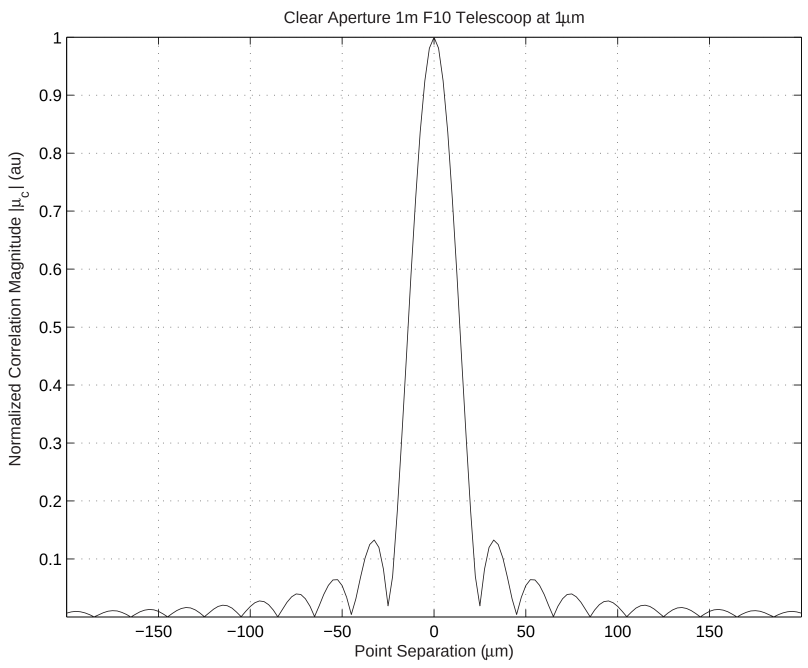


Figure 5.2: Approximate normalized correlation magnitude as a function of point separation for a $1m$ F10 telescope operating at $1\mu m$. The correlation falls off rapidly with increasing point separation.

is small compared with the sampling time of the imaging system. Therefore, temporal independence of speckle realizations is assumed.

Finally, to form the approximate joint probability distribution, independence between observation planes is assumed. Though not rigorously justified, this assumption is necessary to form a tractable reconstruction algorithm. Combining these simplifications and assumptions to form a joint distribution similar to (4.56), the joint probability distribution for a multi-frame active PD data set is given by

$$p_{\mathcal{D}}(\mathcal{D}) = \prod_{f,m,x,y} \frac{1}{g_m^f(x,y)} \exp\left[-\frac{d_m^f(x,y)}{g_m^f(x,y)}\right] \tag{5.3}$$

where $g_m^f(x,y)$ is the ensemble average intensity in the FP or DP for the f^{th} frame and is as given in (4.13).

5.2 Static Aberrations

The first reconstruction case considered is when the image is blurred by an aberration that isn't changing across data frames. This could represent a system that suffers from an unknown aberration due to manufacturing defects or misalignment, or a system with a slowly varying aberration.

5.2.1 Basic Objective Function.

For a static aberration, the model blurred image $g_m(x, y)$ is the same across all data frames, resulting in a slightly simplified probability distribution given by

$$p_{\mathcal{D}}(\mathcal{D}) = \prod_{f,m,x,y} \frac{1}{g_m(x,y)} \exp\left[-\frac{d_m^f(x,y)}{g_m(x,y)}\right] \tag{5.4}$$

Re-casting this as a likelihood equation for the object and aberration, the log-likelihood is given by

$$\mathcal{L}(\mathbf{o}, \vec{\alpha}) = -\sum_{f,m,x,y} \left(\ln g_m(x,y) + \frac{d_m^f(x,y)}{g_m(x,y)} \right) \tag{5.5}$$

Because only the data is frame-dependent, the summation over f can be distributed such that for F frames of data

$$\mathcal{L}(\mathbf{o}, \vec{\alpha}) = -\sum_{m,x,y} \left(F \ln g_m(x,y) + \frac{1}{g_m(x,y)} \sum_f d_m^f(x,y) \right) \tag{5.6}$$

5.2.2 Pupil Plane Data Regularization.

The noisy nature of laser speckle makes regularization particularly important. However, the speckle can also provide additional information that can be used for regularization. Consider the object information encoded in the speckle pattern observed in the PP of the imaging system as described in section 3.4.2. Restating (3.38), the normalized PP speckle PSD can be described by

$$\hat{\Phi}_{PP}(x,y) = \delta(x,y) + \gamma \mathbf{o}(x,y) \circledast \mathbf{o}(x,y) \tag{5.7}$$

where the frequency variables of the PSD have been replaced with (x, y) for convenience in later calculations, $\hat{\Phi}_{PP}(x, y)$ is the ensemble averaged PP-PSD, and $\circledast$ represents a two-dimensional autocorrelation. Consequently, a large amount of information about the object is contained in the PP intensity which can be used to help regularize the problem.

Before this information can be used, a statistical model for the PSD must be formed. As with all data from an actively illuminated system, the PP data will suffer from speckle noise. Recall that the intensity in the PP is exponentially distributed with a constant mean across the aperture. Using the definition for the PSD as

$$\Phi_{PP}(x,y) = \left|\mathcal{F}\{I_{PP}(\xi,\eta)\}\right|^2 \tag{5.8}$$

where I_{PP} is the normalized intensity in the PP, we can consider the effect of speckle statistics on the PSD.

Writing the Fourier transform in its integral form,

$$\mathcal{F}\{I_{PP}(x,y)\} = \iint_{-\infty}^{\infty} I_{PP}(x,y) \exp\left[-j\left(ux+vy\right)\right] \mathrm{d}x\mathrm{d}y \tag{5.9}$$

it can be interpreted as an infinite sum of unit-magnitude complex numbers weighted by exponentially distributed scalars. If the $I_{PP}(x,y)$ pixels are independent, by the central limit theorem the resulting real and imaginary components of the Fourier transform will have Gaussian statistics. Though I_{PP} isn't strictly spatially independent, a histographic analysis of simulated data shows that the transformed intensity closely follows Gaussian statistics. To get the distribution of the PSD, the same variable transformations that were used in (3.28) are applied consistent with (5.8), resulting in an exponential distribution for any given point in the PSD.

In order to form a tractable joint distribution, spatial independence must again be assumed. According to diffraction theory, propagation of light to the FP of an imaging system is mathematically equivalent to a Fourier transform. Using this insight, the same arguments that justified spatial independence for the FP and DP again apply. The mutual intensity in the now mathematically defined FP is proportional to the Fourier transform of the imaging system's pupil, and consequently falls off rapidly for small point separations. Statistical independence is again assumed, and the joint probability distribution for the

PP-PSD is then given by

$$p_{\Phi_{PP}}(\Phi_{PP}) = \prod_{x,y} \frac{1}{\hat{\Phi}_{PP}(x,y)} \exp\left[-\frac{\Phi_{PP}(x,y)}{\hat{\Phi}_{PP}(x,y)}\right] \tag{5.10}$$

To negate any numerical difficulties introduced by the $\delta(x,y)$ in (5.7), it's subtracted out of the PSD during computation. While doing so causes the PSD at the origin to no longer follow the predicted statistics, this effect is ignored for simplicity. The probability distribution for the modified speckle PSD is then given by

$$p_{d_{PP}}(d_{PP}) = \prod_{x,y} \frac{1}{g_{PP}(x,y)} \exp\left[-\frac{d_{PP}(x,y)}{g_{PP}(x,y)}\right] \tag{5.11}$$

where the PP data for the f^{th} frame is defined as

$$d^f_{PP}(x,y) = \left|\mathcal{F}\{I^f_{PP}(\xi,\eta)\}\right|^2 - \delta(x,y) \tag{5.12}$$

and the model data is defined as

$$g_{PP}(x,y) = \hat{\Phi}_{PP}(x,y) - \delta(x,y) \tag{5.13a}$$

$$= \mathbf{o}(x,y) \circledast \mathbf{o}(x,y) \tag{5.13b}$$

It should be noted that in this case the model data doesn't depend on the aberration, resulting in a lack of dependence on either f or $\vec{\alpha}$. Additionally, g_{PP} is invariant to rotation of $\mathbf{o}$ through 180°.

At this point, the log-likelihood function for the object given a PP data set can be written

$$\mathcal{L}_{PP}(\mathbf{o}) = \sum_{f,x,y}\left[\ln g_{PP}(x,y) - \frac{d^f_{PP}(x,y)}{g_{PP}(x,y)}\right] \tag{5.14}$$

This likelihood function can then be added to the likelihood function of (5.5) to produce the regularized objective function for image reconstruction

$$\mathcal{L}(\mathbf{o},\vec{\alpha}) = -\sum_{f,m,x,y}\left[\ln\left(g_m(x,y)\right) + \frac{d^f_m(x,y)}{g_m(x,y)}\right] \tag{5.15}$$

where now $m \in \{FP, DP, PP\}$ to include the PP regularization term.

5.2.3 Optimization Methodology. As with the Poisson-noise limited case, this objective function doesn't lend itself to a reduction in parameter space as the Gaussian noise model did. As a result, large-scale optimization methods must be employed to simultaneously solve for both the object and aberration parameters. Recognizing that the most efficient optimization methods tend to require gradient information, the first step towards optimization is gradient derivation. For the objective function of (5.15), the partial derivative of $\mathcal{L}(\mathbf{o}, \vec{\alpha})$ with respect to any parameter χ will be given by

$$\frac{\partial \mathcal{L}(\mathbf{o}, \vec{\alpha})}{\partial \chi} = -\sum_{f,m,x,y} \left[\frac{1}{g_m(x,y)} - \frac{d_m^f(x,y)}{[g_m(x,y)]^2} \right] \frac{\partial g_m(x,y)}{\partial \chi} \tag{5.16}$$

The task then remains to compute the partial derivative of $g_m(x,y)$ with respect to χ.

Because the model images for both the FP and DP are the same as those for conventional PD imaging, the partial derivatives of g_{FP} and g_{DP} with respect to $\mathbf{o}(x_o, y_o)$ and α_n that were previously derived in (4.63) and (4.66) remain valid. They are re-stated here as

$$\frac{\partial g_m(x,y)}{\partial \mathbf{o}(x_o, y_o)} = s_m(x - x_o, y - y_o) \tag{5.17}$$

and

$$\frac{\partial g_m(x,y)}{\partial \alpha_k} = \mathbf{o}(x,y) \otimes \hat{s}_m^k(x,y) \tag{5.18}$$

where

$$\hat{s}_m^k(x,y) = 2\mathcal{Re}\Big\{ h_m^*(x,y) \mathcal{F}\Big\{ j\psi_k(u,v) P(u,v) \exp\Big[j\Big(\beta_m \psi_4(u,v) + \sum_n \alpha_n \psi_n(u,v) \Big) \Big] \Big\} \tag{5.19}$$

and $m \in \{FP, DP\}$.

Because g_{PP} doesn't depend on $\vec{\alpha}$ its partial derivatives with respect to α_k are zero. The final piece required for the gradient is the partial derivative of g_{PP} with respect to $\mathbf{o}(x_o, y_o)$. The autocorrelation in (5.13) is written as

$$g_{PP}(x,y) = \sum_{\xi,\eta} \mathbf{o}(\xi,\eta) \mathbf{o}(x + \xi, y + \eta) \tag{5.20}$$

differentiating (5.20) with respect to $\mathbf{o}(x_o, y_o)$ interchanging the order of differentiation and summation, and applying the product rule, the partial derivative of $g_{PP}(x, y)$ with respect to $\mathbf{o}(x_o, y_o)$ is given by

$$\frac{\partial g_{PP}(x,y)}{\partial \mathbf{o}(x_o,y_o)} = \mathbf{o}(x_o - x, y_o - y) + \mathbf{o}(x_o + x, y_o + y) \tag{5.21}$$

One final consideration before an optimization routine can be used to determine the reconstructed object and the aberration parameters stems from the form of both (5.15) and (5.16). In both of these expressions, the objective function becomes undefined any time $g_m(x, y)$ goes to zero. In this case, the reconstruction becomes ill-conditioned, and the numerical optimization routine will likely fail to converge to a satisfactory solution.

One possible solution, as suggested for Poisson noise limited PD reconstruction, is the partitioning of the objective function into two discrete sets $\zeta = \{\zeta_1 \cup \zeta_2\}$ where ζ_0 contains all points where $g_m(x, y) = 0$ for all values of m. All contributions to the objective function from any point contained in ζ_0 are then thrown out. However, this approach doesn't adequately deal with the case when the data supports a non-zero value but the current guess object predicts a zero. In this case however, the gradient should indicate a more optimal solution with a non-zero value but would instead indicate a zero slope and an optimal answer. Furthermore, numerical roundoff and associated instability begin to have significant negative impacts when $g_m(x, y)$ and $d^f_m(x, y) \to 0$, which isn't adequately dealt with by partitioning the solution space.

As an alternative, a better conditioned homotopic problem [33] is formed by adding a bias b to both the data and the model such that

$$\mathcal{L}'(\mathbf{o}, \vec{\alpha}) = -\sum_{f,m,x,y} \left[ln(g_m(x,y) + b) + \frac{d^f_m(x,y) + b}{g_m\,(x,y) + b} \right] \tag{5.22}$$

and

$$\frac{\partial}{\partial \chi}\mathcal{L}'(\mathbf{o}, \vec{\alpha}) = -\sum_{f,m,x,y} \left[\frac{1}{g_m(x,y) + b} - \frac{d^f_m(x,y) + b}{[g_m(x,y) + b]^2} \right] \frac{\partial g_m(x,y)}{\partial \chi} \tag{5.23}$$

The homotopic problem converges uniformly to the original problem as $b \to 0$. The added bias can be viewed as an artificial detection dark-current or background illumination. The modified objective function is then passed to the numerical routine to obtain reconstructed images. The optimal value for b is explored in Chapter VI.

5.3 Dynamic Aberrations

To allow for a much broader range of operational scenarios, the unknown aberration must be allowed to vary. For the cases of interest, the aberration evolves with the changing atmosphere, blurring each data frame differently. The changing aberrations correspond with a new set of aberration parameters for each frame that must be included in the objective function. To deal with this, an expanded aberration set $\boldsymbol{\alpha}$ is defined as in (4.71), such that for F data frames

$$\boldsymbol{\alpha} = \left[\vec{\alpha}^1, \vec{\alpha}^2, \ldots \vec{\alpha}^F\right] \tag{5.24}$$

where $\vec{\alpha}^f$ is the aberration vector for the f^{th} data frame.

5.3.1 Basic Objective Function. The only fundamental change to the likelihood function of (5.15) required to extend the problem to include dynamic aberrations is to modify the model data $g_m(x, y)$ such that it depends on the frame number f as in (4.73). With this change, the basic objective function is given by

$$\mathcal{L}(\mathbf{o}, \boldsymbol{\alpha}) = -\sum_{f,m,x,y}\left[\ln\left(g_m^f(x,y)\right) + \frac{d_m^f(x,y)}{g_m^f(x,y)}\right] \tag{5.25}$$

where $m \in \{FP, DP\}$ and all other terms are as previously described.

5.3.2 Pupil Plane Data Regularization. Because the PP data is insensitive to the phase aberration, the PP regularization term of (5.14) is added to the basic objective function of (5.25) without modification. Incorporating this into the basic objective of (5.25) results in the PP-regularized objective function

$$\mathcal{L}(\mathbf{o}, \boldsymbol{\alpha}) = -\sum_{f,m,x,y}\left[\ln\left(g_m^f(x,y)\right) + \frac{d_m^f(x,y)}{g_m^f(x,y)}\right] \tag{5.26}$$

where now $m \in \{FP, DP, PP\}$.

Without further changes, this objective function could be passed to the solving routine to produce reconstructed images. However, additional information becomes available which can be incorporated into the algorithm when multiple aberration realizations are allowed.

5.3.3 Phase Aberration Regularization. Kolmogorov theory for turbulence predicts a well-defined power spectral density for phase fluctuations caused by atmospheric turbulence, parameterized by a measurable quantity r_0 [2]. With multiple realizations of the induced phase and information about r_0, we can begin to form an estimate for this PSD and penalize the objective function for realizations that deviate from the predicted standard. However, given that the aberration is parameterized by Zernike-mode expansion coefficients, a more efficient method is to directly relate the PSD to the expansion coefficients. Because the nature of the turbulence is not changed by the use of active illumination, Thelen's regularization term of (4.79) can be used without modification.

In addition to the inter-mode correlation, the temporal evolution of the aberration may be similarly useful. Under Taylor's frozen flow hypothesis [41], the internal structure of turbulence changes slowly compared with the velocity at which it passes by the field of view. With this hypothesis, and knowledge about the velocity and direction of the wind moving the turbulence, the temporal behavior of the expansion coefficients can be described. Roddier *et al.* [40] developed a description of the temporal correlation of the Zernike expansion coefficients under Taylor's hypothesis. Ten Brummelaar slightly modified Roddier's results [10]. Using ten Brummelaar's formulation, temporal correlations for the first 15 modes excluding piston and tip/tilt were computed and are shown in Fig. 5.3 for a $1m$ aperture, $10cm$ r_0, and a $10m/s$ x-axis oriented wind velocity. The correlations drop rapidly with increasing separation and mode order.

At this point, a regularization term similar to (4.79) could be applied to the objective function by building the temporal correlation matrices for each aberration parameter. However, there are several shortfalls of this approach that make it less attractive. First, the rapid drop in the correlation with small time separations would make the regularization term particularly susceptible to noise and other error sources for systems with a moderately slow frame-rate. Second, and more significantly, this approach requires that the turbulent

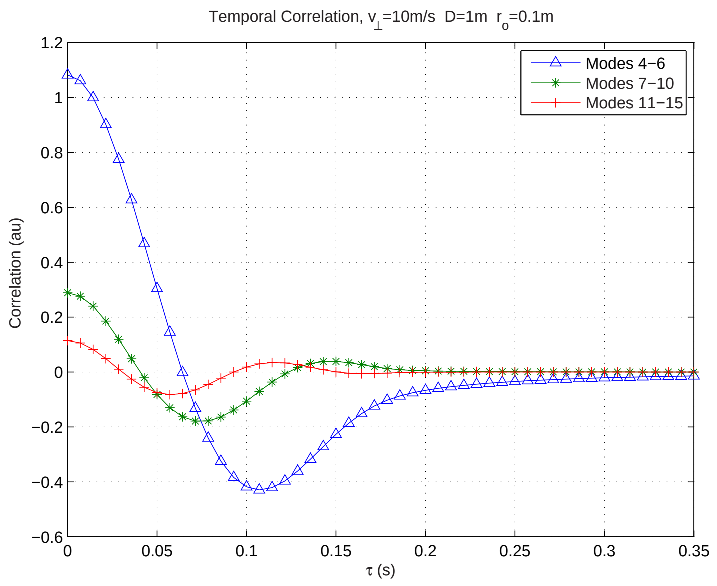


Figure 5.3: Predicted temporal correlations for the first 15 Zernike modes excluding piston and tip/tilt assuming a $1m$ aperture, $10cm$ r_0, and $10m/s$ x-oriented wind velocity. Note that the correlation falls off rapidly with increasing time separation and mode-order.

flow be adequately described by a single layer of turbulence moving with an *a-priori* known speed and direction. While the effects of turbulence on a single frame can be adequately described by a single turbulence layer in the PP, realistic turbulence is a volume effect. Given that wind velocities and directions vary with altitude, describing the effective motion of the turbulence using a single layer becomes significantly difficult. As evidence of the problems that can arise when predicting temporal statistics, ten Brummelaar encountered difficulty attempting to fit his derived temporal power spectra to experimental data [9]. The failure of predicted and measured power spectra to match indicates that any regularization attempting to use temporal statistics could potentially have a significant negative impact on image reconstruction. As a result, no information regarding the temporal evolution of the atmospheric aberration is used during reconstruction.

5.3.4 Optimization Methodology. Applying both the PP-data regularization term of (5.14) and the phase regularization term of (4.79), the regularized objective function for dynamic aberrations is given by

$$\mathcal{L}(\mathbf{o},\boldsymbol{\alpha}) = -\sum_{f,m,x,y}\left[\ln\left(g_m^f(x,y)\right) + \frac{d_m^f(x,y)}{g_m^f(x,y)}\right] + \frac{1}{2}\sum_f \left(\vec{\alpha}^f\right)^T \boldsymbol{\Gamma}_z^{-1}\vec{\alpha}^f \tag{5.27}$$

where as before $m \in \{FP, DP, PP\}$. Once again, an efficient large-scale solution method will be required to obtain the desired result. Aside from the introduction of the time-varying aberation, the first term in (5.27) is identical (5.15), and as a result the gradients will be essentially the same. The only impact of the dynamic aberration is to require computing the gradient of $g_m^f(x,y)$ with respect to the various unknowns for each frame.

As was the case with a static aberration, the objective function is ill-behaved for points when the model object approaches zero. As a result, the same bias term, b, that was used previously is again introduced to better condition the objective. The resulting modified objective function is given by

$$\mathcal{L}(\mathbf{o},\boldsymbol{\alpha}) = -\sum_{f,m,x,y}\left[\ln\left(g_m^f(x,y)+b\right) + \frac{d_m^f(x,y)+b}{g_m^f(x,y)+b}\right] + \frac{1}{2}\sum_f \left(\vec{\alpha}^f\right)^T \boldsymbol{\Gamma}_z^{-1}\vec{\alpha}^f \tag{5.28}$$

The modified gradient of the first term in (5.28) is the same as that given for static-aberration case. Because the second term in (5.28) doesn't include b, its gradient is as given in (4.81).

VI. APDI Results

With the APDI algorithm complete, the next logical step is implementation and performance characterization. Because the derived algorithm doesn't have closed form estimates for the object and aberration, an analytical characterization was impractical. Consequently, a Monte Carlo approach was used to characterize algorithm performance. The estimator was implemented using the L-BFGS-B optimization package [11] with a modified convergence tolerance.

6.1 Chapter Overview

Based on a limited set of initial simulations, optimal conditioning bias and convergence tolerance appeared mutually independent, and independence was assumed throughout. A near-optimal convergence tolerance was selected based on preliminary simulations and used with a set of 100 data realizations to determine the optimal bias under both static and dynamic aberration conditions. Optimal biases for the static and dynamic aberration cases were found to be $b = 10\%$ and $b = 50\%$ of the average intensity respectively.

Once the optimal bias was established, the optimal convergence tolerance was established under a variety of conditions including SNR, source object, array size, and number of data frames. The optimal convergence tolerance was found to be essentially insensitive to SNR, source object, and the number of data frames. However, the optimal tolerance was found to be strongly dependent on the size of the detector arrays. Optimal tolerance values for the static and dynamic aberration cases for the baseline 64×64 configuration were found to be $\gamma_{tol} \leq 6 \times 10^{-8}$ and $\gamma_{tol} = 2 \times 10^{-7}$ respectively.

After determining approximately optimal system parameters, algorithm performance for SNR values in the range $2 \leq \text{SNR} \leq 20$ with 10, 20, 30, 40, and 50-frame data sets for both Gaussian and Poisson dominated detection noise was evaluated. The algorithm proved to be insensitive to detection noise for SNR ≥ 7, and still performed relatively well for SNR values as low as 2. The selection of either Gaussian or Poisson dominated noise sources made little difference in overall performance. As expected, adding additional frames to the data set resulted in better reconstructions. For the static aberration case, only slight gains were seen for more than 20 frames as the noise was averaged out. For the dynamic aberration case, as additional frames were added the image improved rapidly, but the improvement with additional frames began to taper off for more than approximately 30 frames. While the improvement tapered off, the increase in computational burden climbed nearly linearly with the introduction of additional frames.

To quantify the impact of PP data on algorithm performance, variants of the APDI algorithm that excluded either the PP or DP data were compiled. The PP-FP only and DP-FP only variants were then run along with the original APDI algorithm on a set of 100 simulated data realizations. Performance of the PP-FP only and DP-FP only configurations were then compared with the performance of the full APDI algorithm. As expected, the complete APDI algorithm resulted in the best reconstructions. Reconstructions done without PP data included resulted in 54% and 148% increases in average MSE compared with the Full APDI algorithm for the static and dynamic aberration cases respectively. This shows that the PP data has a dramatic impact on overall performance of the reconstruction. Exclusion of the DP data from the APDI algorithm resulted in only a 11% and 24% increase in MSE for the static and dynamic aberration cases respectively.

Finally, the impact of the aberration regularization term developed by Thelen *et al.* [48] was determined by compiling a version of the APDI algorithm with the regularization term disabled. Both the aberration-regularized and unregularized algorithms were then run on a collection of 100 simulation data sets. The results showed that addition of the regularization term resulted in neither a better reconstruction nor a lower computational burden.

This chapter is laid out as follows: section 6.2 describes algorithm implementation, section 6.3 outlines data simulation techniques that were used, section 6.4 defines the performance metrics used for analysis, section 6.5 describes results from including the aberration regularization, section 6.6 shows the effect of the conditioning bias on the algorithm, section 6.7 investigates the effect of system parameters on convergence tolerance, section 6.8 shows the benefits and costs incurred by including additional Zernike modes in the aberration model, section 6.9 sows the impact of detection noise and the number of frames on algorithm performance, section 6.10 shows the impact of PP and DP data on image reconstruction, and section 6.11 describes overall algorithm performance.

6.2 Algorithm Implementation

Given the large space over which optimization must be done, a pre-packaged bound-constrained limited-memory quasi-Newton's method, L-BFGS-B [11], was selected for optimization based on ease of use, demonstrated performance for large-scale problems, and the ability to efficiently constrain the solution to be non-negative. For computational efficiency, circular shifts were assumed and the FFTW3 [19] fast-Fourier-transform library was used to implement the required convolutions and correlations. The objective function and other components of the algorithm were programmed in C using the C99 standard to handle complex numbers. Code used to implement the algorithm is included in Appendix A.

For the static aberration case, much of the computational burden of the reconstruction can be lifted by collapsing the data set into a single set of averaged PP, FP, and DP data. This moves many of the computationally expensive operations outside the solving loop and increases speed, making the problem essentially a single-frame reconstruction. For implementation purposes, static-aberration data was pre-averaged before being passed to the multi-frame reconstruction algorithm as a single-frame data set.

Because the built-in stopping criteria in the L-BFGS-B algorithm depends on machine precision, and because reconstructions were done on several different machine architectures, the built-in stopping criteria terminated the iterations at different points on different machines. To establish a platform-independent stopping criteria, the L-BFGS-B algorithm was set to terminate when

$$\mathcal{L}_{old} - \mathcal{L}_{new} \leq \gamma_{tol} \left|\mathcal{L}_{old}\right| \tag{6.1}$$

where γ_{tol} was selected for optimal performance, and the L-BFGS-B algorithm ensures $\mathcal{L}_{old} \geq \mathcal{L}_{new}$.

6.3 Data Simulation

Simulated data was created in `Matlab`® by generating independent complex circular Gaussian variates with a standard deviation equal to the file-object intensity consistent with (3.25). The resulting complex field was then "propagated" to the imaging system PP using an FFT to implement far-field diffraction. The PP intensity was then computed by taking the magnitude squared of the field.

To include the effects of atmospheric turbulence, temporally uncorrelated random phase screens were generated assuming Kolmogorov statistics as described in section 4.2 using 200 Zernike modes and applied to the PP field for the FP and DP data channels. An amount of defocus phase (Zernike mode 4) sufficient to introduce 0.5λ RMS phase diversity was also applied the PP field for the diversity-channel data. The diversity and focal channel fields were then "propagated" to the DP and FP using additional FFTs. The intensities were then computed by taking the magnitude squared of the complex field.

To account for detection noise processes, both Gaussian-distributed read noise and Poisson distributed shot noise were applied to the data. Gaussian detector noise was characterized by the standard deviation σ_d. The average detection SNR including both noise processes is then defined by

$$\mathrm{SNR} = \frac{p}{\sqrt{p + \sigma_d^2}} \tag{6.2}$$

where p is the average number of photo-electrons (e^-) per pixel in the data set. To generate data with a desired detection SNR, (6.2) was inverted and the data scaled to the required average p. The scaled data was then used to generate Poisson-distributed random numbers with mean p. Uncorrelated Gaussian random numbers with a standard deviation of σ_d were then added to each pixel. For both the dynamic and static aberration cases, the δ-removed PP-PSD was pre-computed as part of the data simulation routine. Before being passed to the solver, and any negative pixels were set to zero, and the data frames were scaled such that the average value in each set of frames (FP, PP, and DP) was one. Code used to generate simulated data is included in Appendix B

6.4 Performance Metrics

Because of the large number of data realizations inherent in a Monte Carlo analysis, easily computed performance metrics are key. Although there exist a myriad of quality metrics for images, they are often subjective and require human interaction. Rather than delve into the realm of choosing an "optimal" quality metric, a simple mean-squared error metric, given by

$$\mathrm{MSE} = \frac{1}{N^2} \sum_{x,y} \left(\mathbf{o}(x,y) - \hat{\mathbf{o}}(x,y)\right)^2 \tag{6.3}$$

where $\mathbf{o}$ is the truth image, and $\hat{\mathbf{o}}$ is the resulting image estimate, was used for characterizing the reconstructed images. While it's easy to contrive scenarios where this metric would result in a poor score for an essentially good image, most (if not all) simple image metrics suffer from similar problems. Furthermore, in the few scenarios where the metric might show a poor reconstruction, the reconstructed image can be subjectively evaluated.

To demonstrate the effectiveness of MSE as an image quality metric, reconstructions with a low (1.83), medium (2.89), and high (6.86) MSE are shown in Fig. 6.1. Fig. 6.1(a) shows the truth image. Fig. 6.1(b) represents a good reconstruction, and was obtained with 50 frames of dynamically aberrated data, $\sigma_d = 6e^-$, SNR=7, and 15 Zernike modes. Fig. 6.1(c) represents an average reconstruction, and was obtained with 20 frames of aberrated data, $\sigma_d = 6e^-$, SNR=7, and 15 Zernike modes. Fig. 6.1(d) represents a poor reconstruction, and was obtained with 10 frames of aberrated data, $\sigma_d = 6e^-$, SNR=3, and 15 Zernike modes. The image clearly becomes worse as MSE increases.

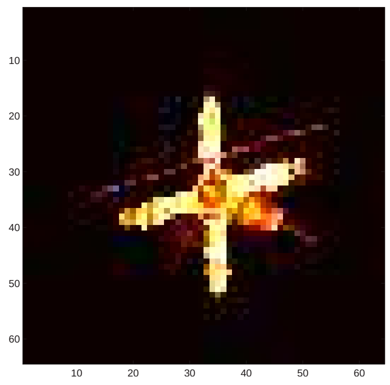
The other key performance characteristics considered were total computation time and the number of objective function evaluations required for the maximization to converge. To the extent possible, each scenario was run on the same machine architecture to facilitate comparison of computation times. However, different scenarios were run on several different machines, making comparison of computation times between them fruitless. In such scenarios, comparison of function evaluations provides a better metric.

6.5 Aberration Regularization

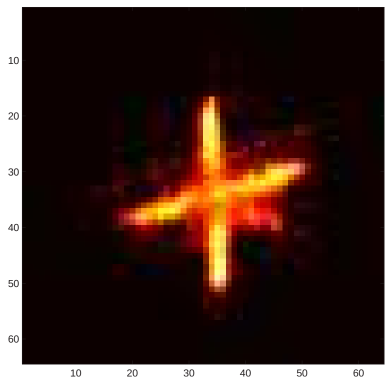
Thelen's atmospheric regularization term of (4.79) was initially implemented as part of the algorithm. However, introduction of the regularization term decreased neither the computational burden nor the residual error in the reconstruction. Because it had a net negative effect on performance, all further analysis on the APDI algorithm was done with aberration regularization disabled.

6.6 Optimal Conditioning Bias

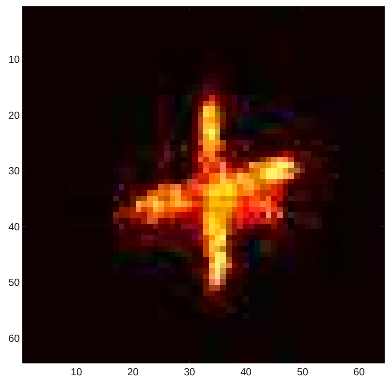
The conditioning bias was implemented as a percentage of the average data value. To determine optimal conditioning bias, 100 30-frame data sets of both static-aberration and



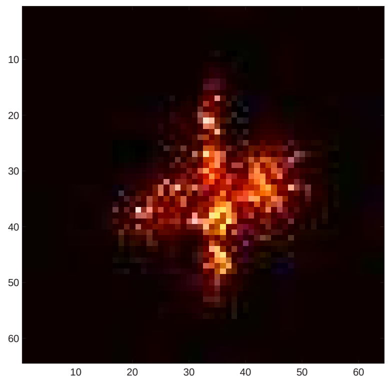
(a) Truth



(b) MSE = 1.83



(c) MSE = 2.89

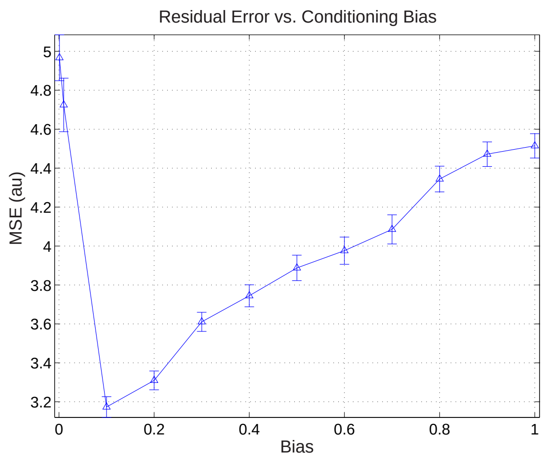


(d) MSE = 6.76

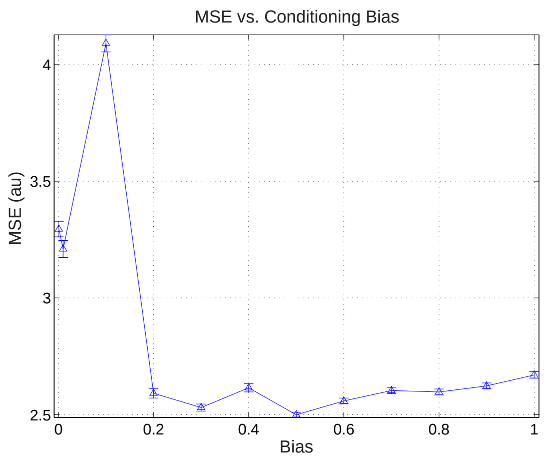
Figure 6.1: (a) Truth image, (b) MSE=1.83, (c) MSE=2.89, and (d) MSE=6.76. (b) was the lowest MSE reconstruction, (c) an average MSE reconstruction, and (d) the highest MSE reconstruction obtained for dynamic aberrations with near-optimal b and γ_{tol}.

dynamic-aberration data were generated using SNR=7 and $\sigma_d = 6e^-$. Each data set was then processed with $\gamma_{tol} = 1 \times 10^{-9}$ for static aberrations, and $\gamma_{tol} = 2 \times 10^{-7}$ for dynamic aberrations. In both the static- and dynamic-aberration cases, only 15 of the 200 Zernike modes were included in the reconstruction.

Simulation results for MSE as a function of conditioning bias are shown in Fig. 6.2. The results show that selecting too small a bias results in worse reconstructions. While a small bias produces a problem that is closer to the statistically derived expression, the objective becomes poorly conditioned, and the results are worse. Large biases also worsen the reconstruction by moving the problem away from the statistics for which it was developed. The spike in MSE for $b = 10\%$ of the average pixel value and the slight rise for $b = 40\%$ seen for dynamic aberrations in Fig. 6.2(b) are unexplained. However, for the purposes of producing low-MSE images, biases of $b = 10\%$ and $b = 50\%$ appear optimal for the static and dynamic aberration cases respectively. The difference between optimal biases for the static and dynamic cases likely stems from the noise reduction that results when static-aberration data frames are pre-averaged. This reduction in effective noise serves to better condition the problem and allow for a smaller bias before negative impacts are seen.



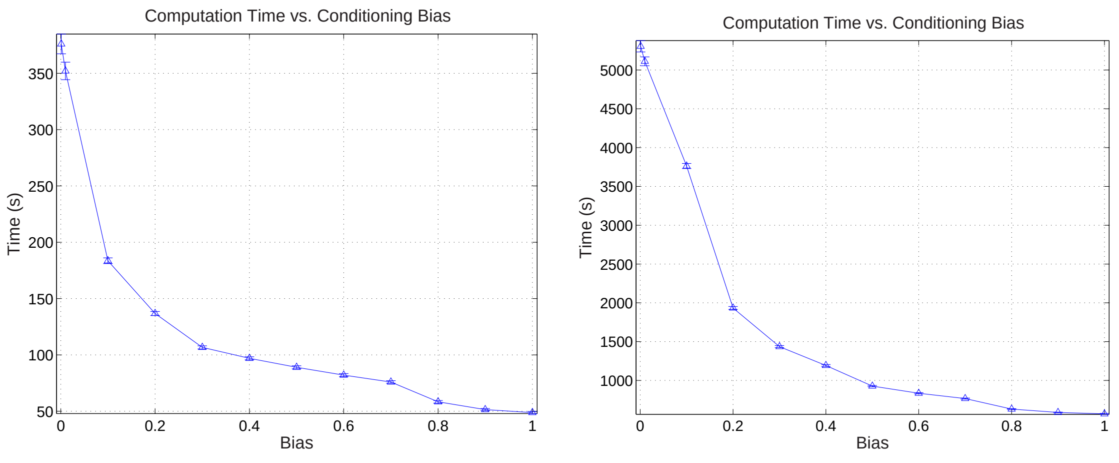
(a) Static Aberration



(b) Dynamic Aberration

Figure 6.2: Residual MSE versus conditioning bias b. Residual MSE increases for small biases as the problem becomes less well-conditioned. MSE also increases for larger biases as the problem moves away from the statistically derived objective.

Computation time as a function of conditioning bias is shown in Fig. 6.3. For both static and dynamic aberrations the computation time drops almost exponentially with increasing bias. As the bias increases and the problem becomes better conditioned, the solver requires fewer and fewer iterations to converge to a solution. If computation time is paramount, a large decrease in time can be obtained for a relatively small increase in residual MSE by simply increasing the conditioning bias. This is particularly true for the dynamic aberration case.



(a) Static Aberration (b) Dynamic Aberration

Figure 6.3: Computation time versus conditioning bias b. Computation time decreases almost exponentially with increasing bias as the problem becomes better conditioned.

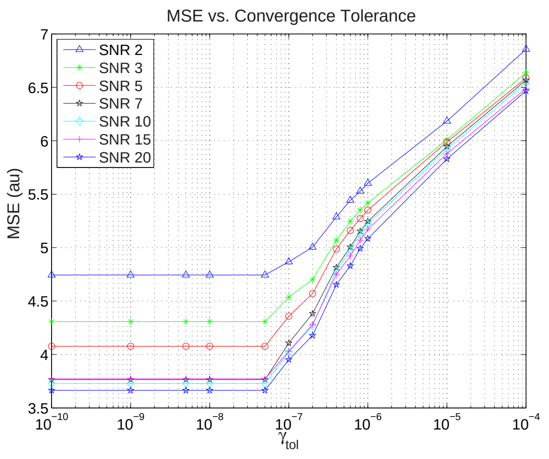
6.7 Optimal Stopping Tolerance

With an established conditioning bias, the next step is characterization of the effect that convergence tolerance has on performance. The effects of detection SNR, number of frames, size of the data array, and object being imaged on the optimal convergence tolerance were all investigated.

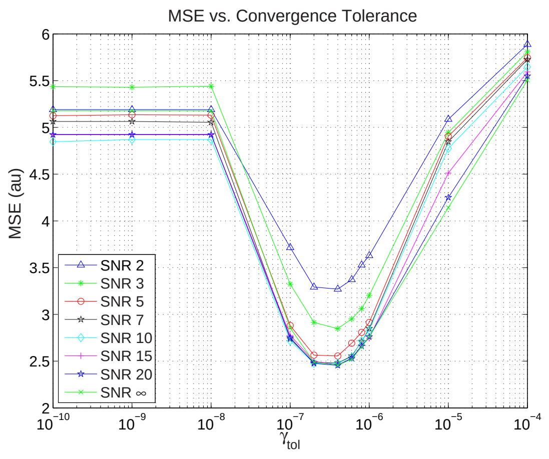
6.7.1 Detection SNR. The effects of detection SNR on the optimal stopping tolerance were quantified by generating 100 30-frame data sets with $\sigma_d = 6$ and SNR equal to 2, 3, 5, 7, 10, 15, 20, and ∞. The simulated data sets were then post processed using 15 Zernike modes, $b = 0.5$ and convergence tolerances ranging from $\gamma_{tol} = 1 \times 10^{-10}$ to $\gamma_{tol} = 1 \times 10^{-4}$. The resulting MSE curves are shown in Fig. 6.4. As can be seen, the only significant impact of detection SNR on the curves is a shift toward higher MSE with decreasing SNR below SNR=7.

As shown in Fig. 6.5 and Fig. 6.6, computation time and function evaluations were only moderately impacted by variations in SNR.

6.7.2 Number of frames. To investigate the effect of additional data frames on optimal stopping tolerance, 100 data realizations with 10, 20, 30, 40, and 50 frames were

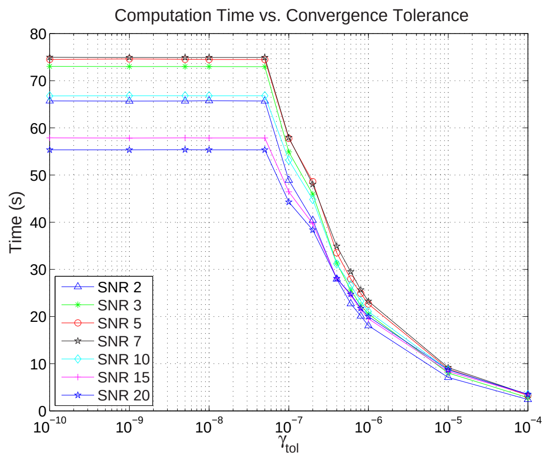


(a) Static Aberration

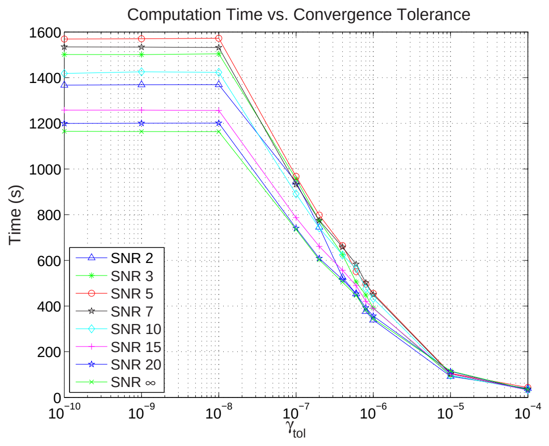


(b) Dynamic Aberration

Figure 6.4: MSE vs. convergence tolerance and SNR. The optimal convergence tolerance doesn't shift significantly with changes in detection SNR.

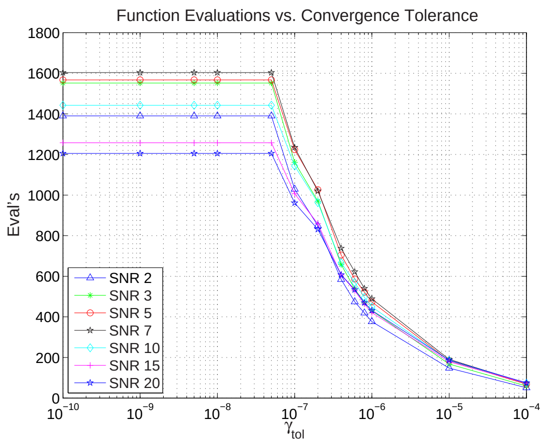


(a) Static Aberration

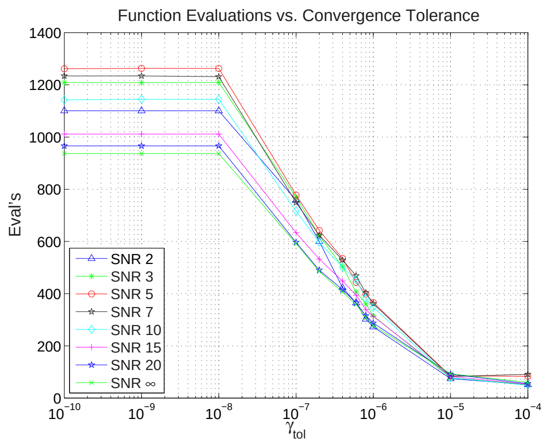


(b) Dynamic Aberration

Figure 6.5: Computation time vs. convergence tolerance and SNR. Computation time shows a slight dependence on detection SNR.



(a) Static Aberration

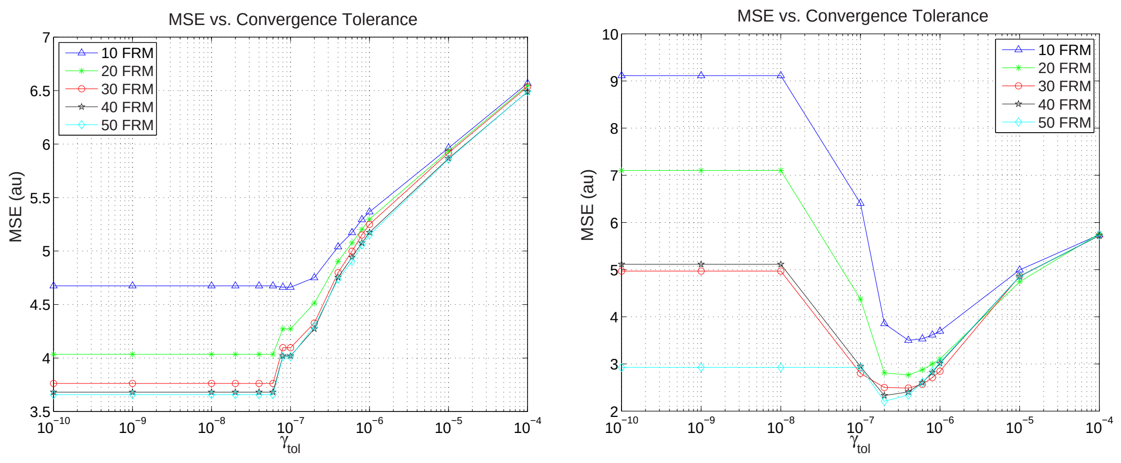


(b) Dynamic Aberration

Figure 6.6: Function Evaluations vs. convergence tolerance and SNR. Function evaluations show a slight dependence on detection SNR.

generated with SNR=7 and $\sigma_d = 6e^-$ for both static and dynamic aberrations. These data sets were then post processed allowing for 15 Zernike modes and with $b = 0.5$. Residual MSE as a function of γ_{tol} is shown in Fig. 6.7. As shown in Fig. 6.7(a), there is no penalty for selecting tolerance values smaller than the optimal value of $\gamma_{tol} \approx 6\times10^{-8}$. This is due to the fact that at this point, the solver reaches a point where $\mathcal{L}_{old} = \mathcal{L}_{new}$, and the iterations stop regardless of γ_{tol}. However, for the dynamic aberration case shown in Fig. 6.7(b), tolerances smaller than $\gamma_{tol} \approx 2 \times 10^{-7}$ result in increased MSE.

Another notable feature of Fig. 6.7(b) is that the relative deviation of MSE from its optimal value caused by setting γ_{tol} too small is a strong function of the number of data frames. The increase in MSE as γ_{tol} goes lower than the optimal value is likely because the algorithm begins fitting the reconstruction to noise in the data. Visual inspection of the results support this hypothesis by showing that the reconstructed image becomes more and more speckled as the solver goes beyond the optimal result. As more and more frames are added the effect of noise becomes less and less pronounced, and the reconstruction results in a smaller deviation from optimum MSE values. A close inspection of the 10 frame line for static aberrations in Fig. 6.7(a) shows the beginnings of the same rise in MSE with decreasing γ_{tol}, bearing out the notion that the rise is due to excess noise in the data.



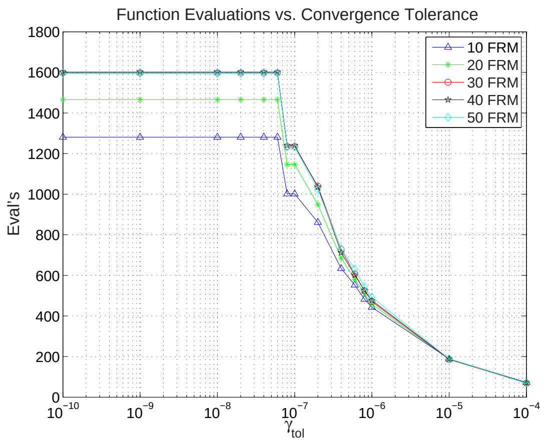
(a) Static Aberration

(b) Dynamic Aberration

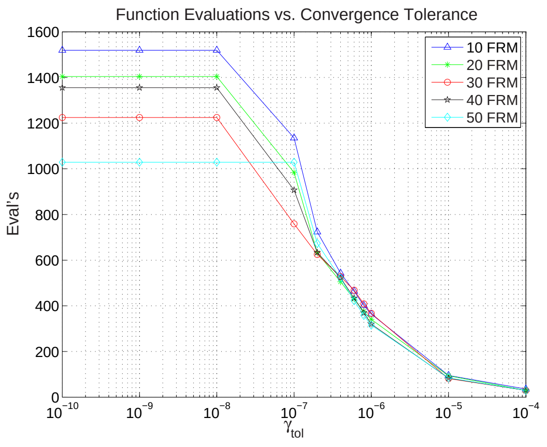
Figure 6.7: MSE vs. convergence tolerance for 10, 20, 30, 40, and 50 frame data sets. (a) The number of frames has little or no effect of optimal γ_{tol} for static aberrations. There is no penalty for selecting a tolerance smaller than the optimum. (b) The number of frames has little effect on the optimal tolerance for dynamic aberrations as well, with only a slight shift for less than 20 frames. If the tolerance is set too low, the algorithm begins fitting to noise and results in much worse reconstructions.

The number of function evaluations and convergence times for these computations are shown in Fig. 6.8 and Fig. 6.9 respectively. As can be seen in Fig. 6.8(b) and Fig. 6.9(b), judicious selection of convergence tolerance for the dynamic aberration case not only significantly reduces the MSE, but can also reduce the number of function evaluations and associated computation time by as much as 50%. The static aberration results of Fig. 6.8(a) and Fig. 6.9(a) indicate that as much as a 25% decrease in computational cost can be obtained for no more than a 10% increase in MSE.

6.7.3 Array Size. To see the effect of detector array size on optimal convergence tolerance, a 128 × 128 pixel object was resampled to 64 × 64, and 32 × 32 pixels. Each of the objects was then used to generate 100 30-frame data sets for both static and dynamic aberrations using SNR=7 and $\sigma_d = 6e^-$. These data sets were then post-processed allowing for 15 Zernike modes and setting $b = 0.5$. The resampled objects are shown in Fig. 6.10, and the resulting MSE in Fig. 6.11. For static aberrations, the convergence tolerance appears to be essentially insensitive to array size, but for dynamic aberrations the optimal tolerance

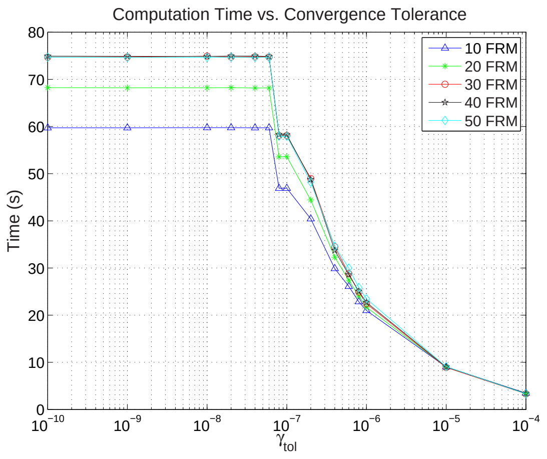


(a) Static Aberration

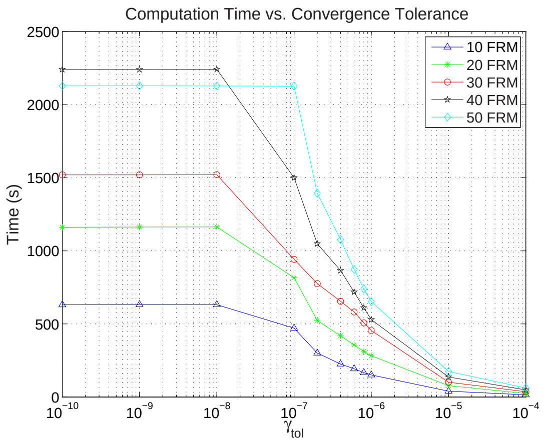


(b) Dynamic Aberration

Figure 6.8: Function evaluations vs. γ_{tol} and frames for (a) static aberrations and (b) dynamic aberrations. The number of frames has only a marginal effect on optimal convergence tolerance.



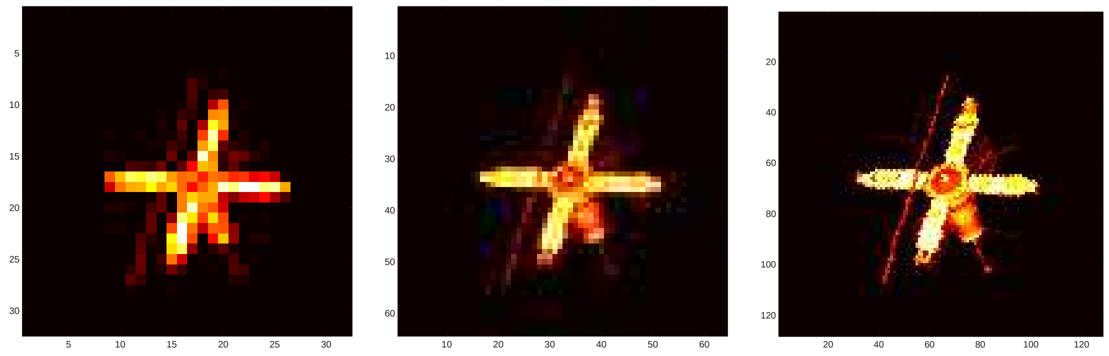
(a) Static Aberration



(b) Dynamic Aberration

Figure 6.9: Computation time vs. γ_{tol} and frames for (a) static aberrations and (b) dynamic aberrations. The number of frames has a significant impact on computation time due to the increase in computational overhead for the dynamic aberration case.

changes by as much as two orders of magnitude between the 128×128 and 32×32 pixel images, generally shifting toward larger values for smaller data arrays.



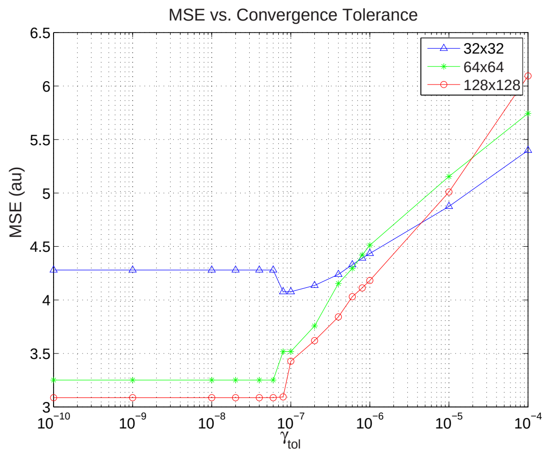
(a) 32×32 (b) 64×64 (c) 128×128

Figure 6.10: Truth objects used for evaluating the effect of object size on optimal convergence tolerance.

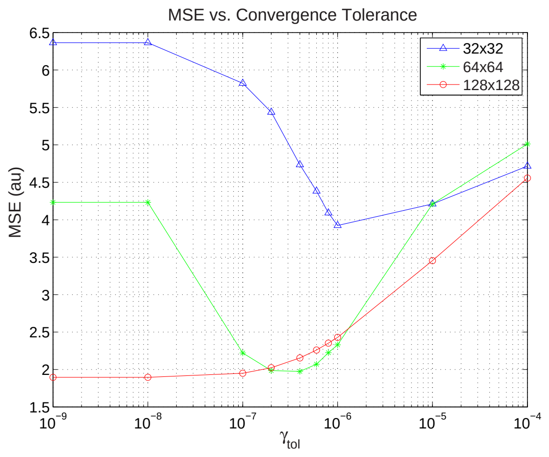
The number of function evaluations and computation times for the varying object sizes are shown in Fig. 6.12 and Fig. 6.13 respectively. As expected, both the number of function evaluations and the computation time are strongly dependent on the object size. As the array size increases, the number of unknowns grow and the optimization routine takes more iterations to find the maximum. In addition to the increased number of function evaluations required for convergence, the computational cost of each evaluation scales with the image size.

6.7.4 Source Object. To verify that the optimal convergence tolerance is not impacted by the source object, four different objects (shown in Fig. 6.14) were each used to generate 100 30-frame data sets with SNR=7 and $\sigma_d = 6e^-$ for both static and dynamic aberrations. These data sets where then post-processed using 15 Zernike modes and $b = 0.5$. Fig. 6.15 shows that while the absolute MSE varies, the location of the optimum convergence tolerance is essentially independent of the truth object, and remains within a factor of one to two for all the tested objects.

Computation time and function evaluations as a function of γ_{tol} for the test objects are shown in Fig. 6.16 and Fig. 6.17. The number of function evaluations, and consequently

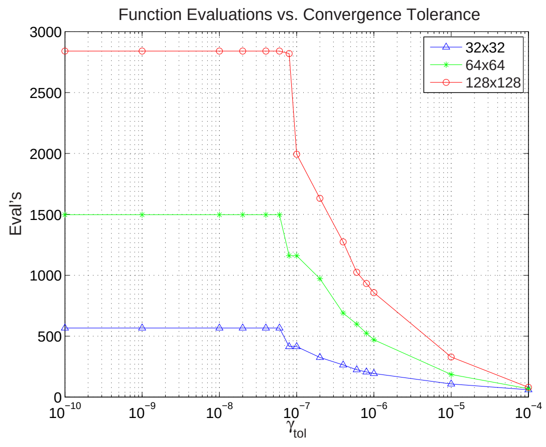


(a) Static Aberration

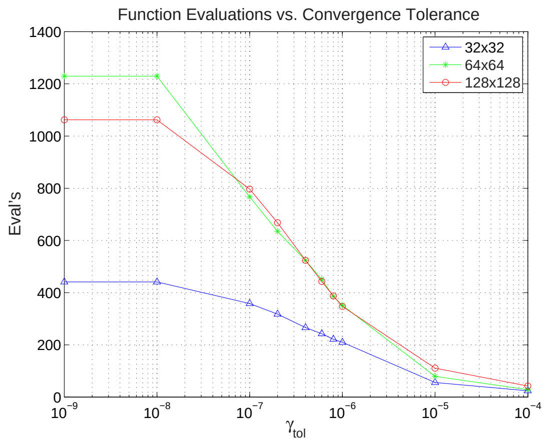


(b) Dynamic Aberration

Figure 6.11: MSE vs. convergence tolerance for the file objects of Fig. 6.10 for (a) static aberrations, and (b) dynamic aberrations. Optimal convergence tolerance for static aberrations appears essentially insensitive to the size of the data array, while the dynamic aberration case shows a strong dependence.

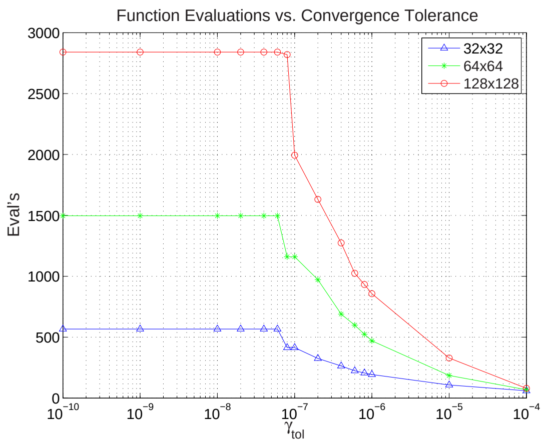


(a) Static Aberration

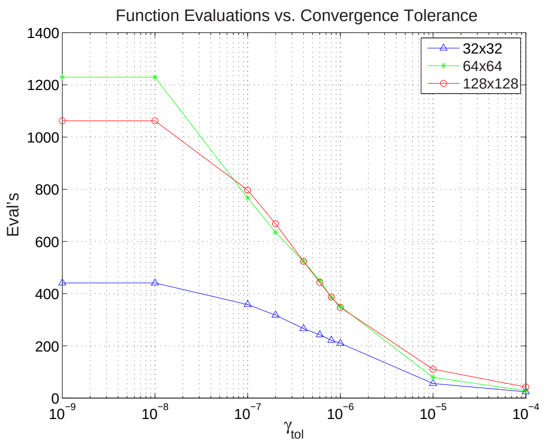


(b) Dynamic Aberration

Figure 6.12: Function evaluations vs. γ_{tol} and array size for (a) static aberrations and (b) dynamic aberrations. As expected, the increase in unknowns associated with larger arrays has a significant impact on the number of iterations.



(a) Static Aberration



(b) Dynamic Aberration

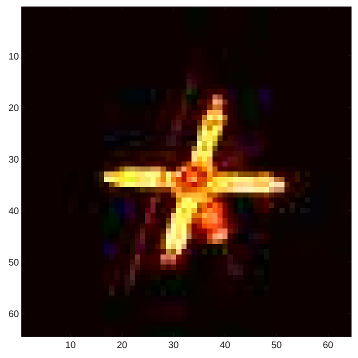
Figure 6.13: Computation Time vs. γ_{tol} and array size for (a) static aberrations and (b) dynamic aberrations. As expected, the increase in unknowns associated with larger arrays has a significant impact on computation time.

the computation time, show a significant dependence on the object being imaged. However, the optimal tolerance doesn't appear to be similarly impacted.

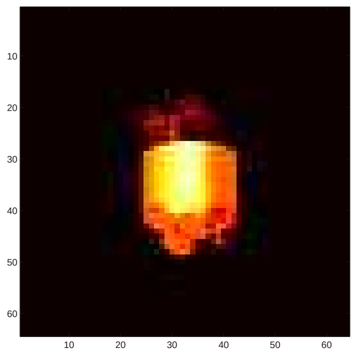
6.8 Corrected Modes

To understand the trade-offs inherent in compensating additional aberration modes, 100 sets of 30-frame data were generated using SNR=7 and $\sigma_d = 6$. These data sets were then post-processed with $b = 0.5$, and $\gamma_{tol} = 1 \times 10^{-9}$ for static aberrations and $\gamma_{tol} = 2 \times 10^{-7}$ for dynamic aberrations. Each data set was processed allowing for 5, 10, 15, 20, 30, 50, and 100 compensated Zernike modes. The resulting MSE is shown in Fig. 6.18. In both cases, MSE drops rapidly as more and more modes are compensated until approximately 30 modes are included. After this point, MSE rises again. The reduction in MSE is consistent with the results expected as the model of the blurring aberration becomes more complete. However, by the time 50 modes are included, the mode variance for high-order modes is approximately four orders of magnitude below that of the low-order modes, and the algorithm is likely fitting to noise in the data.

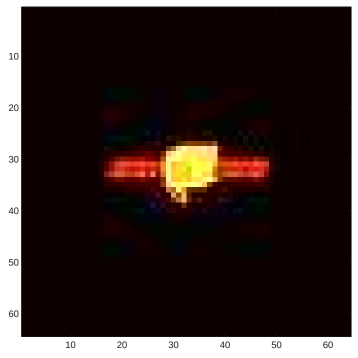
As shown in Fig. 6.19 and Fig. 6.20, the number of corrected modes has a dramatic effect on the computation time, and to a lesser extent on the number of function evaluations



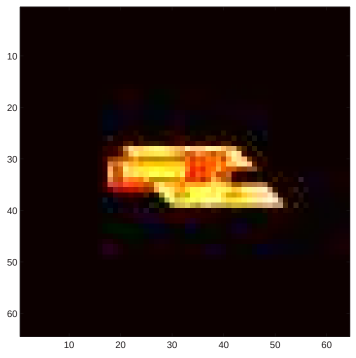
(a) OBJ 1



(b) OBJ 2

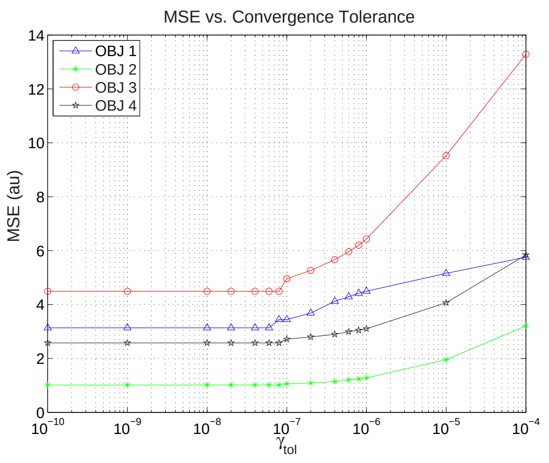


(c) OBJ 3

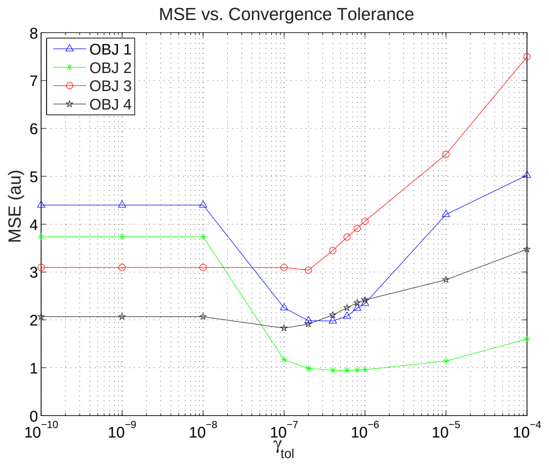


(d) OBJ 4

Figure 6.14: Truth objects used for evaluating the effect of the object on optimal convergence tolerance.

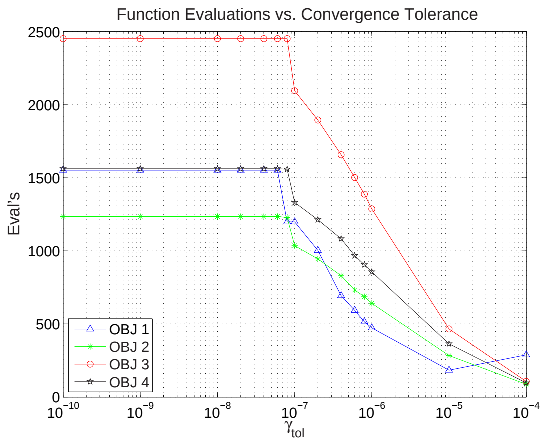


(a) Static Aberration

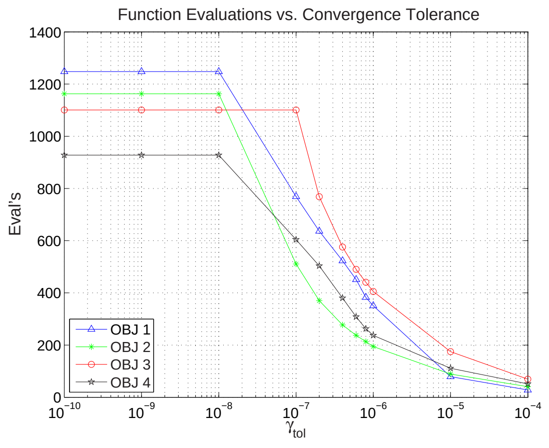


(b) Dynamic Aberration

Figure 6.15: MSE vs. convergence tolerance for the file objects of Fig. 6.14 for (a) static aberrations, and (b) dynamic aberrations. Optimal convergence tolerance appears essentially insensitive to the object being imaged.

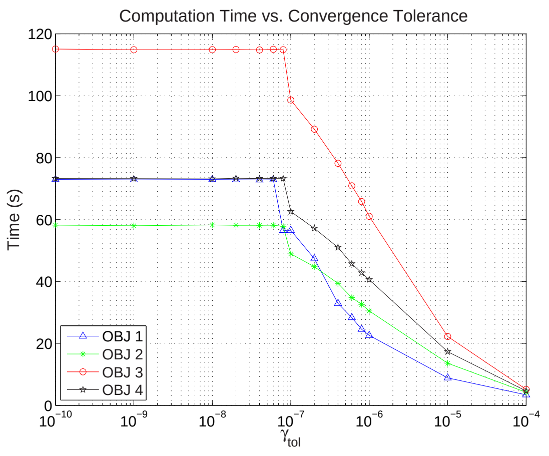


(a) Static Aberration

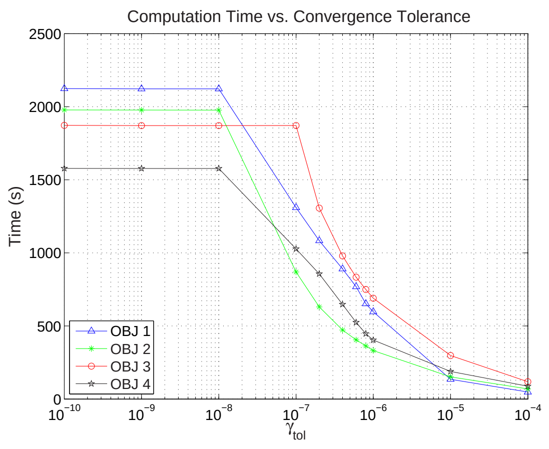


(b) Dynamic Aberration

Figure 6.16: Function evaluations vs. γ_{tol} and truth object for (a) static aberrations and (b) dynamic aberrations. The truth object has a moderately strong impact on the number of evaluations required for convergence.

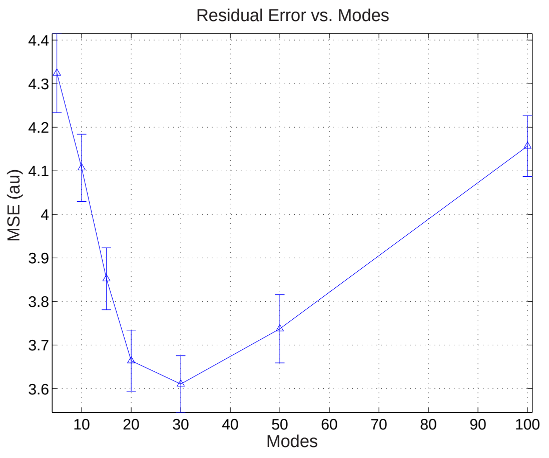


(a) Static Aberration

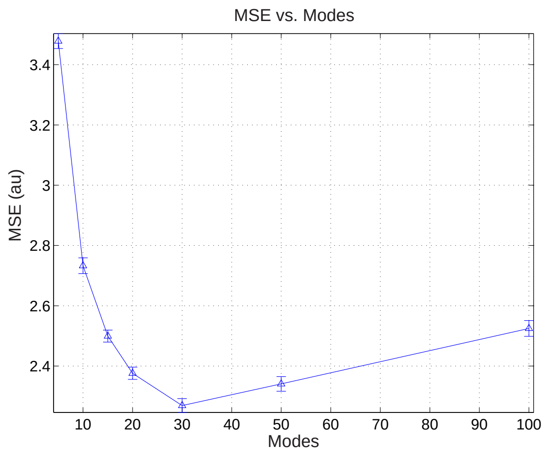


(b) Dynamic Aberration

Figure 6.17: Computation time vs. γ_{tol} and truth object for (a) static aberrations and (b) dynamic aberrations. The truth object has a moderately strong impact on computation time.



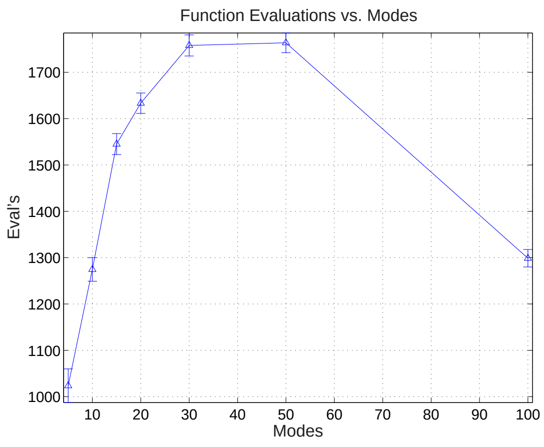
(a) Static Aberration



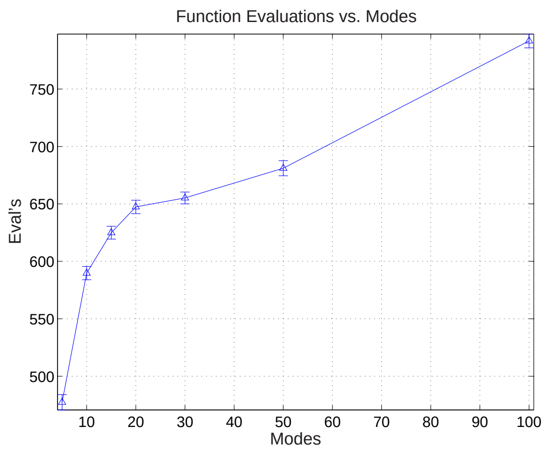
(b) Dynamic Aberration

Figure 6.18: MSE initially drops with increasing modes, then slowly climes beyond 30 modes for (a) static aberrations, and (b) dynamic aberrations.

required for convergence. Because a great deal of the computational time is spent computing the gradients with respect to the aberration parameters, the increase in computation time is expected. The increase in function evaluations can be attributed to the increased dimensionality of the parameter space. The dip in function evaluations for 100 modes with static aberrations is likely due to the optimizer terminating iterations prematurely.



(a) Static Aberration



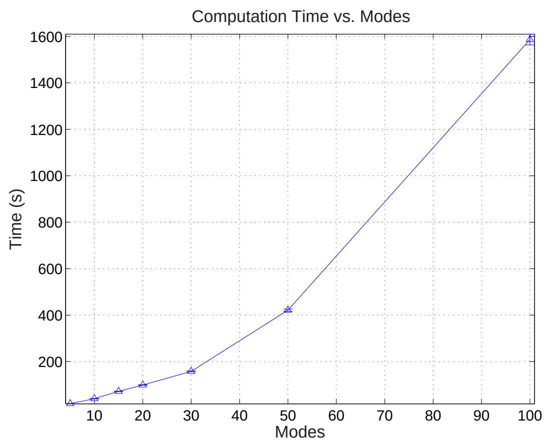
(b) Dynamic Aberration

Figure 6.19: Function evaluations vs. modes for (a) static aberrations and (b) dynamic aberrations. Increasing the number of included modes initially improves MSE at the cost of a moderate increase in function evaluations.

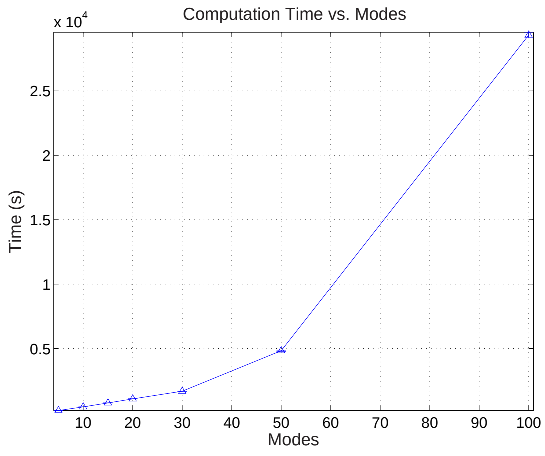
6.9 Number of Frames and Dominant Detection Noise

To characterize the effects of the number of data frames and type and severity of detection noise sources, 100 static and dynamic aberration data sets with 10, 20, 30, 40, and 50 frames were generated with SNR values of 2, 3, 5, 7, 10, 15, and 20 using both $\sigma_d = 6e^-$ and $\sigma_d = 100e^-$. The static aberration data sets were post processed using 15 Zernike modes, $b = 0.1$ and $\gamma_{tol} = 1 \times 10^{-8}$. Dynamic aberration data sets were post-processed using 15 Zernike modes, $b = 0.5$, and $\gamma_{tol} = 2 \times 10^{-7}$. Results for the static aberration case are shown in Fig. 6.21 and Fig. 6.23. Dynamic aberration results are shown in Fig. 6.22 and Fig. 6.24.

For both the static and dynamic aberration data sets, there was little difference between the Poisson dominated ($\sigma_d = 6e^-$) and Gaussian dominated ($\sigma_d = 100e^-$) detection



(a) Static Aberration

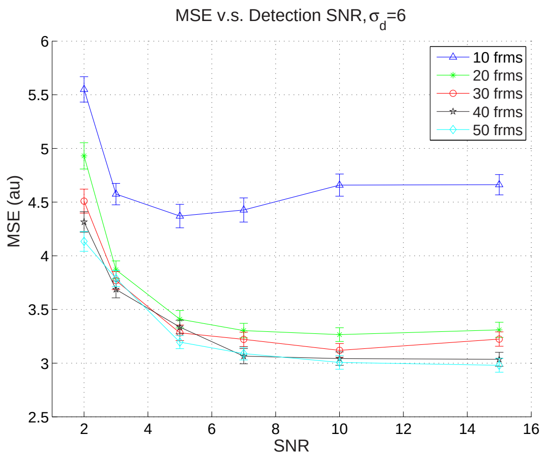


(b) Dynamic Aberration

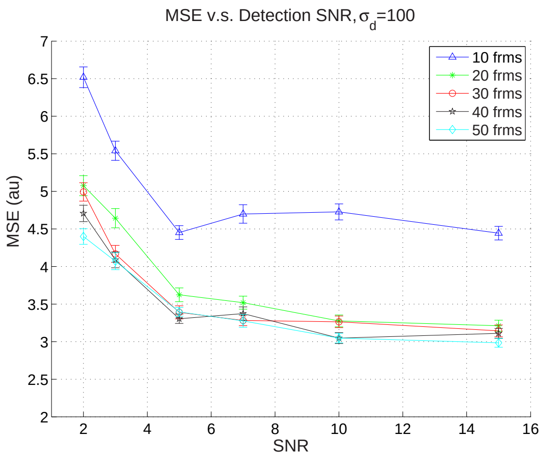
Figure 6.20: Computation time vs. modes for (a) static aberrations and (b) dynamic aberrations. Increasing the number of included modes initially improves MSE at the cost of a significant increase in computation time caused by the computational burden of including the extra modes in the system model.

noise. The slight improvement seen for the static-aberration 10-frame Poisson-dominated case is likely due to the fact that for low signal levels, Poisson statistics are closer than Gaussian statistics to the exponential distribution for which the algorithm was derived. While working with static aberrations, there is little benefit gained by adding more than 20 frames to the dataset. Adding additional frames to the dynamic aberration data sets has a more significant effect. Though the magnitude of the MSE reduction tapers off, there continues to be a significant reduction through 50 frames.

Another notable feature is the overall lower MSE and tighter error bars obtained with dynamic aberration data compared with static aberration data. This can be understood by considering incoherent phase diversity imaging [39] and multi-frame blind deconvolution [42]. With moderate to strongly aberrations, multi-frame blind deconvolution tends to produce better results than phase diversity. With static aberrations, the APDI algorithm is essentially an incoherent phase diversity algorithm with additional data included. When the aberration changes in time, the system begins to look more like a multi-frame deconvolution with the associated increase in performance.

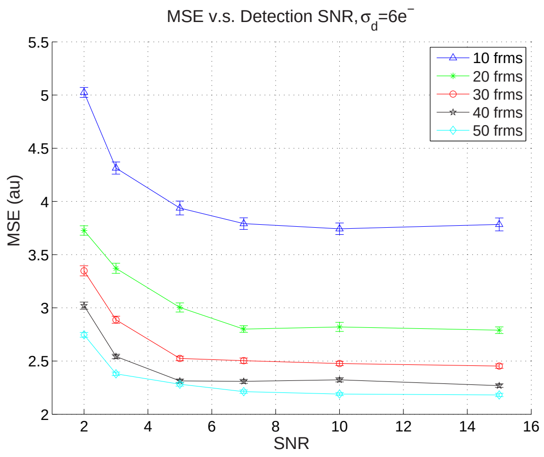


(a) $\sigma_d = 6e^-$

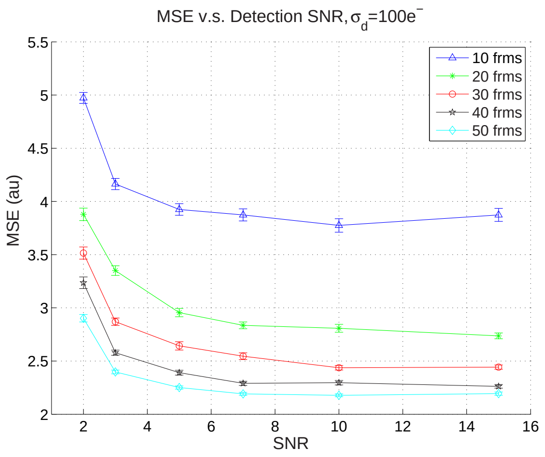


(b) $\sigma_d = 100e^-$

Figure 6.21: Static aberration MSE vs. SNR for 10, 20, 30, 40, and 50 frame data sets. (a) represents Poisson and (b) Gaussian dominated detection noise. For SNR ≥ 7 there is little improvement in MSE, and little is gained by averaging more than 20 frames of data. Low SNR performance appears better for Poisson than Gaussian dominated noise, but results in similar results for higher SNR values.

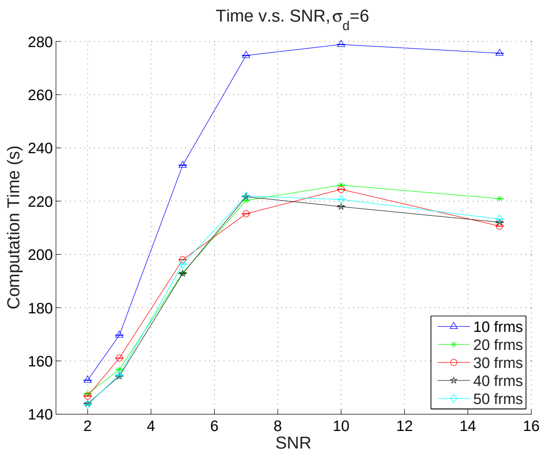


(a) $\sigma_d = 6e^-$

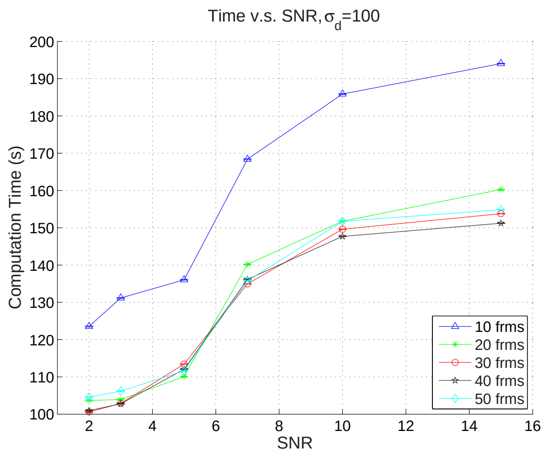


(b) $\sigma_d = 100e^-$

Figure 6.22: Dynamic Aberration MSE vs. SNR for 10, 20, 30, 40, and 50 frame data sets. (a) represents Poisson and (b) Gaussian dominated detection noise. For SNR ≥ 7 there is little, if any, improvement in MSE. Results for the Poisson and Gaussian dominated cases are statistically indistinguishable. Adding extra frames to the dataset initially decrease MSE rapidly, but the reduction tapers off for more than approximately 30 frames.

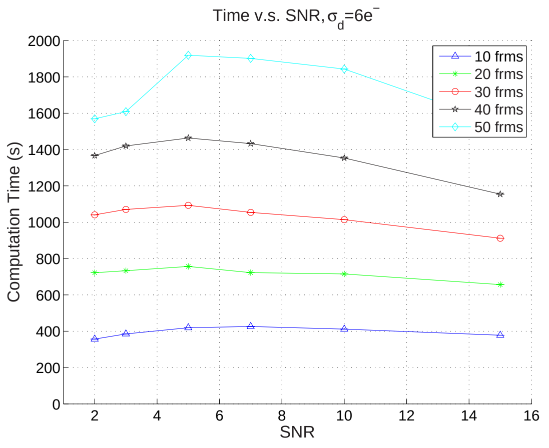


(a) $\sigma_d = 6e^-$

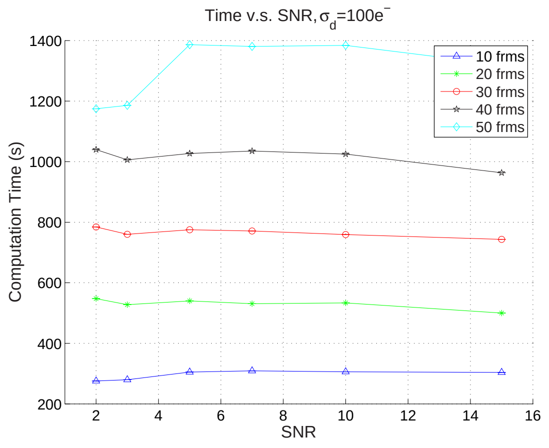


(b) $\sigma_d = 100e^-$

Figure 6.23: Static aberration computation time vs. SNR. Aside from the 10 frame case, additional frames have little or no impact on computation time.



(a) $\sigma_d = 6e^-$

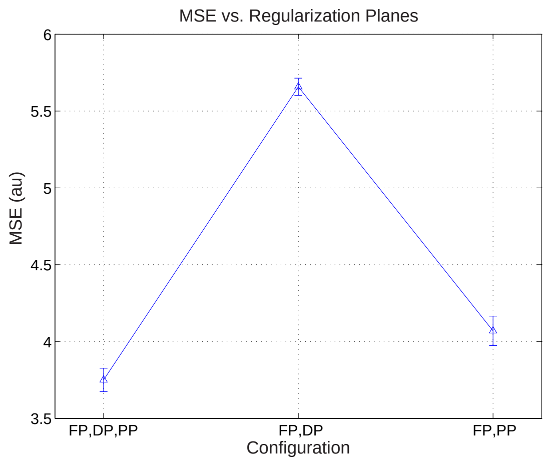


(b) $\sigma_d = 100e^-$

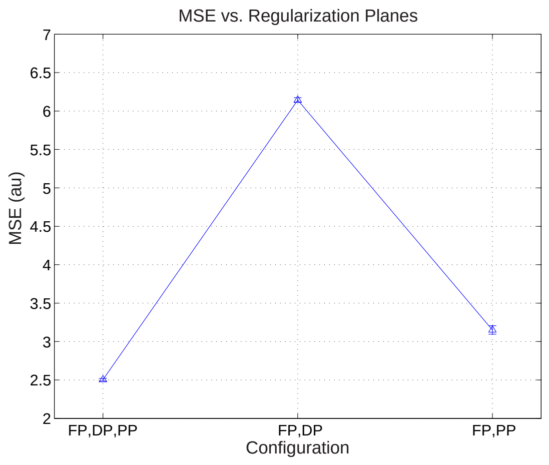
Figure 6.24: Dynamic aberration computation time vs. SNR. SNR has only a slight impact on computation time. The overall scale difference between the $\sigma_d = 6e^-$ and $\sigma_d = 100e^-$ cases is because the two scenarios were run on different computer architectures.

6.10 *Impact of PP and DP Data Planes on Reconstruction*

One of the most radical changes from conventional phase-diversity imaging was the introduction of PP data. Because introduction of an additional data channel implies additional hardware and the lower light levels associated with splitting off another channel, it's important to quantify the improvement offered by adding the new channel. To test the effects of the PP data, 100 data realizations were generated with 30 frames, SNR=7, and $\sigma_d = 6e^-$. The resulting datasets were then processed with 15 compensated Zernike modes, $b = 0.5$, and $\gamma_{tol} = 1 \times 10^{-9}$ and $\gamma_{tol} = 2 \times 10^{-7}$ for static and dynamic aberrations respectively using the full APDI algorithm, the APDI algorithm including just conventional FP and DP data, and with the APDI algorithm using just FP and PP data. The resulting MSE as a function of data planes is shown in Fig. 6.25.



(a) Static Aberration



(b) Dynamic Aberration

Figure 6.25: Residual MSE for the full algorithm, without the DP data, and without the PP data for (a) static aberrations, and (b) dynamic aberrations.

Because of the large amount of information in the PP intensity that hasn't been corrupted by the unknown atmosphere, the PP data has a much more significant impact on system performance than the conventional DP data. For both the static and dynamic aberration cases, the residual MSE was only slightly higher for the FP-PP configuration than it was for the complete FP-DP-PP configuration. Reconstructions done without PP data included resulted in 148% and 54% increases in average MSE compared with the Full

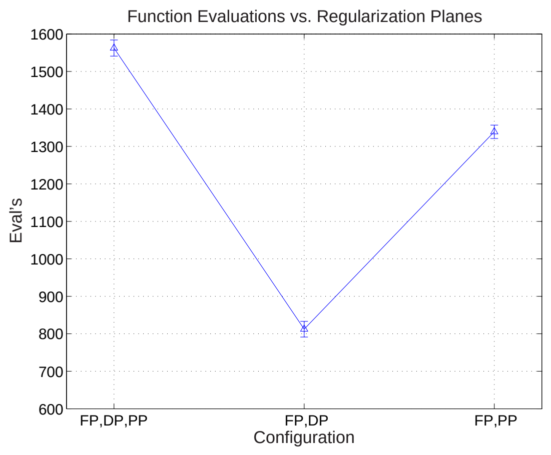
APDI algorithm for the dynamic and static aberration cases respectively. This shows that the PP data has a dramatic impact on overall performance of the reconstruction.

Exclusion of the DP data from the APDI algorithm resulted in only a 11% and 24% increase in MSE for the static and dynamic aberration cases respectively. In cases where low-light levels force a poor SNR, the marginal gain in performance obtained by using all three data planes may be swamped by the decrease in performance caused by a low detection SNR. This indicates that a two-channel system comprised of a FP and PP sensor might be optimal for these kinds of scenarios. Even when SNR isn't a factor, the additional computational burden and hardware requirements for a complete three-channel system may not be justified by the marginal gain in system performance.

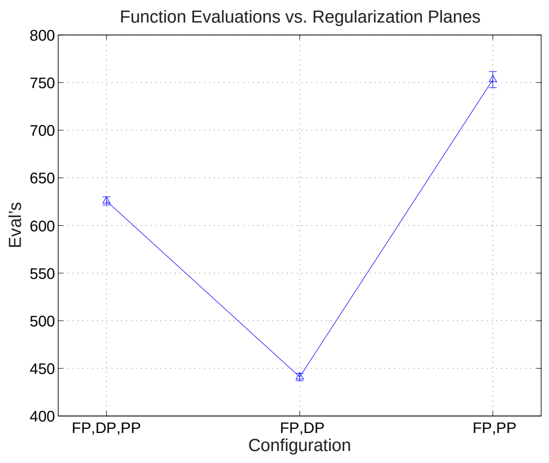
As shown in Fig. 6.26 and Fig. 6.27, in addition to the gain in low-light performance, elimination of the DP data for the dynamic aberration case also serves to reduce the computation time required for convergence. Because the PP data is insensitive to the changing aberration, it can be pre-averaged, and much of the computational burden lifted. However, the DP data depends on the atmosphere, and a reasonably large amount of computation must be performed at every iteration to include this channel. Because of this disparity, the FP-PP configuration converges faster than the other configurations in spite of requiring more function evaluations. For the static-aberration case, the data frames are pre-averaged in all channels, and the computational burden for the DP data becomes less than that of the PP data.

6.11 Overall Performance

Of interest is the reduction in MSE relative to the raw frame-averaged FP intensity. Ideally, the reconstruction should be universally better than the raw data, and result in a lower MSE for the reconstruction than for the averaged raw data. The pre- and post-processed MSE for all of the 14,000+ realizations used to generate Fig. 6.21 and Fig. 6.22 are shown in the scatter plot of Fig. 6.28. The solid line marks the point where the pre- and post-processed MSE are equal. For the static aberration case, approximately 3% of the reconstructions result in an MSE greater than the pre-processed MSE. As the averaged focal plane data becomes worse and worse, the frequency of poor reconstructions increases.

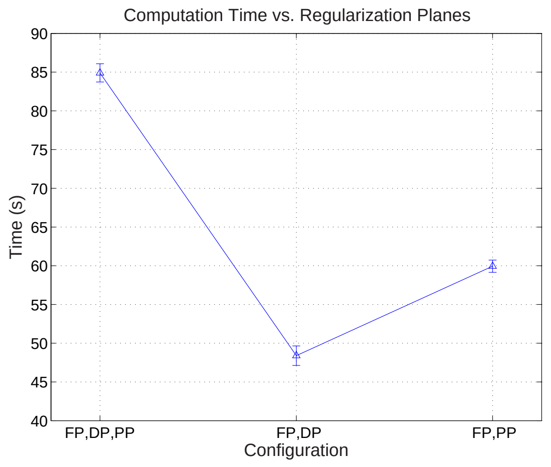


(a) Static Aberration

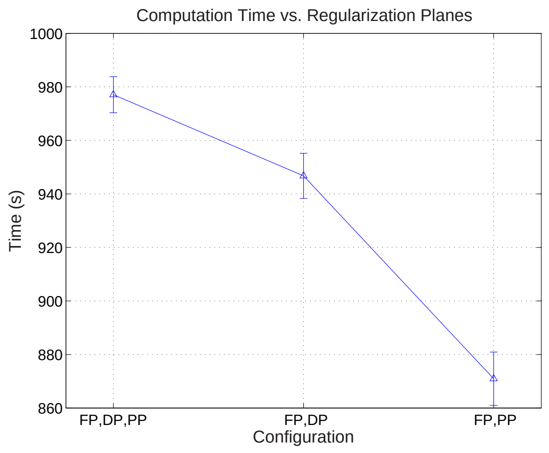


(b) Dynamic Aberration

Figure 6.26: Function evaluations vs. data planes for (a) static aberrations and (b) dynamic aberrations. For the dynamic aberration case, elimination of the DP data reduces the computational burden. The same effect is not seen for static aberration data.



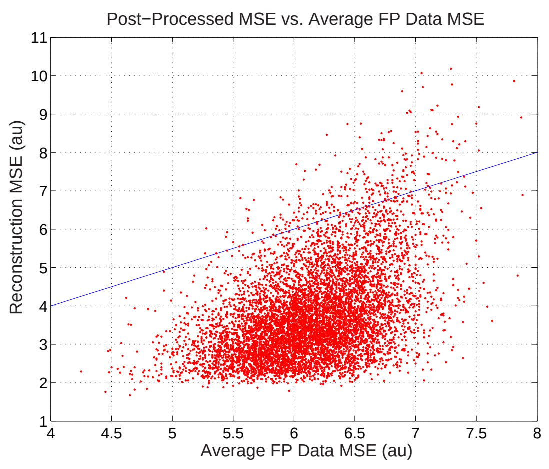
(a) Static Aberration



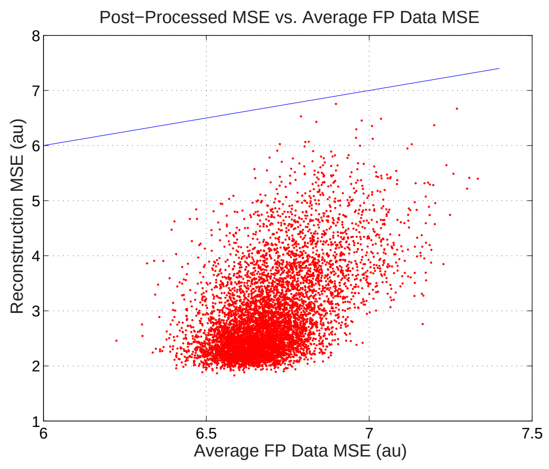
(b) Dynamic Aberration

Figure 6.27: Computation time vs. data planes for (a) static aberrations and (b) dynamic aberrations. For the dynamic aberration case, elimination of the DP data reduces the computational burden. The same effect is not seen for static aberration data.

However, in many of the cases where post-processed MSE is higher than the averaged FP MSE, visual inspection of the reconstruction shows that these points are a result of the error-metric being inadequate. In a few cases, the question of whether the or not the reconstruction is worse than the raw data becomes subjective, and it might be argued that the reconstruction diverged. For the dynamic aberration case, all of the reconstructions result in an MSE that is lower than the frame-averaged FP MSE in spite of the fact that the averaged FP data typically had a higher MSE than was the case for static aberrations.



(a) Static Aberration



(b) Dynamic Aberration

Figure 6.28: Pre- and post-processed MSE for 7,000+ (a) static aberrations, and (b) dynamic aberrations. The solid line marks the point where pre- and post-processed MSE's are equal. Approximately 3% of the reconstructions with static aberrations had a higher MSE than the raw data. The prevalence of high-MSE reconstructions increases as the averaged FP MSE increases. None of the dynamic aberration reconstructions resulted in an MSE that was higher than the raw data.

The lowest, average, and highest MSE reconstructions for static aberration data are shown in Fig. 6.29, Fig. 6.31, and Fig. 6.33 respectively. The lowest, average, and highest MSE reconstructions for dynamic aberrations are shown in Fig. 6.30, Fig. 6.32, and Fig. 6.34. The best reconstruction from static aberration data has the lowest MSE of any reconstruction done as a part of this work, represents a fairly benign aberration, and could be considered a "glory shot" for the system.

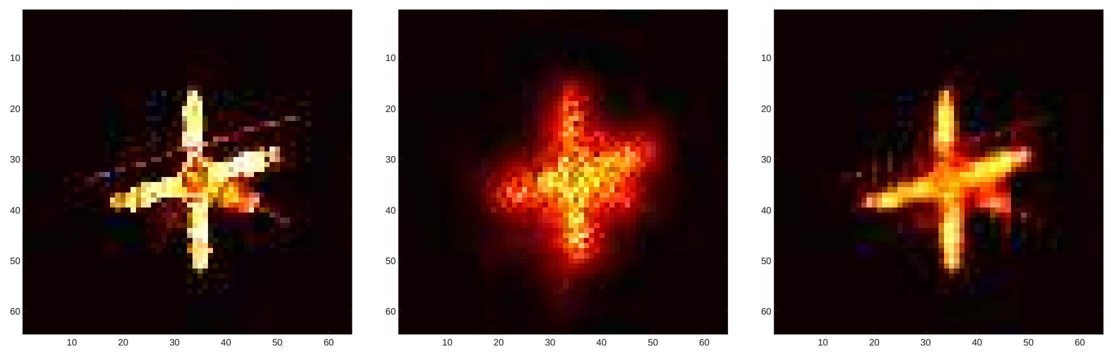

(a) Truth (b) FP Data (MSE=4.65) (c) Result (MSE=1.67)

Figure 6.29: Lowest MSE reconstruction from static aberration data. Generated with SNR=20, 30 frames, and $\sigma_d = 6e^-$. (a) Truth object. (b) Frame-averaged FP data with MSE=4.65. (c) 15-mode reconstruction with MSE=1.67

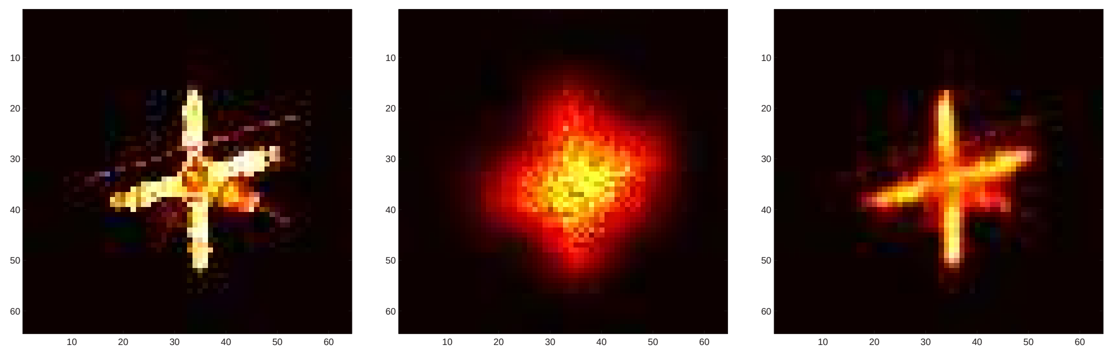
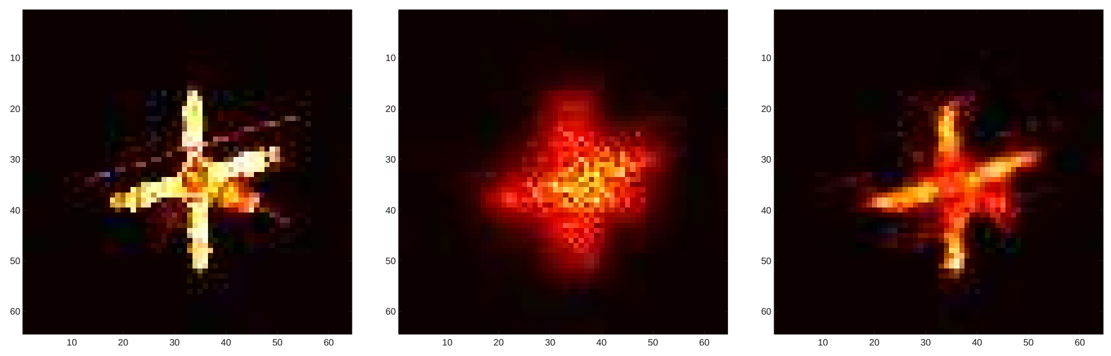
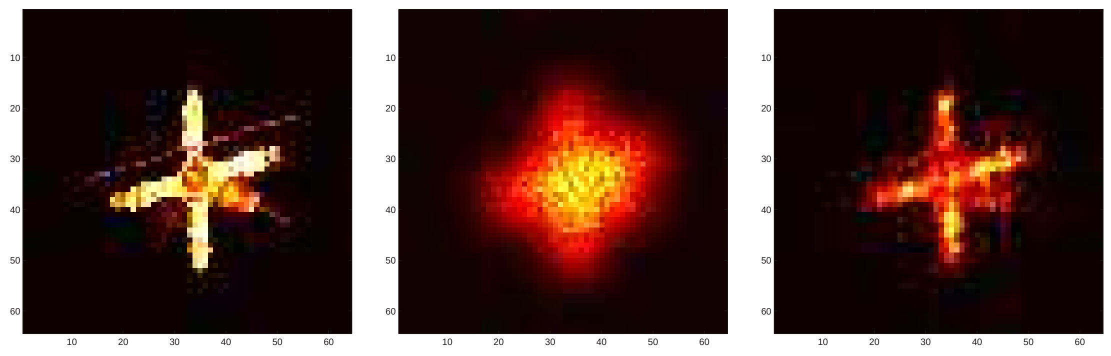

(a) Truth (b) FP Data (MSE=6.59) (c) Result (MSE=1.83)

Figure 6.30: Lowest MSE reconstruction from dynamic aberration data. Generated with SNR=7, 50 frames, and $\sigma_d = 6e^-$. (a) Truth object. (b) Frame-averaged FP data with MSE=6.59. (c) 15-mode reconstruction with MSE=1.83



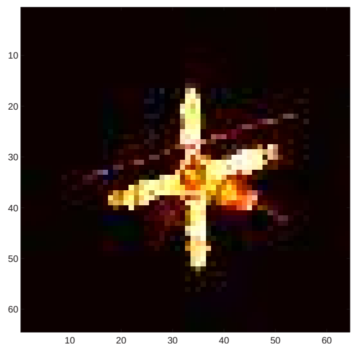
(a) Truth (b) FP Data (MSE=5.54) (c) Result (MSE=3.77)

Figure 6.31: Average MSE reconstruction from static aberration data. Generated with SNR=3, 20 frames, and $\sigma_d = 100e^{-}$. (a) Truth object. (b) Frame-averaged FP data with MSE=5.54. (c) 15-mode reconstruction with MSE=3.77

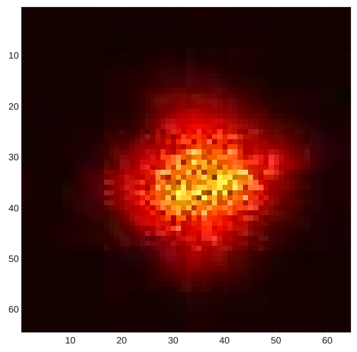


(a) Truth (b) FP Data (MSE=6.69) (c) Result (MSE=2.79)

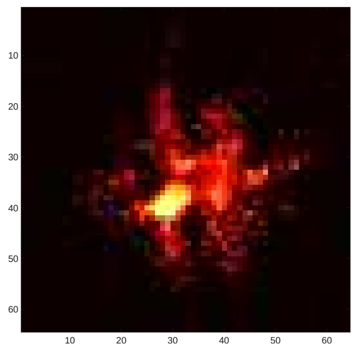
Figure 6.32: Average MSE reconstruction from dynamic aberration data. Generated with SNR=2, 50 frames, and $\sigma_d = 100e^{-}$. (a) Truth object. (b) Frame-averaged FP data with MSE=6.69. (c) 15-mode reconstruction with MSE=2.79



(a) Truth

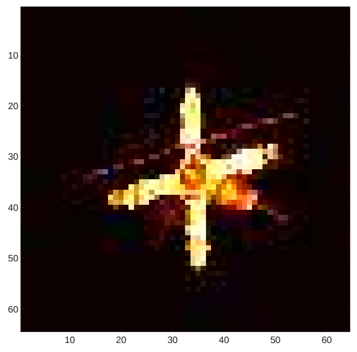


(b) FP Data (MSE=7.29)

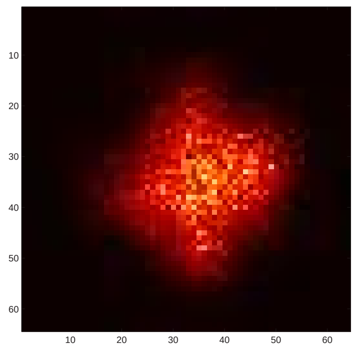


(c) Result (MSE=10.18)

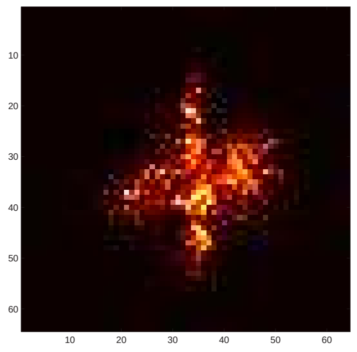
Figure 6.33: Worst MSE reconstruction from static aberration data. Generated with SNR=2, 10 frames, and $\sigma_d = 100e^-$. (a) Truth object. (b) Frame-averaged FP data with MSE=7.29. (c) 15-mode reconstruction with MSE=10.18



(a) Truth



(b) FP Data (MSE=6.90)



(c) Result (MSE=6.76)

Figure 6.34: Worst MSE reconstruction from dynamic aberration data. Generated with SNR=3, 10 frames, and $\sigma_d = 6e^-$. (a) Truth object. (b) Frame-averaged FP data with MSE=6.90. (c) 15-mode reconstruction with MSE=6.76

VII. Polarization Data in Active Imaging

In addition to conventional intensity measurements used for the APDI algorithm, the spatially resolved polarization state of the incident light can be measured, and might potentially provide valuable information for use in an image reconstruction algorithm. This chapter covers the development of three approaches to using PP polarization data in image reconstruction. First, a statistical description for PP polarization data was developed and an EM-algorithm developed to reconstruct the image by estimating the field correlation matrix and relating it to the original object intensity. While this is the most direct and mathematically rigorous approach to using PP polarization data, the derived algorithm proved to be far too complex for implementation.

As an alternative, PP polarization data was mathematically shaped to fit into the basic structure of the APDI algorithm, and minor modifications were made to generate a lensless APDI (LAPDI) variant. Given a good starting guess, the LAPDI algorithm converged to a good reconstruction in a few iterations. When a random starting guess was used however, the algorithm stalled at a local maximum after one iteration, resulting in a reconstruction that was useless. In addition to converging to false reconstructions, the LAPDI algorithm is extremely computationally expensive and would require significant changes in order to be practical.

Finally, a statistical description of the polarization phase was developed with the intent of replacing the PP imaging sensor with a four-channel imaging polarimeter. The relationship between the polarization phase PSD and the truth object was established, and an approximate numerical form developed. Though there is a significant amount of information about the truth object encoded in the PP polarization phase, the developed implementation resulted in worse reconstructions and had a higher computational burden.

7.1 Direct Image Estimation From Pupil Plane Polarization State Measurements

The first question considered is the feasibility of reconstructing an image directly from the spatially resolved PP polarization state. For the purposes of model development, rough surface scattering is assumed to be the dominant factor in depolarization, and statistical independence between polarizations is assumed. Similar to the statistical description of laser speckle, it's assumed that the size of a scattering surface on the object is sufficiently small that contributions from a large number of scatterers are observed at all points located any significant distance from the object. After reflection from the object, a linear, homogeneous, isotropic propagation media is assumed, and scalar diffraction theory is used to treat polarization components independently.

Treating polarizations independently, the polarized portion of the field in the PP is completely described by projecting it against the primary s and p polarization directions

$$U_s = A_s e^{j\theta_s} \tag{7.1a}$$

$$U_p = A_p e^{j(\theta_s+\theta_{12})} \tag{7.1b}$$

where θ_s is the unknown propagation phase for the s polarization, A_s and A_p are the PP amplitudes for the two polarizations, and θ_{12} is the known polarization phase. Assuming the spatially resolved Stokes vector is measured, the available data are given by

$$A_s = \sqrt{\frac{1}{2}(S_0' + S_1)} \tag{7.2a}$$

$$A_p = \sqrt{\frac{1}{2}(S_0' - S_1)} \tag{7.2b}$$

$$\theta_{12} = \arctan\frac{S_2}{S_3} \tag{7.2c}$$

where as before

$$S_0' = \sqrt{S_1^2 + S_2^2 + S_3^2} \tag{7.3a}$$

$$= \mathcal{P} S_0 \tag{7.3b}$$

and each of A_s, A_p, and θ_{12} are $N \times N$ data arrays.

To use statistical methods to estimate the lost phase information, an adequate statistical model for the data must be developed. For the linear-algebra approach used here, two-dimensional data arrays must first be re-ordered or "lexicographically stacked" [1] into an $N^2 \times 1$ column vector which is built up by "raster scanning" the image and dumping the elements sequentially into the output vector such that

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix} \Rightarrow \begin{bmatrix} a_{11} \\ a_{12} \\ \vdots \\ a_{1N} \\ a_{21} \\ \vdots \\ a_{NN} \end{bmatrix} \tag{7.4}$$

Once the data has been restructured, a statistical model can be formed using familiar linear algebra notation.

From Chapter III, the statistics in both the object and detector planes obey circular complex Gaussian statistics, described by

$$P(U) = \frac{1}{(2\pi)^{N^2/2}\sqrt{|\mathbf{\Gamma}|}} \exp\left\{-\frac{1}{2}U^H \mathbf{\Gamma}^{-1} U\right\} \tag{7.5}$$

where U is the field component along either polarization direction and $\mathbf{\Gamma}$ is the associated correlation matrix where the $(i,j)^{th}$ element is $\mathrm{E}\left[U^*(i)U(j)\right]$, which can also be interpreted as the mutual intensity $J_{12}(P_i, P_j)$ of the field components at the points represented by the i and j indexes in the PP. To specify the joint probability of the two polarization components, independence is assumed, and the probability density functions for each component are multiplied. This results in a joint probability distribution function given by

$$P(U_s, U_p) = \frac{1}{(2\pi)^{N^2}\sqrt{|\mathbf{\Gamma}_s|\,|\mathbf{\Gamma}_p|}} \exp\left\{-\frac{1}{2}\left(U_s^H \mathbf{\Gamma}_s^{-1} U_s + U_p^H \mathbf{\Gamma}_p^{-1} U_p\right)\right\} \tag{7.6}$$

Given that A_s, A_p, and θ_{12} are known to within the uncertainty of noise in the data, it makes sense to separate these quantities out of (7.6) and isolate the common phase θ_s. To do this, individual field elements $U_s(i) = A_s(i)e^{j\theta_s(i)}$ and $U_p(i) = A_p(i)e^{j\theta_s(i)}e^{j\theta_{12}(i)}$ can be written $U_s(i) = A_s(i)\vartheta(i)$ and $U_p(i) = A_p(i)e^{j\theta_{12}(i)}\vartheta(i)$ where $\vartheta(i) = e^{j\theta_s(i)}$. In the vector representation this can be written

$$U_s = \mathbf{A}_s \vartheta \tag{7.7a}$$

$$U_p = \mathbf{A}_p \vartheta \tag{7.7b}$$

where ϑ is an $N^2 \times 1$ vector defined by

$$\vartheta = e^{j\theta_s} \tag{7.8}$$

and

$$\mathbf{A}_s = \text{diag}\{A_s\} \tag{7.9a}$$

$$\mathbf{A}_p = \text{diag}\{A_p\}\, \text{diag}\left\{e^{j\theta_{12}}\right\} \tag{7.9b}$$

where diag $\{\cdot\}$ is a diagonal matrix composed of elements of the argument. The exponent of (7.6) can then be written

$$U_s^H \mathbf{\Gamma}_s^{-1} U_s + U_p^H \mathbf{\Gamma}_p^{-1} U_p = \vartheta^H \mathbf{M} \vartheta \tag{7.10}$$

where

$$\mathbf{M} = \mathbf{A}_s^H \mathbf{\Gamma}_s^{-1} \mathbf{A}_s + \mathbf{A}_p^H \mathbf{\Gamma}_p^{-1} \mathbf{A}_p \tag{7.11}$$

Using these simplifications, the probability distribution of ϑ, A_s, A_p, and θ_{12} given $\mathbf{\Gamma}_s$, and $\mathbf{\Gamma}_p$ is given by

$$P(\theta_s, A_s, A_p, \theta_{12}|\mathbf{\Gamma}_s, \mathbf{\Gamma}_p) = \frac{1}{(2\pi)^{N^2}\sqrt{|\mathbf{\Gamma}_s|\,|\mathbf{\Gamma}_p|}} \exp\left\{-\frac{1}{2}\vartheta^H \mathbf{M} \vartheta\right\} \tag{7.12}$$

7.1.1 Phase Estimation With a Known Correlation Matrix. After completing the statistical system model, the next step is to construct an estimation rule that will result in the best possible reconstruction of the object being imaged. One approach is to form an estimate of the phase and reconstruct the image through inverse-transforming the now complete PP field distribution. Alternatively, the correlation matrices $\mathbf{\Gamma}_s$ and $\mathbf{\Gamma}_p$ can be estimated and used to get an estimate of the object intensity distribution by noting that the $(i,j)^{th}$ element of $\mathbf{\Gamma}$ is the mutual intensity for the field at the points represented by the i and j indexes. Once an estimate for the mutual intensity has been formed, the object can be reconstructed using (3.31) via an inverse Fourier transform.

Assuming the full conditional probability is available, the best possible estimate for the lost phase information can be found by forming the ML estimate in accordance with section 2.1.3. The ML estimation rule for the complex phasor ϑ is given by

$$\hat{\vartheta} = \underset{\vartheta}{\operatorname{argmax}}\, P(\vartheta | A_s, A_p, \theta_{12}) \tag{7.13}$$

where $\hat{\vartheta}$ is the ML estimate of ϑ. To eliminate exponentials, the log-likelihood is maximized such that

$$\hat{\vartheta} = \underset{\vartheta}{\operatorname{argmin}} \left[\vartheta^H \mathbf{M} \vartheta\right] \tag{7.14}$$

A quick look at the form of (7.14) shows it to be an N^2-dimensional quadratic that is centered on zero, implying that the corresponding minimum would be located at the point $\hat{\vartheta} = 0$. However, the definition of ϑ from (7.8) places constraints on the possible solutions, and results in non-trivial estimates.

In a previous implementation of an estimator for ϑ [20], a constraint was imposed that required $\|\vartheta\|_2 = N$ or otherwise expressed

$$\vartheta^H \vartheta = N^2 \tag{7.15}$$

Applying this constraint,

$$\hat{\vartheta} = \underset{\vartheta^H\vartheta = N^2}{\operatorname{argmin}} \left[\vartheta^H \mathbf{M} \vartheta\right] \tag{7.16}$$

Using the constrained optimization method outlined in section 2.2.4, the optimum estimate for ϑ is found by solving the eigenvalue problem

$$\mathbf{M}\vartheta = \lambda_l \vartheta \tag{7.17}$$

and setting the estimate $\hat{\vartheta}$ equal to the eigenvector associated with the minimum eigenvalue.

While the constraint given by (7.15) gives the problem a convenient form and limits the set of possible solutions to a set including the possible valid values of ϑ, it also includes a much larger set of possible solution vectors that do not correspond to valid estimates. Given that ϑ is a unit-magnitude complex phasor, the correct constraint function for the optimization can be written

$$g_i(\vartheta) = 1 - \vartheta_i^* \vartheta_i \tag{7.18}$$

where $g_i(\vartheta)$ is the i^{th} element of the vector valued function $g(\vartheta)$, ϑ_i is the i^{th} element of ϑ, and * denotes complex conjugation. Using this constraint function, the Lagrangian for optimization is given by

$$L(\vartheta, \vec{\lambda}_l) = \vartheta^H \mathbf{M} \vartheta + \vec{\lambda}_l g(\vartheta) \tag{7.19}$$

which is nonlinear and requires special care. One approach to solving the correctly constrained problem was presented by Szeto [47], and is paraphrased here.

Given that piston (a constant phase across the reconstruction) is an unobservable quantity that does not directly impact the image reconstruction, it's constrained to be zero to eliminate any ambiguity in the reconstruction result. To implement this, the solution is constrained such that

$$\sum_{i=1}^{n} \mathcal{Im}\{\vartheta_i\} = 0 \tag{7.20}$$

To add this constraint to the Lagrangian, an additional multiplier μ_l is added, resulting in

$$L(\vartheta, \vec{\lambda}_l, \mu_l) = \vartheta^H \mathbf{M} \vartheta + \vec{\lambda}_l^T g(\vartheta) + \mu_l \mathbb{1}^T \mathcal{Im}\{\vartheta\} \tag{7.21}$$

where $\mathbb{1}$ is an N^2-vector of ones and μ_l is a scalar multiplier.

To eliminate the need for complex algebra, ϑ is written in terms of its real and imaginary parts such that

$$\vartheta = \mathbf{u} + j\mathbf{v} \tag{7.22}$$

and $\mathbf{M}$ is split into its real and imaginary parts such that

$$\mathbf{A} = \mathcal{Re}\{\mathbf{M}\} \tag{7.23a}$$

$$\mathbf{B} = \mathcal{Im}\{\mathbf{M}\} \tag{7.23b}$$

where $\mathbf{A}$ is symmetric and $\mathbf{B}$ is skew-symmetric. Substituting these definitions into (7.21) and simplifying results in the modified Lagrangian

$$L(\mathbf{u}, \mathbf{v}, \vec{\lambda}_l, \mu_l) = \mathbf{u}^T(\mathbf{A}\mathbf{u} - \mathbf{B}\mathbf{v}) + \mathbf{v}^T(\mathbf{A}\mathbf{v} + \mathbf{B}\mathbf{u}) + \vec{\lambda}_l^T g(\mathbf{u}, \mathbf{v}) + \mu_l \mathbb{1}^T \mathbf{v} \tag{7.24}$$

At this point it's convenient to define

$$\mathbf{\Lambda} = \text{diag}\left\{\vec{\lambda}_l\right\} \tag{7.25}$$

as a diagonal matrix built from the vector of Lagrange multipliers, and write $g(\mathbf{u}, \mathbf{v})$ in a form more convenient for linear algebra operations

$$g(\mathbf{u}, \mathbf{v}) = \mathbb{1} - (\mathbf{U}\mathbf{u} + \mathbf{V}\mathbf{v}) \tag{7.26}$$

where $\mathbf{U}$ and $\mathbf{V}$ are diagonal matrices such that

$$\mathbf{U} = \text{diag}\left\{\mathbf{u}\right\} \tag{7.27a}$$

$$\mathbf{V} = \text{diag}\left\{\mathbf{v}\right\} \tag{7.27b}$$

Differentiating (7.24) with respect to $\mathbf{u}$, $\mathbf{v}$, $\vec{\lambda}_l$, and μ_l and setting the results equal to zero produces the system of equations

$$\mathbf{A}\mathbf{u} - \mathbf{B}\mathbf{v} = -\mathbf{\Lambda}\mathbf{u} \tag{7.28a}$$

$$\mathbf{A}\mathbf{v} + \mathbf{B}\mathbf{u} = -\mathbf{\Lambda}\mathbf{v} - \mu_l \mathbb{1} \tag{7.28b}$$

$$\mathbf{U}\mathbf{u} + \mathbf{V}\mathbf{v} = \mathbb{1} \tag{7.28c}$$

$$\mathbb{1}^T \mathbf{v} = 0 \tag{7.28d}$$

the solution of which will yield the optimum estimate for ϑ which can then be used in conjunction with the remaining data to form an image. Szeto showed that for his application, this system of equations converged quadratically using Newton's method.

7.1.2 Correlation Matrix Estimation. One critical assumption in the previous approach is that the correlation matrix $\mathbf{\Gamma}$ is known. Alternatively, assuming that the propagation phase is known, the correlation matrix can be estimated using the complex field. Considering (3.31) and ignoring scaling and phase terms, $\mathbf{\Gamma}$ is only a function of the separation $(\Delta u, \Delta v)$ between the two points being correlated. Assuming that both the amplitude and phase information are available, the correlation matrix can be estimated using the fact that there are multiple data pairs within a data set with the same separation. An efficient method for forming this estimate is to use FFT's to generate the autocorrelation of the $N \times N$ complex field arrays assuming circular shifts, then using the autocorrelation to populate the $\mathbf{\Gamma}$ matrix based on point separations.

Unfortunately, the phase information is not available. What is available, however, is the polarization phase. To consider the relationship between the propagation phase and the polarization phase, the phases at two points P_1 and P_2 are written

$$\theta_s^1 = \text{S-polarization phase @ } P_1 \tag{7.29a}$$

$$\theta_p^1 = \text{P-polarization phase @ } P_1 \tag{7.29b}$$

$$\theta_s^2 = \text{S-polarization phase @ } P_2 \tag{7.29c}$$

$$\theta_p^2 = \text{P-polarization phase @ } P_2 \tag{7.29d}$$

The polarization phases are then given by

$$\theta_{12}^1 = \theta_s^1 - \theta_p^1 \tag{7.30a}$$

$$\theta_{12}^2 = \theta_s^2 - \theta_p^2 \tag{7.30b}$$

Evaluating the polarization phase correlation in terms of the propagation phases

$$\mathrm{E}\left[\theta_{12}^1\theta_{12}^2\right] = \mathrm{E}\left[(\theta_s^1 - \theta_p^1)(\theta_s^2 - \theta_p^2)\right] \tag{7.31a}$$

$$= \mathrm{E}\left[\theta_s^1\theta_s^2 - \theta_s^1\theta_p^2 - \theta_s^2\theta_p^1 + \theta_p^1\theta_p^2\right] \tag{7.31b}$$

$$= \mathrm{E}\left[\theta_s^1\theta_s^2\right] - \mathrm{E}\left[\theta_s^1\theta_p^2\right] - \mathrm{E}\left[\theta_s^2\theta_p^1\right] + \mathrm{E}\left[\theta_p^1\theta_p^2\right] \tag{7.31c}$$

Assuming statistical independence of orthogonal polarizations, $\mathrm{E}\left[\theta_s^1\theta_p^2\right] = 0$ and $\mathrm{E}\left[\theta_s^2\theta_p^1\right] = 0$ resulting in

$$\mathrm{E}\left[\theta_{12}^1\theta_{12}^2\right] = \mathrm{E}\left[\theta_s^1\theta_s^2\right] + \mathrm{E}\left[\theta_p^1\theta_p^2\right] \tag{7.32}$$

which when $\mathrm{E}\left[\theta_s^1\theta_s^2\right] = \mathrm{E}\left[\theta_p^1\theta_p^2\right]$ is twice the average of the phase correlation for each polarization. If the two polarizations have similar reflectivities the correlations will be very similar, and the polarization phase correlation will closely match the true phase correlation.

Given the relationship between polarization phase correlation and propagation phase correlation, and that the polarization phase correlation is easily estimated from measured data, the next logical step is to relate the phase correlation to the field correlation. Given the two-point joint phase probability distribution function of (3.45) the correlation between the phases θ_1 and θ_2 at points P_1 and P_2 is given by

$$\mathrm{E}\left[\theta_1\theta_2\right] = \iint_{-\infty}^{\infty} \theta_1\theta_2 \frac{1-\mu^2}{4\pi^2(1-\beta^2)^{3/2}} \left(\beta\sin^{-1}\beta + \frac{\pi\beta}{2} + \sqrt{1-\beta^2}\right) \mathrm{d}\theta_1\mathrm{d}\theta_2 \tag{7.33}$$

where μ_c is the complex correlation coefficient, $\mu = |\mu_c|$, $\beta = \mu\cos(\theta_2 - \theta_1 + \psi)$, and $\psi = \angle\mu_c$. Unfortunately, (7.33) does not lead to a closed-form relationship between $\mathrm{E}\left[\theta_1\theta_2\right]$ and μ_c. Numerical evaluation of (7.33) over the range of μ and ψ results in Fig. 7.1, which shows the predicted strong dependence. However, it can be seen that the correlation is not invertible, having multiple possible values of μ_c for a given value of $\mathrm{E}\left[\theta_1\theta_2\right]$. As a consequence, direct estimation of μ_c or equivalently $\mathbf{\Gamma}$ from polarization data using this approach is not possible.

7.1.3 Iterative Refinement. Given that there are two unknowns in the statistical model, a common approach to generating an estimate is to optimize the first unknown given a current estimate of the second, then update the second based on the results of the first. In a previous implementation developed at the Air Force Research Laboratory (AFRL) [20] this method was used to iteratively refine estimates for the common phase term and the correlation matrix. Given an initial estimate for the correlation matrix, an estimate for the phasor ϑ is formed then used with the remaining data to develop a better estimate for the correlation matrix. This process is then iterated in an attempt to converge to the optimal solution.

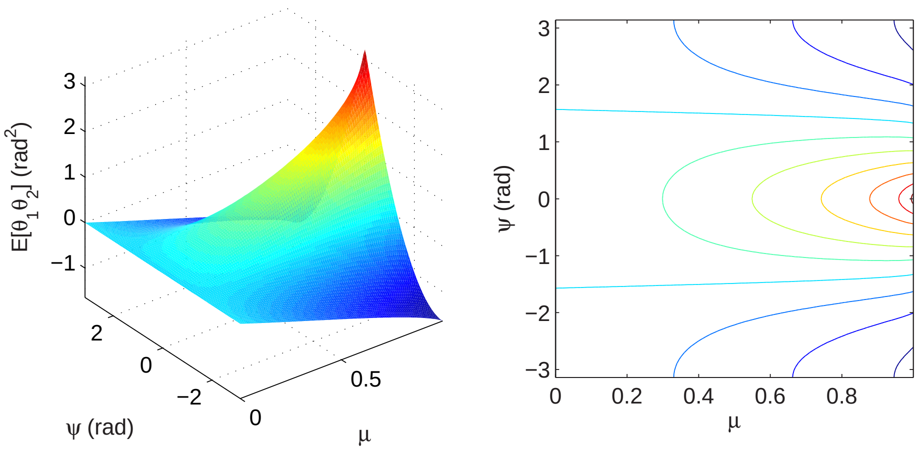


Figure 7.1: Functional dependence of $\mathrm{E}[\theta_1\theta_2]$ on the magnitude and phase of the field correlation μ_c.

7.1.4 EM-Algorithm Correlation Matrix Estimation. A more mathematically rigorous approach than alternating estimation of the two unknowns is to develop an EM-algorithm in accordance with section 2.1.4. The EM algorithm is explicitly designed to deal with estimation of parameters when there is some form of unobservable data; and has been shown to converge to a local maximum in general, and to the true maximum likelihood estimate if the likelihood function is convex over the space of possible solutions [15]. In the problem of interest, the correlation matrices $\mathbf{\Gamma}_s$ and $\mathbf{\Gamma}_p$ are the parameters to be estimated, the common phase/phasor ϑ is the unobservable or hidden data, and A_s, A_p, and θ_{12} are the incomplete or measured data. In contrast to the AFRL implementation, ϑ is not estimated. Rather, estimates of the parameters $\mathbf{\Gamma}_s$ and $\mathbf{\Gamma}_p$ are developed using the measured data and the statistical description of the system.

In this case, the complete data are described by the joint-density function of (7.12), or

$$P(\theta_s, A_s, A_p, \theta_{12}|\mathbf{\Gamma}_s, \mathbf{\Gamma}_p) = \frac{1}{(2\pi)^{N^2}\sqrt{|\mathbf{\Gamma}_s|\,|\mathbf{\Gamma}_p|}} \exp\left\{-\frac{1}{2}\vartheta^H \mathbf{M}\vartheta\right\} \tag{7.34}$$

which leads to a log-likelihood function given by

$$\mathcal{L}(\mathbf{\Gamma}_s, \mathbf{\Gamma}_p|\vartheta, \mathbf{A}_s, \mathbf{A}_p, \theta_{12}) = -\vartheta^H \mathbf{M}\vartheta \tag{7.35}$$

where as in (7.11)

$$\mathbf{M} = \mathbf{A}_s^H \mathbf{\Gamma}_s^{-1} \mathbf{A}_s + \mathbf{A}_p^H \mathbf{\Gamma}_p^{-1} \mathbf{A}_p \tag{7.36}$$

and where $\mathbf{A}_s$ and $\mathbf{A}_p$ are as defined in (7.9). The conditional expectation over which the maximization is performed is then given by

$$\begin{aligned} Q(\mathbf{M}, \mathbf{M}^{(i)}) =& \,\mathrm{E}\left[\mathcal{L}(\mathbf{\Gamma}_s, \mathbf{\Gamma}_p|\mathbf{M})|\mathbf{M}^{(i)}\right] & \text{(7.37a)} \\ =& -\mathrm{E}\left[\vartheta^H \mathbf{M} \vartheta|\mathbf{M}^{(i)}\right] & \text{(7.37b)} \\ =& -\int_{\vartheta \in \mathbf{\Theta}} \vartheta^H \mathbf{M} \vartheta p(\vartheta|\mathbf{M}^{(i)}) \mathrm{d}\vartheta & \text{(7.37c)} \end{aligned}$$

where $\mathbf{\Theta}$ is the space of all unit-magnitude complex phasors.

To complete the expectation, the conditional probability distribution $p(\vartheta|\mathbf{M}^{(i)})$ must be known. Using Bayes' rule, this conditional distribution is given by

$$p(\theta_s|\mathbf{M}^{(i)}) = \frac{p(\theta_s, A_s, A_p, \theta_{12}|\mathbf{\Gamma}_s, \mathbf{\Gamma}_p)}{p(A_s, A_p, \theta_{12}|\mathbf{\Gamma}_s, \mathbf{\Gamma}_p)} \tag{7.38}$$

however, the marginal distribution $p(A_s, A_p, \theta_{12}|\mathbf{\Gamma}_s, \mathbf{\Gamma}_p)$ for this problem is not easily obtainable. As was previously mentioned, because the marginal distribution used for the expectation is not a function of the updated parameter estimate, it's effectively constant and the expectation can be performed using $p(\vartheta, A_s, A_p, \theta_{12}|\mathbf{\Gamma}_s, \mathbf{\Gamma}_p)$. Applying this and dropping the constant multiplier (with the exception of the negative sign), the expectation becomes

$$Q(\mathbf{M}, \mathbf{M}^{(i)}) = -\int_{\vartheta \in \mathbf{\Theta}} \vartheta^H \mathbf{M} \vartheta \exp\left\{-\frac{1}{2}\vartheta^H \mathbf{M}^{(i)} \vartheta\right\} \mathrm{d}\vartheta \tag{7.39}$$

Continuing to the maximization step of the EM algorithm, the updated estimate for $\mathbf{M}$ is then given by

$$\mathbf{M}^{(i+1)} = \underset{\mathbf{M}}{\operatorname{argmax}}\, Q(\mathbf{M}, \mathbf{M}^{(i)}) \tag{7.40}$$

7.1.5 Sparse Solution Approach. In order to implement this EM-algorithm, the $N \times N$ data set must be re-ordered into an $N^2 \times 1$ vector, and then the $N^2 \times N^2$ correlation matrix must be formed. For large images this can result in a problem that is too large to solve directly. For example, a 256×256 image would lead to a 65536×65536 correlation matrix requiring $32Gb$ of memory to store in 64-bit double-precision floating point format. Given that the addressable memory on a 32-bit processor is capped at $4Gb$, the required space for simply storing the correlation matrix puts the problem out of the reach of many currently available computers. Additionally, the computation time involved in directly operating on such a large matrix would be significant.

Based on the definition of $\mathbf{\Gamma}$, the correlation matrix takes on a Toeplitz-block-Toeplitz form. Furthermore, because the magnitude of the correlation depends only on the absolute separation between the points, the magnitude of each block will be identical with only a known phase difference between them. This introduces a large amount of redundancy that can substantially reduce the storage requirements for $\mathbf{\Gamma}$. If the solution algorithm can be customized to take advantage of the structure and redundancy in the matrix, the storage and computational requirements can be reduced significantly.

One approach to utilizing this redundancy is to pose the problem as sets of two point data pairs characterized by a common point separation. This can be done without loss in generality because of the Gaussian statistics which are completely characterized by their first and second moments. An estimator can be built for each point separation, and used to incrementally build up an estimate for the full correlation matrix. To do this, the statistical model is modified to only include the joint two-point distribution of intensity and phase given by (3.43) re-written here as

$$p(I_1, I_2, \theta_1, \theta_2) = \frac{1}{4\bar{I}^2\pi^2(1-\mu^2)} \exp\left[-\frac{I_1 + I_2 - 2\sqrt{I_1 I_2}\mu\cos(\theta_1 - \theta_2 + \psi)}{\bar{I}(1-\mu^2)}\right] \tag{7.41}$$

where $\mu = |\mu_c|$, $\psi = \angle\mu_c$, and $\bar{I}$ is the average intensity in the detector plane.

Assuming independence of orthogonal polarizations and defining the fields along the s- and p-polarizations as

$$U_s = A_s e^{i\theta_s} \tag{7.42a}$$

$$U_p = A_p e^{i\theta_s + \theta_{12}} \tag{7.42b}$$

the joint pdf for any set of data pairs from each polarization is given by

$$p(I_s^1, I_s^2, I_p^1, I_p^2, \theta_s^1, \theta_s^2, \theta_{12}^1\theta_{12}^2) = \frac{1}{(4\pi^2\bar{I}_s\bar{I}_p)^2(1-(\mu_s)^2)(1-(\mu_p)^2)}\times$$
$$\exp\left[-\frac{I_s^1 + I_s^2 - 2\sqrt{I_s^1 I_s^2}\mu_s\cos\left(\theta_s^1 - \theta_s^2 + \psi^s\right)}{\bar{I}_s(1-(\mu_s)^2)}\right.$$
$$\left.-\frac{I_p^1 + I_p^2 - 2\sqrt{I_p^1 I_p^2}\mu_p\cos\left(\theta_s^1 - \theta_{12}^1 - \theta_s^2 + \theta_{12}^1 + \psi^p\right)}{\bar{I}_p(1-(\mu_p)^2)}\right] \tag{7.43}$$

Point pairs with the same separation are treated independently, and the joint distribution for N point pairs with set separations is given by

$$p(\vec{I}_s^1, \vec{I}_s^2, \vec{I}_p^1\vec{I}_p^2, \vec{\theta}_s^1, \vec{\theta}_s^2, \vec{\theta}_{12}^1, \vec{\theta}_{12}^2) = \prod_n p(I_s^1(n), I_s^2(n), I_p^1(n), I_p^2(n), \theta_s^1(n), \theta_s^2(n), \theta_{12}^1(n)\theta_{12}^2(n)) \tag{7.44}$$

This distribution can then be used to develop a more tractable EM-algorithm for estimation of μ_c. To simplify development of an EM-algorithm for this distribution, only one point pair is used at present. The result will be expanded later to include multiple pairs.

Using the joint PDF of (7.43) and throwing out terms that are not functions of the correlation coefficients, the CD log-likelihood function is given by

$$\mathcal{L}(\mu_c^s, \mu_c^p) = -\ln\left\{(1-(\mu_s)^2)(1-(\mu_p)^2)\right\} - \frac{I_s^1 + I_s^2 - 2\sqrt{I_s^1 I_s^2}\mu_s\cos\left(\theta_s^1 - \theta_s^2 + \psi^s\right)}{\bar{I}_s(1-(\mu_s)^2)}$$
$$-\frac{I_p^1 + I_p^2 - 2\sqrt{I_p^1 I_p^2}\mu_p\cos\left(\theta_s^1 + \theta_{12}^1 - \theta_s^2 - \theta_{12}^2 + \psi^p\right)}{\bar{I}_p(1-(\mu_p)^2)} \tag{7.45}$$

To carry out the expectation of (2.16), the conditional PDF for θ_s^1, and θ_s^2, given I_s^1, I_s^2, I_p^1, I_p^2, θ_{12}^1, and θ_{12}^2 must be found using Bayes' rule such that

$$p(\theta_s^1, \theta_s^2 | I_s^1, I_s^2, I_p^1, I_p^2, \theta_{12}^1, \theta_{12}^2) = \frac{p(I_s^1, I_s^2, I_p^1, I_p^2, \theta_s^1, \theta_s^2, \theta_{12}^1\theta_{12}^2)}{p(I_s^1, I_s^2, I_p^1, I_p^2, \theta_{12}^1\theta_{12}^2)} \tag{7.46}$$

which requires the joint marginal distribution of the known parameters I_s^1, I_s^2, I_p^1, I_p^2, θ_{12}^1, and θ_{12}^2.

Because the fields are independent, the marginal distributions for each polarization will also be independent and can be found individually and multiplied to find the desired quantity rather than attempting to integrate (7.43) directly. Doing this and temporarily ignoring the constants in front of the exponential in (7.41), the desired marginal distributions are given by

$$p(I_s^1, I_s^2) = \iint_{-\pi}^{\pi} \frac{1}{4\bar{I}^2\pi^2(1-(\mu_s)^2)} \exp\left[-\frac{I_1^s + I_2^s - 2\sqrt{I_1^s I_2^s}\mu_s \cos\left(\theta_s^1 - \theta_s^2 + \psi^s\right)}{\bar{I}_s(1-(\mu_s)^2)}\right] \mathrm{d}\theta_s^1 \mathrm{d}\theta_s^2 \tag{7.47}$$

and

$$p(I_p^1, I_p^2, \theta_{12}^1, \theta_{12}^2) = \iint_{-\pi}^{\pi} \frac{1}{4\bar{I}^2\pi^2(1-(\mu_p)^2)} \times \exp\left[-\frac{I_1^p + I_2^p - 2\sqrt{I_1^p I_2^p}\mu_p \cos\left(\theta_s^1 - \theta_s^2 + \theta_{12}^1 - \theta_{12}^2 + \psi^p\right)}{\bar{I}_p(1-(\mu_p)^2)}\right] \mathrm{d}\theta_s^1 \mathrm{d}\theta_s^2 \tag{7.48}$$

Using the results of Equation (2.94) in [12], these integrals are the same, and ultimately have no dependence on θ_{12}^1 and θ_{12}^2, with the final result for the joint-marginal PDF given by

$$p(I_s^1, I_s^2, I_p^1, I_p^2, \theta_{12}^1\theta_{12}^2) = \frac{1}{(\bar{I}_s)^2(\bar{I}_p)^2(1-(\mu_s)^2)(1-(\mu_p)^2)} \times \exp\left[-\frac{I_s^1 + I_s^2 + I_1^P + I_p^2}{\bar{I}_s\bar{I}_p(1-(\mu_s)^2)(1-(\mu_p)^2)}\right] I_o\left(\frac{2\sqrt{I_s^1 I_s^2}\mu_s}{\bar{I}_s(1-(\mu_s)^2)}\right) I_o\left(\frac{2\sqrt{I_p^1 I_p^2}\mu_p}{\bar{I}_p(1-(\mu_p)^2)}\right) \tag{7.49}$$

where $I_0(\cdot)$ is a modified Bessel function of the first kind, zero order.

Because the conditional distribution is conditioned on the current estimate for the parameter μ_c in the expectation step, this marginal distribution is a constant, and can be ignored. The expectation can be taken using the complete joint-probability given by (7.43) such that

$$Q(\mu_c, \mu_c^{(old)}) = \iint_{-\pi}^{\pi} \mathcal{L}(\mu_c) p(\theta_s^1, \theta_s^2, I_s^1, I_s^2, I_p^1, I_p^2, \theta_{12}^1, \theta_{12}^2 | \mu^{(old)}) \mathrm{d}\theta_s^1 \mathrm{d}\theta_s^2 \tag{7.50}$$

Applying (7.43) and (2.15) to (7.50), assuming for simplicity that the correlation coefficients for each polarization are the same, and pulling all factors that are independent of the phase terms θ_s^1 and θ_s^2 out of the integral, the expectation is then given by

$$Q(\mu_c, \mu_c^{(old)}) = f_1(\mu) + f_2(\mu, \mu^{(old)}) f_3(\psi) \tag{7.51}$$

where the term of current interest is

$$\begin{aligned} f_3(\psi) = \iint_{-\pi}^{\pi} & \left[\sqrt{I_s^1 I_s^2} \cos(\theta_s^1 - \theta_s^2 + \psi) + \sqrt{I_p^1 I_p^2} \cos(\theta_s^1 - \theta_s^2 + \theta_{12}^1 - \theta_{12}^2 + \psi) \right] \times \\ & \exp\left[\sqrt{I_s^1 I_s^2} \cos(\theta_s^1 - \theta_s^2 + \psi^{(old)}) + \right. \\ & \left. \sqrt{I_p^1 I_p^2} \cos(\theta_s^1 - \theta_s^2 + \theta_{12}^1 - \theta_{12}^2 + \psi^{(old)}) \right] \mathrm{d}\theta_s^1 \mathrm{d}\theta_s^2 \end{aligned} \tag{7.52}$$

Because f_1 and f_2 are everywhere nonnegative, (7.51) can be maximized over the space of possible correlation coefficients by maximizing f_3 with respect to ψ first, then using this result in (7.51) and maximizing over μ. To expand the problem to include multiple data pairs, independence of point pairs is assumed, and the integral of f_3 becomes

$$\int_{\vec{\theta}_s^1(1)} \cdots \int_{\vec{\theta}_s^1(N)} \int_{\vec{\theta}_s^2(1)} \cdots \int_{\vec{\theta}_s^2(N)} f_3'(\vec{\theta}_s^1, \vec{\theta}_s^2, \psi) \exp\left[f_3'(\vec{\theta}_s^1, \vec{\theta}_s^2, \psi^{(old)}) \right] \mathrm{d}\vec{\theta}_s^1 \mathrm{d}\vec{\theta}_s^2 \tag{7.53}$$

where there are twice as many integrals as there are point pairs with the specified separation,

$$\begin{aligned} f_3'(\vec{\theta}_s^1, \vec{\theta}_s^2, \psi) = \sum_{n=1}^{N} & \left[\sqrt{\vec{I}_s^1(n) \vec{I}_s^2(n)} \cos\left(\vec{\theta}_s^1(n) - \vec{\theta}_s^2(n) + \psi \right) + \right. \\ & \left. \sqrt{\vec{I}_p^1(n) \vec{I}_p^2(n)} \cos\left(\vec{\theta}_s^1(n) - \vec{\theta}_s^2(n) + \vec{\theta}_{12}^1(n) - \vec{\theta}_{12}^2(n) + \psi \right) \right] \end{aligned} \tag{7.54}$$

$\vec{\theta}_s^1$, $\vec{\theta}_s^2$, $\vec{I}_s^1$, $\vec{I}_s^2$, $\vec{I}_p^1$, $\vec{I}_p^2$, $\vec{\theta}_{12}^1$, and $\vec{\theta}_{12}^2$ are vectors containing the polarization phase for all point pairs with a given separation, and the integration is over all the unknown phase variables.

Because this integral has no known solution, numerical integration was considered. However, assuming for simplicity that the nested integrals could be separated, a 20 frame 64×64 pixel image maximization of (7.51) at all point pair separations would require approximately 1.3×10^9 numeric integrations per iteration. This also assumes that the result of (7.51) has a well defined maximum on the interval $(-\pi < \psi < \pi]$ which may or may not be the case. Because of the extreme cost of forming this estimate, further work on an EM algorithm for PP polarization data was halted at this point.

7.2 lensless APDI

Another approach to reconstructing images from PP polarization state data is to mathematically shape the problem to look like a phase-diversity (PD) problem and generate a lensless APDI (LAPDI) algorithm. Recall that the fields in the system PP immediately after the effective lens for a PD system were described by

$$U_{FP} = U_o \exp[j\phi] \tag{7.55a}$$

$$U_{DP} = U_o \exp[j(\phi + \phi_D)] \tag{7.55b}$$

where $U_o = A \exp[j\theta_1]$, θ_1 is the vacuum propagation phase, ϕ is the unknown aberration phase and ϕ_D is the known diversity phase. If all data is taken in the PP, the system is insensitive to the aberration phase, and the aberration phase can be arbitrarily set such that $\phi = -\theta_s$. In this case, the PP field could be written

$$U_{FP} = A \tag{7.56a}$$

$$U_{DP} = A \exp[j\phi_D] \tag{7.56b}$$

The FP and DP intensities would then be given by

$$d_{FP} \propto |\mathcal{F}\{A\}|^2 \tag{7.57a}$$

$$d_{DP} \propto |\mathcal{F}\{A \exp[j(\phi_D)]\}|^2 \tag{7.57b}$$

Allowing the amplitudes of (7.56) to have channel-dependent noise, (7.56) becomes

$$U_{FP} = A_s \tag{7.58a}$$

$$U_{DP} = A_p \exp[j\phi_D] \tag{7.58b}$$

Comparing (7.58) and (7.2), the data available from PP polarization state field can completely describe the field if we let $\phi_D = \theta_{12}$. The field description for each polarization is simply the square-root of the intensity. The operations in (7.57) required to generate an aberrated image can then be mathematically performed, and the APDI algorithm can be modified to work with only polarization data. The updated noiseless model data is given by

$$g_m^f = \mathbf{o} \otimes s_m^f \tag{7.59}$$

where $m \in \{FP, DP\}$, and

$$s_{FP}^f = \left|\mathcal{F}\{\exp\left[-j\phi\right]\}\right|^2 \tag{7.60a}$$

$$s_{DP}^f = \left|\mathcal{F}\{\exp\left[-j\left(\phi + \theta_{12}\right)\right]\}\right| \tag{7.60b}$$

The "measured data" for the FP and DP channels is defined as

$$d_{FP}^f\left(x, y\right) = \gamma_1 \left|\mathcal{F}\{A_s^f\}\right|^2 \tag{7.61a}$$

$$d_{DP}^f\left(x, y\right) = \gamma_2 \left|\mathcal{F}\{A_p^f\}\right|^2 \tag{7.61b}$$

where γ_1 and γ_2 are normalization constants, A_s^f and A_p^f are the measured field amplitudes for the principle polarizations in the f^{th} frame, and θ_{12}^f is the polarization phase for the f^{th} frame. To account for the non-zero average of A_s and A_p, it is subtracted out before the model images are formed. Another notable change is that the "diversity phase" is now a part of the data set, and changes with each frame. The modulus squared of A_s^f and A_p^f can be passed into the APDI PP-regularization term without additional changes.

The only major change to algorithm development is modification of the aberration parameterization. Because the aberrations being considered for the APDI algorithm were somewhat smooth and had a power spectrum that was predictable, a reduced representation for them could be easily found and the aberration represented with only a few parameters. However, when the unknown phase is the vacuum propagation phase, the complex structure of the phase makes projecting it against Zernike modes counterproductive. A suitable representation would require roughly as many parameters as there are pixels, and numerical roundoff and aliasing of the high-order modes would adversely impact the representation.

As an alternative, the natural basis set (shifted δ-functions) is used, making each phase pixel in each frame a variable of optimization. Other than expanding the variable space, the only change required is in the computation of the objective function gradient with respect to phase parameters.

Restating (5.18) and (5.19) for the partial derivative of g_m with respect to phase parameter $\phi(u_0, v_0)$ and substituting $\psi_n = \delta(u - u_0, v - v_0)$,

$$\frac{\partial g_m(x,y)}{\partial \phi(u_0, v_0)} = \mathbf{o}(x,y) \otimes \hat{s}_m^k(x,y) \tag{7.62}$$

where

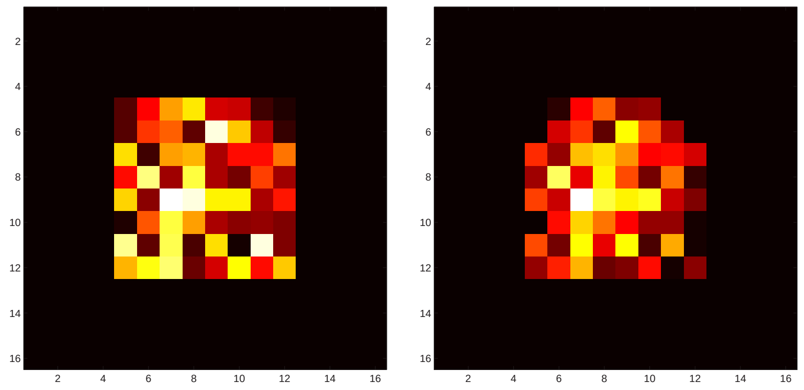
$$\begin{aligned}\hat{s}_m^k(x,y) = 2\mathcal{Re}\Big\{ & h_m^*(x,y)\mathcal{F}\Big\{ j\delta(u-u_0, v-v_0)P(u,v)\times \\ & \exp\Big[j\Big(\sum_n \left(\beta_m \phi_D(u,v) + \phi(u,v)\right)\delta(u-u_0, v-v_0)\Big)\Big]\Big\}\end{aligned} \tag{7.63}$$

$\beta_m = 1$ when $m = DP$ and 0 otherwise, and the aberration phase has not been parameterized. Assuming $P(u,v) = 1$ for all points in the data array, using the definition for the discrete Fourier transform, and employing the sifting property of the δ-function, (7.63) becomes

$$\hat{s}_m^k(x,y) = \frac{2}{N^2} \mathcal{Im}\left\{ \left(h_m^*(x,y)\right)^* \exp\left\{ -j \left[\frac{2\pi}{N}\left(u_0 x + v_0 y\right) + \beta_D \phi_D(u_0, v_0) + \phi(u_0, v_0) \right] \right\} \right\} \tag{7.64}$$

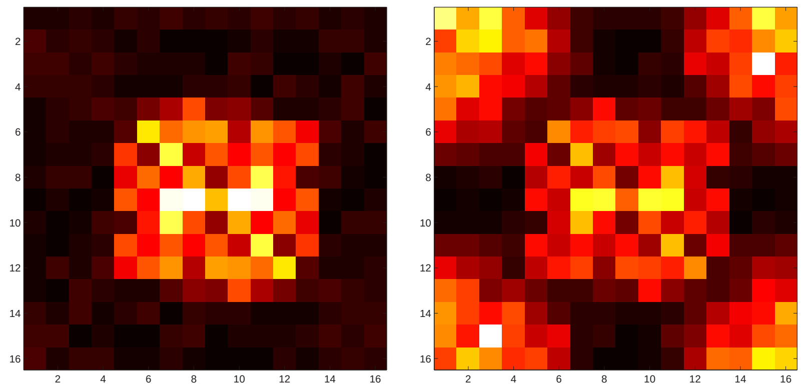
One major consequence of shifting from a truncated Zernike mode representation to a pixel-value representation for phase is that the computational costs skyrocket. For the original APDI implementation, each iteration requires $(6M + 12)\,F + 5$ FFTs, where there are F data frames, and M Zernike modes are compensated. This translates to 3065 FFTs per iteration for a 30-frame data set of any size with 15 modes per frame. By contrast, the LAPDI algorithm requires $(4N^2 + 12)F + 5$ FFTs for each iteration using an $N \times N$ data set, translating to roughly $5{\times}10^5$ FFTs per iteration for a 30-frame 64×64 pixel data set, and $2{\times}10^6$ for a 128×128 pixel 30-frame data set.

Because of the extreme computational cost, a 16×16 test image, shown in Fig. 7.2(a), was used for initial algorithm performance analysis. The reconstruction shown in Fig. 7.2(b) resulted from using the truth for the starting guess, and converged after six iterations. Given the inherently noisy data, the reconstruction is quite good. However, when starting with a random guess, the algorithm stalled at a local maximum after one iteration. The resulting reconstruction bore no resemblance to the original object. To get a better starting guess, the polarization phase PSD was used for the truth object and uniform random numbers were generated for the phase guess. Again, the optimization stalled after one iteration. The starting guess and reconstruction are shown in Fig. 7.3(a) and Fig. 7.3(b) respectively.



(a) Truth (b) Reconstruction from Truth

Figure 7.2: LAPDI (a) Truth object/initial guess, and (b) reconstruction. The algorithm converged after 6 iterations.



(a) Guess (b) Reconstruction

Figure 7.3: LAPDI (a) Initial guess from θ_{12} PSD, and (b) reconstruction. The algorithm stalled after 1 iteration.

The failure to converge is likely due to the fact that there are only $3FN^2$ data points for $(F+1)N^2$ unknowns, a ratio of slightly less than three to one for any realistic N and F. While this should be sufficient in a nearly noiseless system, speckle noise caused by laser statistics creates local minima that trap the optimization routine. Because each additional frame introduces N^2 additional unknowns while only adding $3N^2$ data points, the problem won't be significantly better conditioned by simply adding frames. An alternative approach must be taken to regularize the objective if this approach is to work. Because of its prohibitively expensive nature and failure to converge to satisfactory solutions, further work on the LAPDI algorithm was terminated.

7.3 Polarization Enhanced ADPI

An implication of the relationship between the polarization phase correlation, phase correlation, (3.35) and Fig. 7.1, is that there is a large amount of information about the object encoded in the polarization phase measurements. An alternative approach to incorporating this information into an image reconstruction algorithm is to replace the PP sensor in the APDI system described in Chapter V with an imaging polarimeter. The intensities for the two polarizations can then be used as PP intensities in the APDI algorithm without modification. The remaining step is to form a statistical description for the polarization phase that will fit in the framework of the APDI algorithm.

Rather than work directly with the (unknown) joint PDF of the polarization phase, the spatial PSD of the polarization phase is used. Though there is no known analytic expression for the distributions of the PSD or autocorrelation, the definition of the PSD as the modulus square of the Fourier transform can be used to develop an approximate form. As before, the Fourier transform is written in integral form, and the power spectrum of the polarization phase is given by

$$d_{12}(x,y) = \left| \iint_{-\infty}^{\infty} \theta_{12}(u,v) \exp\left[-j2\pi\left(ux+vy\right)\right] \mathrm{d}u\mathrm{d}v \right|^2 \tag{7.65}$$

Assuming for simplicity that the θ_{12} values are independent and viewing the integral as an infinite sum of randomly weighted unit-magnitude complex phasors, the central limit theorem indicates that the integral will be a complex Gaussian random variable. Applying the same transformations used for (3.28), the power spectrum will be exponentially distributed. As with the PP-PSD data, a histographic analysis agrees with exponential statistics. To maintain tractability, spatial independence is again assumed.

As an indication of the object information contained in d_{12}, the truth object and 100 averaged simulation realizations are shown in Fig. 7.4. To relate d_{12} to the object being imaged, define

$$\mu'(\Delta\xi, \Delta\eta) = \mathrm{E}\left[\theta_{12}(\xi_1, \eta_1)\,\theta_{12}(\xi_2, \eta_2)\right] \tag{7.66}$$

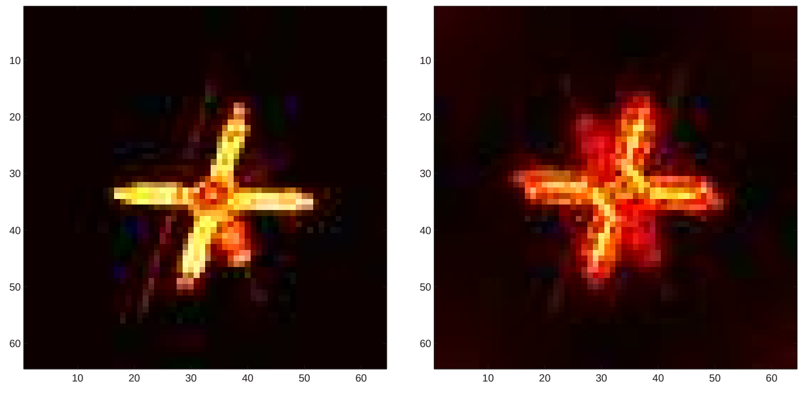
where $\mathrm{E}\left[\theta_{12}(\xi_1, \eta_1)\,\theta_{12}(\xi_2, \eta_2)\right]$ is a function of μ and ψ, and is shown in Fig. 7.1. Applying the Weiner-Khintchine theorem to relate the autocorrelation and the PSD such that the noiseless model data is given by

$$g_{12}(x, y) = \mathcal{F}\{\mu'(\Delta\xi, \Delta\eta)\} \tag{7.67}$$

where μ' is as given in (7.66). As previously stated, μ_c is a normalized Fourier transform of the object given by

$$\mu_c = \frac{\mathcal{F}\{\mathbf{o}(x, y)\}}{\sum_{\xi,\eta} \mathbf{o}(\xi, \eta)} \tag{7.68}$$

and $\mu = |\mu_c|$ and $\psi = \angle\mu_c$.



(a) Truth (b) Average PSD

Figure 7.4: (a) Truth object, and (b) 100 averaged polarization phase PSD realizations. It's clear that a large amount of object information is encoded in the polarization phase PSD.

Statistical independence between data frames is assumed, and the joint distribution for the polarization phase PSD is given by

$$p\left(\mathcal{D}_{12}\right) = \prod_{f,x,y}\left[\frac{1}{g_{12}\left(x,y\right)} + \frac{d_{12}\left(x,y\right)}{g_{12}\left(x,y\right)}\right] \tag{7.69}$$

leading to a log-likelihood given by

$$\mathcal{L}\left(\mathbf{o}\right) = \sum_{f,x,y}\left[\ln g_{12}\left(x,y\right) + \frac{d_{12}^{f}\left(x,y\right)}{g_{12}\left(x,y\right)}\right] \tag{7.70}$$

The only remaining pieces required for incorporation into the APDI algorithm are development of a method for efficiently determining $\mu'\left(\mu,\psi\right)$, and computation of the gradient of g_{12} with respect to the object pixels and aberration parameters. Because the polarization phase is insensitive to the aberration, the partial derivative with respect to the aberration parameters is zero.

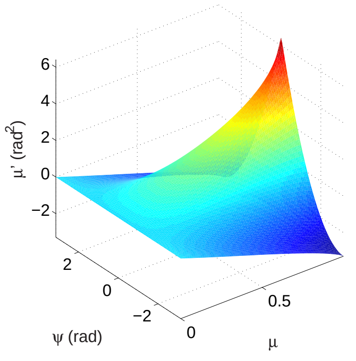
Initial attempts to fit $\mu'\left(\mu,\psi\right)$ to analytical expressions including Gaussian, Lorentzian, and cosine series didn't produce a good fit. Furthermore, evaluation of the partial derivatives of the fit were unacceptably computationally expensive. As an alternative, μ' and its partial derivatives with respect to μ and ψ were numerically evaluated by integrating (3.45) and its derivatives on a linearly spaced 256×256 grid covering the ranges $(-\pi < \psi < \pi)$ and $(0 \leq \mu < 1)$. The results of the numeric integration are shown in Fig. 7.5. The mapping between μ_c and μ', and the gradients of μ' with respect to μ and ψ were obtained by performing a 2-D linear interpolation using nearest neighbor points.

The remaining pieces required for integration are the partial derivatives of μ and ψ with respect to a given object pixel. Using the definition of the discrete Fourier transform,

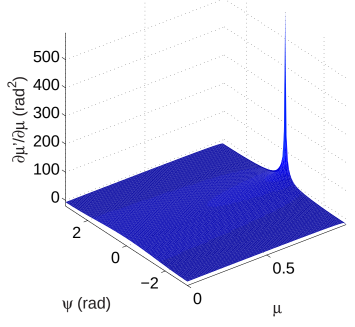
$$\mu_c\left(x,y\right) = \frac{1}{\sum_{\xi,\eta}\mathbf{o}\left(\xi,\eta\right)}\sum_{u,v}\mathbf{o}\left(u,v\right)\exp\left[-\frac{j2\pi}{N}\left(ux+vy\right)\right] \tag{7.71}$$

and defining

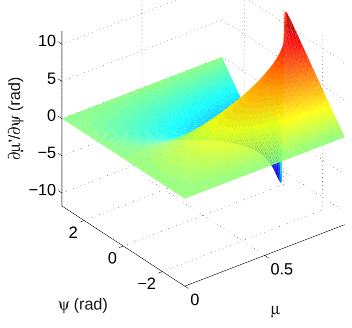
$$\mu = \left(\mu_c^{*}\mu_c\right)^{1/2} \tag{7.72}$$



(a) $\mu'(\mu,\psi)$



(b) $\frac{\partial}{\partial\mu}\mu'(\mu,\psi)$



(c) $\frac{\partial}{\partial\psi}\mu'(\mu,\psi)$

Figure 7.5: Numerical evaluation of $\mu'(\mu,\psi)$, $\partial\mu'/\partial\mu$, and $\partial\mu'/\psi$. 2-D linear interpolation is used for μ and ψ values that don't correspond to grid points.

and

$$\psi = \tan^{-1}\left(\frac{\mathcal{Im}\{\mu_c\}}{\mathcal{Re}\{\mu_c\}}\right) \tag{7.73}$$

the partial derivatives of μ and ψ with respect to object pixel $\mathbf{o}(x_o, y_o)$ are given by

$$\frac{\partial\mu(x,y)}{\partial\mathbf{o}(x_0,y_0)} = \frac{1}{\sum_{\xi,\eta}\mathbf{o}(\xi,\eta)}\left[\cos\left(\frac{2\pi}{N}(x_0x+y_0y)+\psi(x,y)\right)-\mu(x,y)\right] \tag{7.74}$$

and

$$\frac{\partial\psi(x,y)}{\partial\mathbf{o}(x_0,y_0)} = \frac{1}{\sum_{\xi,\eta}\mathbf{o}(\xi,\eta)}\frac{-\sin\left(\frac{2\pi}{N}(x_0x+y_0y)+\psi(x,y)\right)}{\mu(x,y)} \tag{7.75}$$

Finally, to produce the required partial derivatives (7.74) and (7.75) were passed through a Fourier transform such that

$$\frac{\partial g_{12}(x,y)}{\partial\mathbf{o}(x_0,y_0)} = \mathcal{F}\left\{\frac{\partial\mu'}{\partial\mu}\frac{\partial\mu}{\partial\mathbf{o}(x_0,y_0)}+\frac{\partial\mu'}{\partial\psi}\frac{\partial\psi}{\partial\mathbf{o}(x_0,y_0)}\right\} \tag{7.76}$$

This regularization term was coded and included in a modified Polarization Enhanced (POLE) APDI algorithm using only PP and FP measurements. Because the regularization term requires $O(N^2)$ computations per iteration whereas the remaining terms are computed using FFTs with $O(N\log N)$ computations, the regularization is rather expensive. To determine if the cost of this regularization term is worthy of further investigation, 100 sets of 30-frame data were generated with a $D/r_0 = 15$ and assuming perfect detection, then processed with and without polarization phase regularization and without DP data. Results of the simulations are shown in Table 7.1. Introduction of the polarization phase term causes a significant increase in both computation time and MSE.

Table 7.1: Performance comparison between POLE and FP-PP APDI

Metric	POLE APDI	FP-PP APDI
MSE	4.1032	1.7865
Time (s)	5523.2	304.37
Function Eval's	181	367

Even though it appears that much information is contained in the regularization term, this implementation has an overall negative effect. The approximations made for the functional form of μ' and its gradients likely caused deviations from the desired behavior. If this form of regularization is to be used, a better statistical relationship between the object and the polarization phase must be found. If an inexpensive analytic solution were found, the error caused by approximation would be reduced and computational burden could be less of a factor. However, because the APDI algorithm performs exceptionally well as is, the trade-off in signal strength implied by addition of a 4-channel detector in the PP would likely drown out any benefit gained.

7.4 Future Directions

In order to reconstruct images from only PP polarization data, an alternative approach must be found. The computational burden associated with the methods developed here make them impractical. Ideally, the EM-algorithm approach should be simplified and made tractable. There is likely remaining structure and redundancy in the problem that can be used to cut down the costs and make it a practical alternative.

For the LAPDI algorithm, a possible avenue for progression might be to investigate the ability of alternative basis sets to efficiently represent the propagation phase with fewer terms. Looking at things like JPEG compression techniques, Huffman coding, or other similar techniques to represent the phase with fewer terms might help to reduce the number of phase variables. If the number of problem unknowns can be reduced, the LAPDI approach will become better conditioned, less costly, and more likely to converge to an acceptable solution. In addition to considering additional basis sets, introduction of auxiliary information about the object such as physical support like that done for correlography [17] might provide the extra information needed to lead to proper convergence.

Finally, if an inexpensive and accurate model for the polarization phase PSD and its gradients can be found, inclusion of PP data in the POLE algorithm might produce better results. To evaluate the possibility of improved reconstructions with a better model, the numeric integrations used to produce the mapping and gradients shown in Fig. 7.5 should be evaluated on a much finer mesh and with tighter tolerance on the numeric integration. This was attempted, but was aborted due to time constraints and equipment malfunctions.

VIII. Conclusions

Coherence properties of laser illumination were used to develop an algorithm for producing imagery of objects using laser illumination. A maximum-likelihood multi-frame active phase-diversity imaging algorithm was derived for coherent statistics induced as a consequence of active illumination. A statistical model for object information encoded in the pupil-plane intensity distribution was developed and incorporated into the derivation. The resulting algorithm was characterized by use of Monte Carlo simulations and shown to perform exceptionally well. Pupil-plane data was shown to have a dramatic positive impact on algorithm performance and be a major contributor to overall system performance. Provided that the algorithm functions as well operationally as it does in simulation, it will be a unique and invaluable tool for imaging satellites or other space-based objects obscured by the earth's shadow and will expand overall Space Situational Awareness.

In addition to developing a functioning algorithm for use with conventional measurement techniques, approaches for using polarization data were developed and presented; establishing a baseline for future work. The groundwork for an EM algorithm to directly estimate the object from polarization measurements was laid, but proved too cumbersome to implement. The APDI algorithm was modified to process polarization state data independent of other measurements, but proved too ill-conditioned to produce satisfactory results. Finally, the APDI algorithm was extended to utilize polarization state measurements in place of the pupil-plane measurements, and areas ripe for improvement were identified.

8.1 Summary of Results

Monte Carlo analysis of the APDI algorithm showed that the impact of target object, number of data frames, detection SNR, and detector array size on the optimal convergence tolerance is minimal. Under static aberrations, the optimal convergence tolerance was insensitive to variations in system parameters. For the dynamic aberration case, the optimal convergence tolerance was independent of all but the detector array size, indicating that minimal characterization is required for a specific system configuration.

For both the dynamic and static aberration cases, the optimal conditioning bias was found to dramatically impact computation time, and to a lesser extent MSE. For the static aberration case, conditioning biases below 10% of the average frame intensity resulted in worse reconstructions and longer computation times as the problem became less well-conditioned. For conditioning biases above 10% of the average frame intensity, the residual error climbed steadily and the computation time dropped quickly as the objective function moved further away from the statistically derived objective function and became better conditioned. For the static aberration case, the optimal conditioning bias was found to be 10%.

For the dynamic aberration case, anomalous results for bias values of 40% and 10% are unexplained. However, the general trend is consistent with that seen in the static aberration data with the exceptions that the increase for smaller biases begins at larger values and is slower than with the static aberration case, and that the increase in MSE with increasing bias is less pronounced. For the dynamic aberration case, the optimal bias was found to be 50%.

The APDI algorithm is robust under a broad range of scenarios. With over 7,000 realizations each for static and dynamic aberrations including realizations with as few as ten data frames and detection SNR values as low as 2, none of the dynamic-aberration reconstructions converged to an image that was worse than the average of the focal-plane data. For static aberrations, roughly 3% of reconstructions resulted in an MSE that was higher for the reconstruction than the averaged data. However, in most of those cases a visual comparison of the reconstruction with the averaged raw data showed that the reconstruction was significantly better than the raw data in spite of the poor quality metric. In all other cases, the judgment of whether or not a reconstruction was "worse" than the raw data became somewhat subjective.

Simulations using varying detection noise conditions, including both photon and thermally dominated noise sources, showed that the algorithm is essentially insensitive to noise for SNR values above approximately 7, and that it still performs relatively well for SNR values as low as 2. Given that the algorithm is designed to deal with data that has a noiseless-detection SNR of 1, it's no surprise that it performs well under noisy-detection/low-light conditions.

Comparing results from the PP-FP APDI and DP-FP APDI algorithms with the complete APDI algorithm, it's clear that the PP data has a much more significant impact on estimator performance than the conventionally used DP data. Reconstructions done without PP data included resulted in 148% and 54% increases in average MSE compared with the Full APDI algorithm for the dynamic and static aberration cases respectively. This shows that the PP data has a dramatic impact on overall performance of the reconstruction.

Exclusion of the DP data from the APDI algorithm resulted in only a 11% and 24% increase in MSE for the static and dynamic aberration cases respectively. In cases where low-light levels force a poor SNR, the marginal gain in performance obtained by using all three data planes may be swamped by the decrease in performance caused by a low detection SNR. This indicates that a two-channel system comprised of a FP and PP sensor might be optimal for these kinds of scenarios. Even when SNR isn't a factor, the additional computational burden and hardware requirements for a complete three-channel system may not be justified by the marginal gain in system performance.

The framework developed for an EM algorithm proved too complex and computationally expensive to be implementable. Even after making simplifying assumptions, the maximization step for the unknown correlation phase required $O(N^3)$ integrations per point separation, resulting in $O(N^4)$ numeric integrations per iteration. Without development of a simplified statistical model or derivation of an analytic maximum, the EM algorithm approach is intractable and impractical.

Attempts to use an APDI-type algorithm to post-process PP polarization state measurements did not result in satisfactory results. As implemented, the algorithm is poorly conditioned and converges to local maxima that are not satisfactory reconstructions of the object. To illustrate the poorly conditioned nature of the problem, F frames of $N \times N$ PP polarization state data consists of $3FN^2$ extremely noisy data values for a problem with $(F+1)\,N^2$ unknowns. In a noiseless world, this should be sufficient to solve the system. However, in the presence of noise, it appears that more measurements per unknown are required for convergence to an adequate solution. Because every additional frame of $N \times N$ data introduces $3N^2$ data points and an additional N^2 unknowns, the problem will not be significantly better conditioned by simply adding additional frames.

In addition to being ill-conditioned, the computational burden for making each phase pixel an optimization variable is considerable. For the original APDI implementation, each iteration requires $(6M + 12)\,F + 5$ FFTs, where M is the number of modes being compensated. This translates to to 3065 FFTs per iteration for a 30-frame data set of any size with 15 modes per frame. By contrast, the LAPDI algorithm requires $(4N^2 + 12)F + 5$ FFTs for each iteration, translating to roughly $5{\times}10^5$ FFTs per iteration for a 30-frame 64×64 pixel data set, and $2{\times}10^6$ for a 30-frame 128×128 pixel data set. If this approach is to work, a condensed form for the unknown phase, similar to the Zernike representation used for the unknown turbulence phase in the APDI algorithm, must be developed to reduce the parameter space and allow for a better conditioned problem. Alternatively, additional auxiliary information about the object being reconstructed must be included in the algorithm.

While the original APDI algorithm was demonstrated to produce good results, inclusion of PP polarization data did not result in better performance. Because of mathematical difficulties with the statistical description of the polarization data, approximations were made which adversely impacted estimator performance. Addition of the polarization data to the estimator resulted in fewer function evaluations, but the additional computational burden more than counterbalanced this improvement. Furthermore, the resulting reconstructions were significantly worse than those obtained using the original APDI algorithm. If polarization data is to be used in place of PP intensity measurements in the APDI algorithm, a better statistical description and/or a formulation with a smaller computational burden must be developed and incorporated into the estimator.

8.2 Significant Contributions

The following items summarize the significant contributions, extensions, and developments made as a result of this investigation:

1. The first phase diversity algorithm utilizing PP intensity measurements and designed specifically to deal with the unique statistics of data from an active imaging system was developed. This advancement will enable imaging of exo-atmospheric objects over a broad range of engagement scenarios that are impossible using conventional imaging techniques.

2. It was demonstrated that for image reconstruction, PP data has a more significant impact on performance than the conventional DP data. PP data provides a 34% reduction in MSE with static aberrations and a 60% reduction in MSE with dynamic aberrations over a conventional FP-DP phase diversity geometry. Additionally, simultaneous utilization of FP, DP, and PP data planes produced results that were only marginally better than those obtained with just a FP-PP configuration.
3. It was demonstrated that the APDI algorithm is robust under a broad range of detection SNR, and performance was characterized under a range of scenarios. The optimal conditioning bias and stopping criteria were found, and were shown to be essentially insensitive to system configuration and operating conditions.
4. Multiple frameworks for utilizing polarization data to improve image reconstruction from actively illuminated targets were developed, and areas worthy of future consideration were identified.

8.3 Future Work

Opportunities for future work can be broken into two major areas: continued work on the APDI algorithm and work related to using PP polarization data for imaging. Primary tasks inviting future work on the APDI algorithm were identified and include:

1. Modify the PP-data model to better account for non-square and partially obscured pupil functions.
2. Design and construct a laboratory experiment to test the ADPI algorithm with laboratory data.
3. Perform a Cramér-Rao bound analysis on the estimator.

The most promising areas for future work related to image reconstruction from PP-polarization data include:

1. Numerically evaluate the mean and gradients of the polarization phase autocorrelation on a finer mesh for use in the POLE algorithm and compare with results from the APDI algorithm.

2. Derive an analytic expression for the polarization-phase autocorrelation and its gradients for use in the POLE algorithm.

3. Develop an inexpensive approximation for the polarization-phase and its gradients for use in the POLE algorithm and compare with the APDI algorithm.

4. Incorporate prior information into the LAPDI algorithm to better condition the reconstruction.

5. Investigate techniques for reducing the number of parameters required to represent the phase in the LAPDI algorithm including evaluationg basis-sets other than Zernike modes for use in representing the unknown propagation phase.

6. Simplify and implement the PP-EM algorithm.

8.4 Summary

As a consequence of this work, A multi-frame active phase diversity imaging (APDI) algorithm was derived for coherent light statistics and demonstrated. In addition to conventional focal-plane and diversity-plane data, a statistical description for pupil-plane (PP) intensity was formed and included in the derivation. The algorithm was implemented and characterized via Monte Carlo simulation. Analysis showed that the algorithm is robust, insensitive to detection noise for SNR ≥ 7, performs well for SNR's as low as 2, and that the effect of system configuration on optimal parameters is minimal. Furthermore, introduction of PP data resulted in a 60% better reconstruction from dynamically aberrated data than obtained using only focal-plane and diversity-plane data. Both an EM-algorithm and a lensless-APDI approach were presented for generating imagery directly from PP polarization measurements. However, both approaches are currently impractical. Suggestions for improvement were offered. Finally, the APDI algorithm was modified to use PP polarization data in place of PP intensities. An initial statistical model was offered, and suggestions for performance improvement were presented.

This appendix contains the code sections used to build the APDI algorithm. The algorithm was built using Gnu Make and gcc 3.4.6. Both the FFTW3 and gsl software packages are required to build the executable. If the phase regularization term is disabled, gsl can be removed from the build. The `routines.f` file containing the L-BFGS-B code must be placed in the same directory as `objective_solver.f`.

Listing A.1: `Makefile`

```
# set up directory variables
#HOME=~pjohnson
CROOT=.
FROOT=$(CROOT)/solver
MLIB=$(CROOT)/lib

# set up the relative paths
psd_reg=$(CROOT)/psd_reg
solver=$(FROOT)
objctive=$(CROOT)/regd_objective
myhdrs=$(CROOT)/lib
core_reg=$(CROOT)/core_objective
phase_reg=$(CROOT)/phs_reg

# set up compilers and flags...
CC=gcc
FC=g77
CFLAGS=-O3 -march=prescott -Wall -std=c99
FFLAGS=-O3 -march=prescott -Wall
#CFLAGS=-g
#FFLAGS=-g
LIBS=-L$(HOME)/lib/ -L/usr/local/lib
INC=-lfftw3 -lm -lgsl -lgslcblas
FINC=-lfrtbegin -lg2c -lm -lgcc_s -lgcc -lc
SEARCH=-I$(myhdrs) -I$(psd_reg) -I$(solver) -I$(objctv) -I$(core_reg)\
        -I$(CROOT) -I$(phase_reg)

# set up other things that we'll use...
mylib=$(MLIB)/mylibs
psdreg=$(psd_reg)/psd_reg
objctv=$(objctive)/regd_objective
slvr=$(solver)/objective_solver
main=$(solver)/freestanding_solver
routines=$(solver)/routines
core=$(core_reg)/core_objective
zerns=$(MLIB)/zernikes
phs_reg=$(phase_reg)/phs_reg

# define the objects needed to build APDIsolve
OBJS=$(mylib).o $(psdreg).o $(objctv).o $(slvr).o $(main).o $(routines).o \
        $(core).o $(zerns).o $(phs_reg).o

# now set up what we need to build the program
APDIsolve: $(OBJS)
        $(CC) $(OBJS) $(CFLAGS) $(INC) $(FINC) $(SEARCH) $(LIBS) -o APDIsolve

$(phs_reg).o:$(phs_reg).c $(phs_reg).h
        $(CC) -c $(phs_reg).c $(CFLAGS) $(SEARCH) -o $(phs_reg).o

$(psdreg).o:$(psdreg).c $(psdreg).h $(mylib).o
        $(CC) -c $(psdreg).c $(CFLAGS) $(SEARCH) -o $(psdreg).o
```

```
$(threg).o:$(threg).c $(mylib).o
	$(CC) -c $(threg).c $(CFLAGS) $(SEARCH) -o $(threg).o

$(slvr).o:$(slvr).f $(routines).o
	$(FC) -c $(slvr).f $(FFLAGS) $(SEARCH) -o $(slvr).o

$(routines).o:$(routines).f
	$(FC) -c $(routines).f $(FFLAGS) $(SEARCH) -o $(routines).o

$(objctv).o:$(objctv).c $(objctv).h
	$(CC) -c $(objctv).c $(CFLAGS) $(SEARCH) -o $(objctv).o

$(main).o:$(main).c
	$(CC) -c $(main).c $(CFLAGS) $(SEARCH) -o $(main).o

$(core).o:$(core).c $(core).h
	$(CC) -c $(core).c $(CFLAGS) $(SEARCH) -o $(core).o

$(mylib).o:$(mylib).c
	$(CC) -c $(mylib).c $(CFLAGS) $(SEARCH) -o $(mylib).o

$(zerns).o:$(zerns).c
	$(CC) -c $(zerns).c $(CFLAGS) $(SEARCH) -o $(zerns).o


.PHONY: clean
clean:
	rm $(OBJS) APDIsolve
```

Listing A.2: `config.h`

```
/* Peter Johnson
 * 6 Sep 2006
 *
 * config.h
 *
 * this file is for configuration options that affect all of the modules of
 * the APDI algorithm...
 */

// decide if this is the top-level or not for global variables...
#ifdef LOCAL
double bias=0.5;
double *inverse;
int inverse_exists=0;
#else
extern double bias;
extern double *inverse;
extern int inverse_exists;
#endif
```

Listing A.3: `freestanding_solver.c`

```
/*==================================================================
 * Peter Johnson
 * 16 Feb 2006
 *
 * this program is for implementing the APDI imaging algorithm
 * without matlab.   The motiviation for this is for debugging and
 * speed.  Output from the L-BFGS-B algorithm cannot be fed into
 * matlab, and seg-faults and the like are difficult to debug when
 * running under matlab.  Instead, the simulated data are pre-computed
 * with matlab, and loaded via data files here.
 *=================================================================*/
#define LOCAL
#include "config.h"
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


/* include the prototype for the fortan solving function */
void apdslv_(double *L, double *guess, double *G, double *data,
             int *frms, double *div, int *dim, int *k, double *Dr0,
             double *tol );

// prototype for the help print function
void prnthelp();

int main(int argc, char *argv[])
{

  FILE *out_fl=NULL, *data_fl=NULL;              // file pointers
  FILE *guess_fl=NULL;                           // file pointers

  size_t dbl=sizeof(double);                     // sizes
  size_t cntr;                                   // counter

  int m, k, f, mm;                               // objective parameters
  int fl_error;                                  // file error flag
  int c0;                                        // loop counter
  int runk=0;                                    // aberrations to correct

  extern double bias;                            // conditioning bias
  double tol;                                    // tolerance for convergence
  double L, div, Dr0, Dr0d;                      // objective variables
  double *guess, *G, *data;                      // ptr to data, guess, grad
  double tmp1;

  char outfl[256], dtafl[256], gssfl[256];      // filenames

  // set the default command-line parameters
  bias = 0.5;
  tol = 2e-7;
  strcpy(outfl,"results.dat");
  strcpy(dtafl,"data.dat");
  strcpy(gssfl,"guess.dat");
  runk = 15;
  Dr0 = 0;

  ///////////////////////////////////////////////////////////////////////////
  // parse the command-line arguments
  ///////////////////////////////////////////////////////////////////////////

  if ( argc >1 )
    {
      for (c0=1; c0<argc; c0++)
        {
          // see if we were passed a bias
          if (!strcmp(argv[c0],"-bias"))
            {
              bias = atof(argv[c0+1]);  // convert bias to a double
              c0++;
            }

          // see if we were passed a convergence tolerance
          else if (!strcmp(argv[c0],"-tol"))
            {
              tol = atof(argv[c0+1]);   // convert tolerance to a double
              c0++;
            }
          // see if we were passed an output filename
          else if (!strcmp(argv[c0],"-o"))
            {
              cntr = strlen(argv[c0+1]);
              if (cntr<=256)
                strcpy(outfl, argv[c0+1]);
              c0++;
            }

          // see if we were passed an initial guess file name
          else if (!strcmp(argv[c0],"-guess"))
            {
              cntr = strlen(argv[c0+1]);
              if (cntr<=256)
                strcpy(gssfl, argv[c0+1]);
              c0++;
            }

          // see if we were passed a datafile name
          else if (!strcmp(argv[c0],"-data"))
            {
              cntr = strlen(argv[c0+1]);
              if (cntr<256)
                strcpy(dtafl, argv[c0+1]);
              c0++;
            }

          // see if we were passed a number of modes to correct
          else if (!strcmp(argv[c0],"-modes"))
            {
              runk = atoi(argv[c0+1]);   // convert modes to int
              c0++;
            }

          // see if we were passed a Dr0 to run for aberr regulation
          else if (!strcmp(argv[c0],"-Dr0"))
            {
              Dr0 = atof(argv[c0+1]);   // convert Dr0 to a double
              c0++;
            }

          // print help message if asked for
          else if (!strcmp(argv[c0],"-help") || !strcmp(argv[c0],"-h"))
            {
              prnthelp();
              return 0;
            }

          // print help message if we're confused
          else
            {
              printf("\nInvalid Command Line Option\n\n");
              prnthelp();
              return 0;
            }
        }
      printf("\n");
      printf("Bias = %g\n",bias);
      printf("Tolerance = %g\n",tol);
      printf("Output File = %s\n",outfl);
      printf("Guess File = %s\n",gssfl);
      printf("Data File = %s\n",dtafl);

    }

  ///////////////////////////////////////////////////////////////////////////
```

```
// load the data
///////////////////////////////////////////////////////////////////////////

//open the data file
data_fl = fopen(dtafl,"r");

if (data_fl == NULL)
  {
    printf("\nError: Could not open the data file for reading\n");
    return 1;
  }

// load the data from the file...
fread(&tmp1, dbl, 1, data_fl);      // read the array dim
m = (int)tmp1;                      // cast it to an int
fread(&tmp1, dbl, 1, data_fl);      // get number of frames
f = (int)tmp1;                      // cast it
fread(&tmp1, dbl, 1, data_fl);      // read # of aberrations
k = (int)tmp1;                      // cast it
fread(&Dr0d, dbl, 1, data_fl);       // get D/r0
fread(&div, dbl, 1, data_fl);       // read the diversity factor
mm = m*m;                           // compute number of pixels
data = malloc((2*f+1)*mm*dbl);      // allocate data array
cntr = fread(data, dbl, (2*f+1)*mm, data_fl); // read in data frames
if( cntr != (2*f+1)*mm)
  {
    printf("\nError: Couldn't read right number of pixels from file");
    printf("\n\t%d elements requested, %d elements read\n", (2*f+1)*mm,
           cntr);
    return 1;
  }

// close the data file
fl_error = fclose(data_fl);
if (fl_error)
  {
    printf("\nError closing data file\n");
    return 1;
  }

//////////////////////////////////////////////////////////////////////////
// load or generate the initial guess data
//////////////////////////////////////////////////////////////////////////

// try to open the guess file
guess_fl = fopen(gssfl,"r");

// if we can't open the guess file, generate a starting guess
if (guess_fl == NULL)
  {
    printf("\nCouldn't open guess file, generating guess internally\n");
    guess = calloc((mm+f*k),dbl);      // allocate memory for the guess

    // set obj pixels to a constant unit-average
    for (c0=0; c0<mm; c0++)
      *(guess+c0) = 1.0;              // constant initial obj guess

    // set the number of modes if called for
    if (runk>0)
      k = runk;

    // allocate memory doe the guess
    guess = calloc((mm+f*k),dbl);

    // set the aberration guess to zero
    for (c0=mm; c0<(mm+f*k); c0++)
```

```
        *(guess+c0) = 0;                 // zero aberration initial guess
    }

  // otherwise open the data file and read in the guess
  else
    {
      printf("\nReading guess data from %s\n",gssfl);

      // read and verify m
      fread(&tmp1, dbl, 1, guess_fl);
      if ((int)tmp1 != m)
        {
          printf("\nError: Object dimensions don't match data dimensions");
          return 1;
        }

      // read and verify f
      fread(&tmp1, dbl, 1, guess_fl);
      if((int)tmp1 != f)
        {
          printf("\nError: Different number of frames from data file");
          return 1;
        }

      // read k don't need to verify... will work with whatever is provided
      fread(&tmp1, dbl, 1, guess_fl);

      // read and verify Dr0
      fread(&tmp1, dbl, 1, guess_fl);
      if(tmp1 != 0 && tmp1 != Dr0d)
        {
          printf("\nError: D/r0 is different or not zero");
          printf("\n\tData => %g\nGuess => %g",Dr0d,tmp1);
          return 1;
        }
      if (Dr0==-1)  // use the data Dr0 to run if called for
        Dr0 = Dr0d;

      // read and verify div
      fread(&tmp1, dbl, 1, guess_fl);
      if(tmp1 != div)
        {
          printf("\nError: Diversity is different from data file");
          printf("\n\tData => %g\nGuess => %g",div,tmp1);
          return 1;
        }

      // allocate memory for the guess and read it in
      guess = malloc((mm+f*k)*dbl); // allocate guess array
      cntr = fread(guess, dbl, mm+f*k, guess_fl); // read in the guess
      if (cntr != mm+f*k)
        {
          printf("\nError: Could not read from data file");
          printf("\n\t%d elements requested, %d elements read",mm+f*k, cntr);
          return 1;
        }

      // close the data file
      fl_error = fclose(guess_fl);
      if (fl_error)
        {
          printf("\nError: Error closing connections to guess file.\n");
          return 1;
        }
    }
```

```
////////////////////////////////////////////////////////////////////////////
// call the solver routine...
////////////////////////////////////////////////////////////////////////////

// put out some info
printf("\nImage is %d by %d",m,m);
printf("\nProcessing %d frames", f);
printf("\nProcessing %d aberrations per frame",k);
printf("\nUsing D/r0 = %g",Dr0);
printf("\nUsing %g waves of diversity\n",div);

// allocate the gradient array
G = malloc((mm+f*k)*dbl);

// call the solver routine
apdslv_(&L, guess, G, data, &f, &div, &m, &k, &Dr0, &tol);

////////////////////////////////////////////////////////////////////////////
// write the results to the output file
////////////////////////////////////////////////////////////////////////////
out_fl = fopen(outfl,"w");
if(out_fl==NULL)
  {
    printf("\nError: Could not open output file for writing\n");
    return 1;
  }
else
  printf("\nWriting results to output file\n");

tmp1 = m;
cntr = fwrite(&tmp1, dbl, 1, out_fl);
if(cntr != 1)
  {
    printf("\nError: Could not write to output file\n");
    return 1;
  }
else printf("\nm = %d",m);

tmp1 = f;
cntr = fwrite(&tmp1, dbl, 1, out_fl);
if(cntr != 1)
  {
    printf("\nError: Could not write to output file\n");
    return 1;
  }
else printf("\nf = %d",f);

tmp1 = k;
cntr = fwrite(&tmp1, dbl, 1, out_fl);
if(cntr != 1)
  {
    printf("\nError: Could not write to output file\n");
    return 1;
  }
else printf("\nk = %d",k);

cntr = fwrite(&Dr0d, dbl, 1, out_fl);
if(cntr != 1)
  {
    printf("\nError: Could not write to output file\n");
    return 1;
  }
else printf("\nD/r0 = %g",Dr0d);

cntr = fwrite(&div, dbl, 1, out_fl);
if(cntr != 1)
```

```
    {
      printf("\nError: Could not write to output file\n");
      return 1;
    }
  else printf("\nDiversity = %g",div);

  cntr = fwrite(guess, dbl, mm+f*k, out_fl);
  if(cntr != mm+f*k)
    {
      printf("\nError: Could not write to output file\n");
      return 1;
    }
  else printf("\nWrote %d object and aberration mm\n",cntr);

  // close the output file
  fl_error = fclose(out_fl);
  if(fl_error)
    {
      printf("\nError: Error closing connections to data files.\n");
      return 1;
    }

  // free up working arrays
  free(G);
  free(data);
  free(guess);

  return 0;
}

void prnthelp()
{
  printf("\n input options:\n");
  printf("\t-bias [double] == bias offset for regularizaiton\n");
  printf("\t-tol [double] == convergence tolerance\n");
  printf("\t-o [filename] filename for the resulting output data\n");
  printf("\t-guess [filename] filename with initial guess data in it\n");
  printf("\t-data [filename] filename with measured data in it\n");
  printf("\t-modes [int] == number of Zernike modes to correct. This\n");
  printf("\t              can't be used with an external starting guess\n");
  printf("\t              If a guess is supplied, it's mode number wins.\n");
  printf("\t-Dr0 [double] == Dr0 for running phase regularization term.\n");
  printf("\t              set to -1 to use Dr0 from the data");
  printf("\t              set to 0 to disable regularization (default)");
  printf("\t-help == print this message\n");
  printf("\t-h == print this message\n");

}
```

Listing A.4: `objective_solver.f`

```
C     Peter Johnson
C     15 Feb 2006
C
C     This section of fortran code is intended to call the objective
C     function and the numerical solver routine.  The objective function
C     is written in C, but should be callable from fortran as-is.  The
C     numerical solver routine is the LBGFS-B code available from
C     argonne national labs, and is used as-is.  This file is based on
C     the driver1.f file that is provided with the numerical solver
C     package.

C     This currently can handle a 128x128 pixel array with up to a total
C     of 128^2 aberration parameteres evenly distributed across all the
C     frames.  To expand that, increase nmax below.
```

```
      subroutine apdslv(f, x, g, dta, frms, div, dim, k, Dr0, mytol)

      integer frms, dim, k;
      double precision f, x(dim*dim+frms*k), g(dim*dim+frms*k),
     +     dta((2*frms+1)*dim*dim), div, Dr0, mytol

C     set up some solver parameters

      integer  pixels
      integer nmax, mmax
      parameter (nmax=32786, mmax=15)
C     mmax is the dimension of the largest problem to be solved
C     nmax is the maximum number of limited memory corrections

C     Declair variables needed by the solver...
      character*60     task, csave
      logical  lsave(4)
      integer n, m, iprint,
     +        nbd(nmax), iwa(3*nmax), isave(44)
      double precision factr, pgtol,
     +                 l(nmax), u(nmax), dsave(29),
     +                 wa(2*mmax*nmax+4*nmax+12*mmax*mmax+12*mmax)
      character*10  interI, interG

C     declair any additional variables...
      integer c0, iter
      double precision fold, myfctr

      iter = 1
      c0 = 0
      fold = 0
      interI='i0000.dat'
      interG='g0000.dat'

C     squelch output...
C      iprint = -1
C     or output every iteration
      iprint = 1


C     specify the tolerances in the stopping criteria
C      factr = frms*1.0d+10
C      pgtol = 1.0d-3
      factr = 0
      pgtol = 0
      myfctr = mytol

C     specify the size of the problem and the number of corrections
      pixels=dim*dim
      n = pixels+frms*k
      m = 15

C     now set up nbd, and the upper and lower bounds on the variables
C     nbd(i) is the type of bound for this variable
C     nbd = 0 => no bound 1 => lower bound 2=> upper and lower bound 3
C     => upper bound only
C
C     l(i) is the lower bound on the i'th variable
C     u(i) is the upper bound on teh i'th variable

C     set the lower-bound on the pixel intensities...

c     lower-bound the image positive
      do 10 c0=1,pixels
         nbd(c0) = 1
         l(c0) = 0
```

```
 10   continue

C      set the aberration variables to be unbounded
       do 12 c0=1,frms*k
          nbd(c0+pixels) = 0
 12   continue

C      Start the iteration by initializing task
       task = 'START'


C      ----------- Begin the solving loop -------------
 110  continue

C      call the L-BFGS-B code
       call setulb(n,m,x,l,u,nbd,f,g,factr,pgtol,wa,iwa,task,iprint,
     +     csave,lsave,isave,dsave)

       if (task(1:2) .eq. 'FG') then

C      if we get to this point it is because the solver wants the
C      objective value and the gradient at the current guess

       f = objctv(g, x, dta, div, frms, k, dim, Dr0)

C      go back to the minimization routine
          goto 110
       endif

       if (task(1:5) .eq. 'NEW_X') then
C      if we make it here, the solver has come up with a new guess and
C      wants to continue execution...

C      kill the iterations if we have evaluated the objective too many times.
          if (isave(34) .ge. 10000)
     +         task='STOP: exceeded 10000 function evals'

C      kill iterations if we don't make enough progress...  have to do it
C      here because machine precision is different across platforms so b/in
C      factr doesn't work

       if (abs(fold-f) .le. abs(f*myfctr))
     +         task="STOP: relative reduction too small to continue"

C      reset fold to the current guess for the next iteration
       fold = f

C      go back to the top of the loop and keep going
          goto 110
       endif

C      if the task isn't NEW_X or 'FG', then we are done, and we stop.

       end subroutine
```

Listing A.5: regd_objective.h

```
/* Peter Johnson
 * 27 Feb 2006
 *
 * This is the header file for the function that is intended to combine all
 * of the regularizaiton and objective terms into one function that can be
 * called from fortran for use with the numerical solver.
 *
 * The measured psd data passed to this function needs to be averaged across
 * all of the frames before being passed to this function.
 *
 * The focal- and diversity plane data should all be passed in as a single
 * array with the focal-plane and diversity plane interleaved in the
 * following order...
 *
 * focal1, div1, focal2, div2, etc...
 *
 * The guess-object pixel values and the unknown phase aberration parameters
 * are contained in a single array with the phase aberration parameters
 * concatenated to the end of the object array.  The gradient array is
 * identically distributed.
 *
 * Because this needs to be callable from fortran, all arguments are passed
 * by reference, and the function name is mangled appropriately
 */


#include "core_objective.h"
#include "psd_reg.h"

double objctv_(double *G, double *guess, double *data, double *div,
               int *f, int *k, int *m, double *Dr0);
/* double  objctv_(double *G, double *guess, double *data, double *div,
 *                 int f, int *k, int *m, double Dr0)
 *
 * This function returns the value of the regularized objective function
 * give the focal-plane, diversity-plane, and pupil-plane data from the
 * active-imaging problem.  The input data array should contain the
 * focal-plane and diversity plane intensity frames interleaved as described
 * above, followed by the final frame of averaged pupil-plane speckle psds.
 * The first 2*f*m^2 elements of the data array contain the focal and
 * diversity terms, and the final m^2 elements contain the averaged
 * pupil-plane speckle psd.
 *
 * The current guess for which the objective function is to be evaluated is
 * contained in the guess array, and consists of the object pixel values in
 * the first m^2 elements, and the aberration parameters in the remaining
 * f*k elements.
 *
 * The known aberration for the diversity plane is described by the value
 * contained in div.  This is the maximum number of wavelengths of phase
 * added to the phase-front in the pupil-plane (i.e. the amount of phase
 * added in the corners of the array).
 *
 * The computed gradient at the current guess is evaluated and returned via
 * the G array.  This has the same structure as the guess array, with the
 * gradients wrt to pixel values returned in the first m^2 elements, and the
 * gradients wrt the aberration parameters given in the final k elements.
 *
 * The estimate for D/r0 needs to be passed in for the phase-regularization
 * to work.  If it is unknown, set D/r0==0 to bypass that part of the
 * regularization.
 */
```

Listing A.6: `regd_objective.c`

```
/* Peter Johnson
 * 27 Feb 2006
 *
 * Because this needs to be callable from fortran, all arguments are passed by
 * reference, and the function name is mangled appropriately
 */

#include "mylibs.h"
#include "regd_objective.h"
#include "phs_reg.h"
```

```
double objctv_(double *G, double *guess, double *data, double *div,
               int *f, int *k, int *m, double *Dr0)
{
  int c0, abers;                      // counter
  int pixels = *m * *m;               // pixels in the object
  double *aberr, *obj;                // pointers to guess obj and aberr
  double *focal, *psd;                // pointers to measured data
  double *G_I, *G_pp, *G_phs, *G_p; // pointers to gradient components
  double L=0;                         // objective function value

  abers = *f * *k;
  // reset the gradient to zero
  for (c0=0; c0<pixels + abers; c0++)
    *(G+c0) = 0;

  // break out the different pieces of the data and the gradients from
  // the passed-in data and guess arrays
  obj = guess;                   // object in the first mxm slots
  aberr = guess+pixels;          // aberration in the last f*k slots
  G_I = G;                       // obj grad in first mxm slots
  G_phs = G+pixels;              // aberr grad in last f*k slots
  focal = data;                  // focal-plane data in first 2*f*m^2 slots
  psd = data+(2*(*f)*pixels);    // psd data in last m^2 data slots

  // call the core objective function to get the initial values
  L = objective(obj, aberr, *div, focal, G_I, G_phs, *f, *k, *m);

  // add the regularization terms

  // do the intenisty regularization
  G_pp = calloc(pixels,sizeof(double));       // allocate working array
  L += *f * psd_reg(obj, psd, G_pp, *m);      // compute the contribution
  for (c0=0; c0<pixels; c0++)                 // add the gradient contrib
    *(G_I+c0) += *f * *(G_pp+c0);

  // do the phase regularization if we know D/r0
  if (*Dr0 != 0)
    {
      G_p = calloc(abers, sizeof(double));       // allocate memory
      L += covar_reg(aberr, G_p, *k, *f, *Dr0); // compute the contribution
      for (c0=0; c0<abers; c0++)                 // add the grad contrib
        *(G_phs+c0) += *(G_p+c0);
      free(G_p);                                 // free memory
    }

  // free working arrays
  free(G_pp);

  // return the objective function value
  return L;
}
```

Listing A.7: `core_objective.h`

```
/* Peter Johnson
 * 25 Jan 2006
 *
 * core_objective.h
 *
 * This file is the header file containing the function prototypes for
 * calling the non-regularized active focal-plane phase-diversity image
 * reconstruction objective function and gradient.
 */
```

```
/* #include "mylibs.h" */

double objective(double *obj, double *aberr, double div, double *data,
                 double *G_I, double *G_phs, int f, int k, int m);
/* ========================================================================
 * double objective(double *obj, double *aberr, double *data, double
 *                   *G, int k, int m);
 *
 * master function for computing the focal-plane PD objective function value
 * and associated gradients.  The function returns the value of the
 * objective function, and stores the gradients wrt to each object pixel in
 * the array pointed to by G.
 *
 * double *obj == pointer to m x m object array (current guess)
 * double *aberr == pointer to the k-element unknown aberration
 *                 parameter array
 * double div == number of waves of KNOWN defocus on the diversity
 *               channel
 * double *data == pointer to m x m x 2 array of avgd speckle data
 *                 the first frame is the avg of the in-focus images,
 *                 and the second is the avg of the diversity images.
 * double *G == pointer to m x m gradient array
 * int f == number of frames of data begin processed
 * int k == number of aberration parameters (zernike modes)
 * int m == number of pixels along each dimension of the input array
 */



void mkscreen(complex *screen, int *msk, int m, double *aberr, int k);
/* ========================================================================
 * void mkscreen(complex *screen, int m, double *aberr, int k);
 *
 * generate an m x m phase screen using the first k zernike modes, with the
 * zernike mode weights contained in the array pointed to by aberr.  Piston
 * is not counted in the aberr array, so the first element of the array
 * corresponds to the weight of the 2nd Noll-indexed zernike mode (tilt).
 *
 * complex *screen == pointer to m*m complex array to store result in int m
 * == number of pixels in each dimension of the array double *aberr ==
 * pointer to Noll-indexed zernike mode weights int k == number of elements
 * in the weight array/number of modes
 */



void conj_dotmultiply(complex *out, complex *in, int m);
/* ========================================================================
 * void conj_dotmultiply(complex *out, complex *in, int m)
 *
 * does a dot-multiply where the result overwrites the first input, and the
 * result is the 2*creal(out * conj(in))
 *
 * complex *out == pointer to the output/first input array
 * complex *in == pointer to the second input array
 * int m == number of elements in the arrays.
 */

void flip(complex *in, complex *out, int m);
/* ========================================================================
 * void flip(complex *in, complex *out, int m)
 *
 * flips the array over fft-style so f(x,y) => f(-x,-y) with the origin
 * still in the right place...
 */
```

Listing A.8: core_objective.c

```
/* Peter Johnson
 * 7 Nov 2006
 *
 * this file contains the functions used for evaluation of the
 * un-regularized focal-plane objective function and gradient.
 *
 * as of 8 Nov 2006 this has been verified completely, and appears to work
 * as expected.
 *
 * see core_objective.h for calling sequence and parameter details.
 */

#include "config.h"
#include "mylibs.h"
#include "core_objective.h"
#include "zernikes.h"

double objective(double *obj, double *aberrin, double div, double *datain,
                 double *G_obj, double *G_phsin, int f, int k, int m)
{
  int c0, c1, c2;                    // counters
  int mm=m*m;                        // number of pixels in each frame
  int *msk;                          // binary mask array

  size_t dbl = sizeof(double);       // some sizes for malloc and friends
  size_t cplx = sizeof(complex);

  extern double bias;                // bias away from zero
  double L=0;                        // objective function value
  double *divaberr;                  // storage for div aberr coefficients
  double defocus;                    // multiplier for z-mode 4 defocus
  double dtmp, gtmp;                 // scratch
  double *zmode;                     // zernike mode array
  double *data, *aberr;              // working pointers

  complex *g1, *g2, *psf1, *psf2;    // cmplx array ptrs
  complex *f1, *f2;                  // conv kernel for grads
  complex *H1, *H2, *h1, *h2;        // arrays for OTF's & impuls resp.
  complex scl1, scl2;                // psf scaling constants
  complex *otmp;                     // storage for fft(obj)
  complex *scratch;                  // complex scratch array

  // allocate storage arrays. All arrays that are fft'd need to be allocated
  // with fftw_malloc() and freed with fftw_free(), others can be allocated
  // with malloc() and freed with free()
  g1 = fftw_malloc(mm*cplx);          // model array
  g2 = fftw_malloc(mm*cplx);          // div model
  psf1 = fftw_malloc(mm*cplx);        // psf array
  psf2 = fftw_malloc(mm*cplx);        // div psf array
  H1 = fftw_malloc(mm*cplx);          // focal OTF
  H2 = fftw_malloc(mm*cplx);          // div OTF
  h1 = fftw_malloc(mm*cplx);          // impulse resp.
  h2 = fftw_malloc(mm*cplx);          // div impuls resp.
  f1 = fftw_malloc(mm*cplx);          // conv kernel for obj
  f2 = fftw_malloc(mm*cplx);          // conv kernel for G_obj
  scratch = fftw_malloc(mm*cplx);     // complex scratch
  otmp = fftw_malloc(mm*cplx);        // fft(obj)
  divaberr = malloc(k*dbl);           // div aberration
  zmode = malloc(mm*dbl);             // zernike mode
  msk = malloc(mm*sizeof(int));       // mask array

  // initialize variables and do frame-independent stuff
  L = 0;                              // initialize L to zero...
  defocus = div*M_PI/sqrt(3);         // amount of z4 to add to div chanel

```

```
// compute the fft of the object
rl_to_cmplx_copy(obj,g1,mm);
fft2(g1,otmp,m,m);

// cycle through the frames of data and accumulate the results...
for (c0=0; c0<f; c0++)
  {
    // get pointer to this frame's data, div, and aberr array
    data = datain + c0*2*mm;       // pointer to data frames
    aberr = aberrin + c0*k;        // pointer to frame aberration

    ///////////////////////////////////////////////////////////////////////
    // make the OTF's, asf's, psf's and the blurred images
    // The OTF's and asfs are stored for use in the gradient
    // calculation.  The psf's are stored for use in the gradient wrt
    // intensity calculations. .
    ///////////////////////////////////////////////////////////////////////

    //---------------------------------------------------------------------
    // start with channel 1
    mkscreen(H1, msk, m, aberr, k);               // make the first OTF
    ifft2(H1,h1,m,m);                             // make the first asf
    cmplx_fftshift(h1, m, m);                     // recenter
    for (c1=0; c1<mm; c1++)
      *(psf1+c1) = *(h1+c1) * conj(*(h1+c1));     // mod-square
    scl1 = 1/cmplx_sum(psf1,mm);                  // get scale const
    cmplx_sclr_mult(psf1,scl1,mm);                // scale to unit-sum
    cmplx_sclr_mult(H1,csqrt(scl1),mm);           // scale H1
    cmplx_sclr_mult(h1,csqrt(scl1),mm);           // scale h1

    //---------------------------------------------------------------------
    // now do channel 2
    rl_copy(aberr, divaberr, k);                  // copy aberr params
    *(divaberr+2) += defocus;                     // add defocus
    mkscreen(H2, msk, m, divaberr, k);            // make 2nd OTF
    ifft2(H2,h2,m,m);                             // make 2nd asf
    cmplx_fftshift(h2, m, m);                     // recenter
    for(c1=0; c1<mm; c1++)
      *(psf2+c1) = *(h2+c1) * conj(*(h2+c1));     // mod-square
    scl2 = 1/cmplx_sum(psf2,mm);                  // get scale const
    cmplx_sclr_mult(psf2,scl2,mm);                // scale to unit-sum
    cmplx_sclr_mult(H2,csqrt(scl2),mm);           // scale H2
    cmplx_sclr_mult(h2,csqrt(scl2),mm);           // scale h2


    ///////////////////////////////////////////////////////////////////////
    // make the model data by convolving obj with psf's
    ///////////////////////////////////////////////////////////////////////

    //---------------------------------------------------------------------
    // start with channel 1
    fft2(psf1, f1, m, m);
    cmplx_dotmultiply(f1, f1, otmp, mm);
    ifft2(f1, g1, m, m);
    cmplx_fftshift(g1, m, m);

    //---------------------------------------------------------------------
    // do channel 2
    fft2(psf2, f2, m, m);
    cmplx_dotmultiply(f2, f2, otmp, mm);
    ifft2(f2, g2, m, m);
    cmplx_fftshift(g2, m, m);

    ///////////////////////////////////////////////////////////////////////
    // compute L and f1, and f2 for the gradient
    ///////////////////////////////////////////////////////////////////////
    for (c1=0; c1<mm; c1++)
      {
        //-------------------------------------------------------------
        // do channel 1 contribs
        dtmp = *(data+c1) + bias;             // add  bias to the data
        gtmp = creal(*(g1+c1)) + bias;        // model + bias
        L += log(gtmp) + dtmp/gtmp;           // compute L contribution
        *(f1+c1) = (1 - dtmp/gtmp)/gtmp;      // store this for gradient

        //-------------------------------------------------------------
        // do channel 2 contribs (same as above)
        dtmp = *(data+mm+c1) + bias;          // add bias
        gtmp = creal(*(g2+c1))+bias;          // get model + bias
        L += log(gtmp) + dtmp/gtmp;           // compute L contrib
        *(f2+c1) = (1-dtmp/gtmp)/gtmp;        // get f2 for gradient
      }

    //////////////////////////////////////////////////////////////////
    // compute gradient wrt I, use g1 and g2 as complex scratch arrays.
    //////////////////////////////////////////////////////////////////

    //----------------------------------------------------------------
    // start with channel 1
    fft2(f1, g1, m, m);                             // fft f1
    flip(psf1, scratch, m);                         // flip the psf
    fft2(scratch, psf1, m, m);                      // fft the psf
    cmplx_dotmultiply(g1, psf1, g1, mm);            // multiply
    ifft2(g1, scratch, m, m);                       // ifft the result
    cmplx_fftshift(scratch, m, m);                  // recenter
    for (c1=0; c1<mm; c1++)
      *(G_obj+c1) += creal(*(scratch+c1));          // add up contributions

    //----------------------------------------------------------------
    // now do channel 2
    fft2(f2, g2, m, m);                             // fft f2
    flip(psf2, scratch, m);                         // flip the psf
    fft2(scratch, psf2, m, m);                      // fft the psf
    cmplx_dotmultiply(g2, psf2, g2, mm);            // multiply
    ifft2(g2, scratch, m, m);                       // ifft the result
    cmplx_fftshift(scratch, m, m);                  // recenter
    for (c1=0; c1<mm; c1++)
      *(G_obj+c1) += creal(*(scratch+c1));          // add up contributons


    //////////////////////////////////////////////////////////////////
    // calculate the gradient wrt the aberration parameters now...
    // use g1, g2, psf1, and psf2 for scratch arrays
    //////////////////////////////////////////////////////////////////


    for (c1=2; c1<k; c1++)   // cycle through aberration parameters
                             // start at 2 to skip tip/tilt
      {
        // make the zernike mode for the current derivative
        zernike_grid(zmode, msk, c1+2, m);   // add 2 to skip piston

        //-------------------------------------------------------------
        // do channel 1 contribs
        rl_cmplx_dotmultiply(psf1, zmode, H1, mm); // phi_n*H1
        cmplx_sclr_mult(psf1, I, mm);              // multiply by j

        // do the ifft and multiply by the impulse responses
        ifft2(psf1, g1, m, m);                     // ifft
        cmplx_fftshift(g1, m, m);                  // recenter
        conj_dotmultiply(g1, h1, mm);              // 2*Re(conj(h1)*...)
        // convolve with the object
        fft2(g1, psf1, m, m);                          // fft
        cmplx_dotmultiply(g1, psf1, otmp, mm);         // multiply
        ifft2(g1, psf1, m, m);                         // ifft
        cmplx_fftshift(psf1, m, m);                    // recenter

        // add the contributions
        for (c2=0; c2<mm; c2++)
          *(G_phsin+c0*k+c1) += *(f1+c2)*creal(*(psf1+c2));

        //-----------------------------------------------------------------
        // now do channel 2 contribs
        rl_cmplx_dotmultiply(psf2, zmode, H2, mm); // phi_n*H2
        cmplx_sclr_mult(psf2, I, mm);                  // multiply by j

        // do the ifft and multiply by the impulse responses
        ifft2(psf2, g2, m, m);                         // ifft
        cmplx_fftshift(g2, m, m);                      // recenter
        conj_dotmultiply(g2, h2, mm);                  // 2*Re(conj(h2)*...)

        // convolve with the object
        fft2(g2, psf2, m, m);                          // fft
        cmplx_dotmultiply(g2, psf2, otmp, mm);         // multiply
        ifft2(g2, psf2, m, m);                         // ifft
        cmplx_fftshift(psf2, m, m);                    // recenter

        // add the contributions
        for (c2=0; c2<mm; c2++)
          *(G_phsin+c0*k+c1) += *(f2+c2)*creal(*(psf2+c2));

      }
    }

  ////////////////////////////////////////////////////////////////////////////
  // free up the allocated arrays
  ////////////////////////////////////////////////////////////////////////////

  fftw_free(g1);
  fftw_free(g2);
  fftw_free(psf1);
  fftw_free(psf2);
  fftw_free(H1);
  fftw_free(H2);
  fftw_free(h1);
  fftw_free(h2);
  fftw_free(f1);
  fftw_free(f2);
  fftw_free(scratch);
  free(divaberr);
  free(zmode);
  free(msk);

  ////////////////////////////////////////////////////////////////////////////
  // return L. The gradients are passed out as arrays
  ////////////////////////////////////////////////////////////////////////////
  return L;
}

void mkscreen(complex *screen, int *msk, int m, double *aberr, int k)
{
  int c0, c1, mm=m*m;
  double *screentmp;

  screentmp = malloc(mm*sizeof(double));
  // make the phase screen
  for (c0=2; c0<k; c0++)                // start defocus to avoid & tip/tilt
    {

      zernike_grid(screentmp, msk, c0+2, m);      // make the circular mode
      rl_sclr_mult(screentmp, *(aberr+c0), mm);   // scale it
      if (c0==2)
        rl_to_cmplx_copy(screentmp, screen, mm); // preload 1 mode
      else
        for (c1=0; c1<mm; c1++)
          *(screen+c1) += *(screentmp+c1);
    }

  // turn it into a phasor...
  for (c0=0; c0<mm; c0++)
    {
      if (*(msk+c0) == 1)
        *(screen+c0) = cexp(I*creal(*(screen+c0)));
      else
        *(screen+c0) = 0;
    }

  // free up working memory
  free(screentmp);
}



void conj_dotmultiply(complex *out, complex *in, int m)
{
  int c0;
  for (c0=0; c0<m; c0++)
    *(out+c0) = 2*creal(*(out+c0) * conj(*(in+c0)));
}


void flip(complex *in, complex *out, int m)
{
  int x,y,u,v;

  // this fctn is for mapping f(x,y) => f(-x,-y) with the origin remaining
  // at x=0 and y=0 using circular shifts (fft-style)
  for (x=0; x<m; x++)
    {
      if (x==0)
        u=0;
      else
        u = m-x;
      for (y=0; y<m; y++)
        {
          if (y==0)
            v = 0;
          else
            v=m-y;
          *(out + u + m*v) = *(in + x + m*y);
        }
    }

  return;
}
```

Listing A.9: psd_reg.h

```
// #include "mylibs.h"

/* This file should contain the functions needed to compute the
```

```
 * intensity regularization based on the pupil-plane speckle data and
 * the current object autocorrelation data.
 */

double psd_reg(double *obj, double *datain, double *G, int m);
/* ==================================================================
 * double psd_reg(double *obj, double *data, double *G, int m, int f);
 *
 * master function for computing the speckle-psd regularization term
 * and associated gradients.  The function returns the value of the
 * objective function, and stores the gradients wrt to each object
 * pixel in the array pointed to by G.
 *
 * double *obj == pointer to m x m object array
 * double *data == pointer to m x m array of averaged p-p speckle psd's
 * double *G == pointer to m x m gradient array
 * int m == number of pixels along each dimension of the input array
 * int f == number of frames used to generate the data array
 */
```

Listing A.10: psd_reg.c

```
/* This file should contain the functions needed to compute the
 * intensity regularization based on the pupil-plane speckle data and
 * the current object autocorrelation data.  See psd_reg.h for calling
 * sequence etc...
 *
 * this has been verified agains finite differences for the gradient, and the
 * objective is correct as well.   Need the input data to be unit-mean
 */

#include "mylibs.h"
#include "config.h"


double psd_reg(double *obj, double *datain, double *G, int m)
{
  extern double bias;           // bias away from 1/0
  int c0;                       // counters
  int mm=m*m;                   // pixels in each frame
  double *ocor;                 // working arrays
  double L;                     // return objective function value
  double gtmp, dtmp;            // scratch double variables
  complex *ftmp, *otmp;         // complex working arrays
  complex *tmp1, *tmp2, *tmp3; // more complex working arrays
  double *data;                 // working array for data...

  // allocate temporary arrays
  ocor = malloc(mm*sizeof(double));
  tmp1 = fftw_malloc(mm*sizeof(complex));
  tmp2 = fftw_malloc(mm*sizeof(complex));
  tmp3 = fftw_malloc(mm*sizeof(complex));
  ftmp = fftw_malloc(mm*sizeof(complex));
  otmp = fftw_malloc(mm*sizeof(complex));
  data = malloc(mm*sizeof(double));

  /* compute the object autocorrelation */
  rl_autocorr(obj, ocor, m);                  // do the autocorrelation

  /* copy the data to a working array */
  rl_copy(datain,data,mm);

  /* scale the correlation and data to be a mean of 1 */
  gtmp = 0;
  for(c0=0; c0<mm; c0++)
    {
      *(ocor+c0) /= (double)mm;
      gtmp += *(data+c0);
    }
  for(c0=0; c0<mm; c0++)
    *(data+c0)/=(gtmp/(double)mm);


  /* compute the objective function contribution
   *
   * The objective function is derived by assuming spatially
   * independent negative exponential statistics on the psd, where the
   * parameter is determined by the scaled object autocorrelation.
   * The log-likelihood of the joint negative exponential distribution
   * is used as the likelihood function
   *
   * also compute a term (ftmp) that will be used for the gradient...
   */


  // compute the objective and the convolution function for the grad
  L = 0;
  for (c0=0; c0<mm; c0++)
    {
      gtmp = *(ocor+c0)+bias;              // get the model value
      dtmp = *(data+c0)+bias;              // get the data value
      L += log(gtmp) + dtmp/gtmp;          // compute contributon
      *(ftmp+c0) = (1 - dtmp/gtmp)/gtmp;   // term for use in gradient
    }

  // Now compute the gradients

  // do the required convolutions/correlations
  rl_to_cmplx_copy(obj, tmp1, mm);           // copy the obj to cmplx arry
  fft2(tmp1, otmp, m, m);                    // fft the object
  fft2(ftmp, tmp1, m, m);                    // fft ftmp

  cmplx_dotmultiply(tmp2, tmp1, otmp, mm);   // multiply for conv
  ifft2(tmp2, tmp3, m, m);                   // inverse fft it
  cmplx_fftshift(tmp3, m, m);                // fft-shift it

  for (c0=0; c0<mm; c0++)                    // conjugate for correlation
    *(tmp1+c0) = conj(*(tmp1+c0));
  cmplx_dotmultiply(tmp2, tmp1, otmp, mm);   // multiply for corr
  ifft2(tmp2, tmp1, m, m);                   // inverse fft it
  cmplx_fftshift(tmp1, m, m);                // fft-shift it

  // sum the contributions and spit out the result
  for (c0=0; c0<mm; c0++)
  *(G+c0) = (creal(*(tmp3+c0)) + creal(*(tmp1+c0)))/(double)mm;


  /* free up all the working arrays */
  free(ocor);
  fftw_free(tmp1);
  fftw_free(tmp2);
  fftw_free(tmp3);
  fftw_free(ftmp);
  fftw_free(otmp);
  free(data);

  /* return the objective function contribution */
  return L;
}
```

Listing A.11: phs_reg.h

```c
/* Peter Johnson
 * 19 Dec 2006
 *
 * This file contains the stuff required for phase-regularization of the
 * image reconstruction problem.
 *
 */
#include "config.h"
#include "mylibs.h"
#include "zernikes.h"
#include <gsl/gsl_linalg.h>
#include <math.h>

//extern int inverse_exists=0;  // logical for use later set it to zero here
//extern double *inverse;       // storage for the inverse covariance matrix

void calc_covar_inverse(double *inv, double Dr0, int k);
/* void calc_covar_inverse(double *inv, double Dr0, int k) compute the
 * inverse of the noll-covariance matrix for a given number of modes, k, and
 * a specified D/r0.  Allocate the array prior to calling this fctn.
 */

void make_inv_covar(double Dr0, int k);
/* allocate the inverse covariance matrix, compute it, and set the flag
 * saying that we allready computed it.  The matrix and flag are stored in
 * global variables *inverse and inverse_exists respectively
 */

void free_inv_covar();
/* free the inverse covariance matrix and unset the flag
 */


double noll_covariance(int n, int m);
/* C = noll_covariance(int n, int m);
 * compute the zernike-mode covariance between the n'th and m'th Noll-indexed
 * zernike modes and assuming Kolmogorov turbulence.  This can be used to
 * build the covariance matrix, which in turn can be used to generate phase
 * screens.  To get the true covariance, this needs to be multiplied by
 * (D/r0)^(5/3).
 *
 * inputs:
 * int n == noll-index of the first mode
 * int m == noll-index of the second mode
 *
 * output:
 * double C == covariance of the requested modes
 */

double covar_reg(double *aberr, double *grad, int k, int f, double Dr0);
/* L = covar_reg(double *aberr, int k, int f, double Dr0);
 *
 * compute the regularization term derived from the expected intermodal
 * covariance.
 *
 * inputs:
 * double *aberr == pointer to the aberration array
 * int k == number of aberrations per frame
 * int f == number of frames
 *
 * outputs:
 * L == regularization contribution
 * double *grad == pointer to the output gradient array
 */
```

Listing A.12: phs_reg.c

```c
/* Peter Johnson
 * 8 Mar 2006
 *
 * This file contains the stuff required for phase-regularization of the
 * image reconstruction problem. This should eventually encapsulate the
 * inter-modal and the temporal statistics.
 *
 */

#ifndef M_PI
#define M_PI        3.14159265358979323846   /* pi */
#endif
#include <math.h>
#include <stdlib.h>
#include "phs_reg.h"

//int inverse_exists=0;        // logical for use later
//double *inverse              // storage for the inverse covariance matrix

void calc_covar_inverse(double *inv, double Dr0, int k)
{

  int c0,c1;                  // counters
  int s;                      // signum for the LU decomp

  double *covar;              // pointer to covariance matrix
  double tmp;
  double Dr053 =  pow(Dr0,5.0/3.0);

  // allocate the covariance matrix
  covar = malloc(k*k*sizeof(double));

  // make gsl_style matricies out of the arrays
  gsl_matrix_view covarm = gsl_matrix_view_array(covar,k,k);
  gsl_matrix_view invm = gsl_matrix_view_array(inv,k,k);
  gsl_permutation *p = gsl_permutation_alloc(k);

  // fill in the covariance matrix
  for (c0=0; c0<k; c0++)
    {
      for (c1=0; c1<=c0; c1++)
        {
          // get the covariance
          // have to index starting at two to skip piston...
          tmp = Dr053*noll_covariance(c0+2,c1+2);
          // matrix is symmetric so don't overdo it...
          *(covar + c0 + k*c1) = tmp;
          if (c1!=c0)
            *(covar+c1 + k*c0) = tmp;
        }
    }

  // use gsl to compute the inverse.....
  // start by LU factoring the covariance, then compute the inverse
  gsl_linalg_LU_decomp(&covarm.matrix, p, &s);
  gsl_linalg_LU_invert(&covarm.matrix, p, &invm.matrix);

  gsl_permutation_free(p);  // free permutaiton matrix
  free(covar);              // free up covariance array
}

void make_inv_covar(double Dr0, int k)
{
  extern double *inverse;
  extern int inverse_exists;
  // allocate memory
  inverse = malloc(k*k*sizeof(double));
  inverse_exists=1;

  // compute the covariance matrix
  calc_covar_inverse(inverse, Dr0, k);

  return;
}

void free_inv_covar()
{
  extern double *inverse;
  extern int inverse_exists;
  free(inverse);
  inverse_exists = 0;
  return;
}

double noll_covariance(int i, int j)
{
  double C, A, B, D;     // working doubles
  int ni, mi, nj, mj;    // radial and angular orders for the modes
  int tmp, one;

  // compute the radial and angular mode numbers
  zernikemode(i, &ni, &mi);
  zernikemode(j, &nj, &mj);

  // see if we even need to bother
  if (((i-j)%2==0) && (mi==mj))
    {
      // get the exponent on the negative 1 and decide if -1^m is negative
      tmp = (ni + nj - 2*mi)/2;
      if (tmp%2 == 0)
        one = 1;
      else
        one = -1;

      A = 0.0072*one*sqrt((ni+1)*(nj+1))*pow(M_PI,(double)8/3);
      B = tgamma((double)14/3)*tgamma((ni+nj-(double)5/3)/2);
      D = tgamma((ni-nj+(double)17/3)/2)*tgamma((nj-ni+(double)17/3)/2) *
        tgamma((ni+nj+(double)23/3)/2);
      C = A*B/D;
    }
  else
    C = 0;
  return C;
}

double covar_reg(double *aberr, double *grad, int k, int f, double Dr0)
{
  // make sure the grad vector is zeros before passing it to this fctn

  int c0, c1, c2;               // counters
  extern int inverse_exists;    // logical 1 if the covariance inverse has
                                // already been computed
  double L=0;                   // objective value
  double *frmaber;              // pointer to frame aberration params
  double *frmgrad;              // pointer to frame aberration gradient
  extern double *inverse;       // pointer to covariance inverse array

  // make the inverse covariance matrix if we need it... make sure you free
  // it in the main program...
  if (!inverse_exists)
```

```
    make_inv_covar(Dr0,k);

  // loop over the number of frames
  for (c2=0; c2<f; c2++)
    {
      // get pointers
      frmaber = aberr + c2*k;      // point to this frame's aberr vector
      frmgrad = grad + c2*k;       // point to this frame's aberr gradient

      // compute alpha^T*inv*alpha for the regulation term
      // and -inv*alpha for the gradient...
      for (c0=2; c0<k; c0++)       // start at 2 to skip tip/tilt
        {
          for (c1=2; c1<k; c1++)  // start at 2 to skip tip/tilt
            {
              // do first vector multiply
              *(frmgrad+c0) += *(inverse+c1+k*c0) * *(frmaber+c1);
            }
          L += *(frmgrad+c0) * *(frmaber+c0);  // add objective contribs
          *(frmgrad+c0) *= 2.0;    // multiply by 2
        }

    }

  // return the objective function value...
  return L;
}
```

Listing A.13: `mylibs.h`

```
/* Header file containing all of the function prototypes for the routines
   that I have built in c.  Most of these are to implement MATLAB-like
   routines.  */

#include <stdlib.h>
#include <math.h>
#include <complex.h>
#include <fftw3.h>
#include <time.h>

/* define some constants */

#ifndef M_PI
#define M_PI         3.14159265358979323846   /* pi */
#endif

/* set up function prototypes */

void rl_copy(double *from, double *to, int n);
/* rl_copy(from, to, n) copies the array pointed to by from into the array
   pointed to by to.  Both arrays need to be real and contain n elements */

void rl_to_cmplx_copy(double *from, complex *to, int n);
/* rl_to_cmplx_copy(from, to, n) copies the array pointed to by from into
   the array pointed to by to.  Both arrays need to be of size n. From needs
   to be a pointer to a double, and to a pointer to a complex. */

void cmplx_to_cmplx_copy(complex *from, complex *to, int n);
/* cmplx_to_cmplx_copy(from, to, n) copies the array pointed to by from into
   the array pointed to by to.  Both arrays need to be of size n, and both
   of type complex. */

void cmplx_to_rl_copy(complex *from, double *to, int n);
/* cmplx_ro_rl_copy(from, to, n) copies the real part of the array pointed
   to by from into the array pointed to by to.  Both arrays need to be of
   size n. From needs to be a pointer to a complex, and to a pointer to a
   double. */

void cmplx_abs_copy(complex *from, double *to, int m);
/* cmplx_abs_copy(complex *from, double *to, int m); copies the absolute
   value of an m-element array from the complex array pointed to by *from to
   the double array pointed to by *to */

void abssq(complex *array, int m);
/* abssq(array, m, n) computes the magnitude squared of every element in the
   m-element array pointed to by array.  The array must be complex. */

void rl_sclr_mult(double *array, double c, int n);
/* rl_sclr_mult(array, c, n) multiplies every element of the n-element array
   by the scalar c. array must point to an array of type double  */

void cmplx_sclr_mult(complex *array, complex c, int n);
/* cmplx_sclr_mult(array, c, n) multiplies every element of the n-element
   array by the scalar c. array must point to an array of type complex  */

void rl_fftshift(double *array, int n, int m);
/* implement the fftshift for 2-D arrays to put the origin at the center of
   the array.  Only works for even-sized double arrays. */

void cmplx_fftshift(complex *array, int n, int m);
/*  implement the fftshift for 2-D arrays to put the origin at the center of
   the array.  Only works for even-sized complex arrays. */

void fft2(complex *in, complex *out, int m, int n);
/* wrapper for the fftw to make 2-d fft's easier.  If coupled with the
   ifft2() function below, this is a scaled transform pair, with all the
   scaling done in the inverse transform. For optimal performace, the arrays
   should be allocated with the fftw_malloc() function. */

void ifft2(complex *in, complex *out, int m, int n);
/* wrapper for the fftw to make inverse 2-d fft's easier.  When paired with
   fft2() above, this is a scaled transform pair with all of the scaling
   done in the ifft.For optimal performace, the arrays should be allocated
   with the fftw_malloc() function. */

void inplace_fft2(complex *in, int m, int n);
/* wrapper for in-place fftw 2-d transform.  When paired with
   inplace_ifft2() below, this is a scaled transform pair with all of the
   scaling done in the ifft. For optimal performace, the arrays should be
   allocated with the fftw_malloc() function. */

void inplace_ifft2(complex *in, int m, int n);
/* wrapper for in-place fftw inverse 2-d transform.  When paired with
   inolace_fft2() above, this is a scaled transform pair with all of the
   scaling done in the ifft. For optimal performace, the arrays should be
   allocated with the fftw_malloc() function. */

void dotmultiply(double *out, double *in1, double *in2, int m);
/* void dotmultiply(double *out, double *in1, double *in2, int m)
 *
 * out == pointer to the output array
 * in1 == pointer to the first input array
 * in2 == pointer to the second input array
 * n == integer number of elements in the array
 *
 * implements the equivalent of the .* operator in matlab such that
 * out(i) = in1(i).*in2(i)
 */

void cmplx_dotmultiply(complex *out, complex *in1, complex *in2, int m);
/* void dotmultiply(double *out, double *in1, double *in2, int m)
 *
 * out == pointer to the output array
 * in1 == pointer to the first input array
 * in2 == pointer to the second input array
 * n == integer number of elements in the array
 *
 * implements the equivalent of the .* operator in matlab such that
 * out(i) = in1(i).*in2(i)
 */

void rl_cmplx_dotmultiply(complex *out, double *in1, complex *in2, int
m);
/* void dotmultiply(complex *out, double *in1, complex *in2, int m)
 *
 * out == pointer to the output array
 * in1 == pointer to the first input array
 * in2 == pointer to the second input array
 * n == integer number of elements in the array
 *
 * implements the equivalent of the .* operator in matlab such that
 * out(i) = in1(i).*in2(i)
 */

double factorial(double n);
/* uses recursion to compute the factorial of an integer.  Will crap
   out for large integers... */

void rl_autocorr(double *in, double *out, int n);
/* ================================================================
 * void rl_autocorr(double *in, double *out, int n)
 *
 * This function computes the autocorrelation function of a
 * real-valued n x n array using the fft.  The function must be square
 * for this to work as advertised.
 *
 * double *in == pointer to input array
 * double *out == pointer to output array
 * int n == number of elements along each dimention of the input array
 */

void spec_psd(double *in, double *out, int n);
/* =================================================================
 * void spec_psd(double *array, double *rslt, int n)
 *
 * This function computes the non-normalized delta-removed speckle
 * psd. In theory this psd is related to the reflecting object's
 * autocorrelation through a simple scaling constant.
 *
 * double *in == pointer to the input array
 * double *out == pointer to the output array
 * int n == number of pixels along each dimention of the input array
 */

void rl_normalize(double *in, int m);
/* =================================================================
 * void rl_normalize(double *in, int m) normalizes a real-valued array
 * of m elements such that the maximum value of any element is 1.
 *
 * double *in == pointer to the first array element
 * int m == integer number of elements in the array
 */

double rl_max(double *in, int m);
/* =================================================================
 * double rl_max(double *in, int m) returns the value of the largest
 * element of the input array
 *
 * double *in == pointer to the input array
 * int m == integer number of elements in the array
 *
 * returns mx == maximum value in the array
 */

void cmplx_normalize(complex *in, int m);
/* =====================================================================
 * void cplx_normalize(complex *in, int m) normalizes a complex-valued array
 * of m elements such that the maximum absolute value of any element is 1.
 *
 * complex *in == pointer to the first array element
 * int m == integer number of elements in the array
 */

double cmplx_max(complex *in, int m);
/* =================================================================
 * double cplx_max(complex *in, int m) returns the largest absolute value
 * of the input array
 *
 * complex *in == pointer to the input array
 * int m == integer number of elements in the array
 *
 * returns mx == maximum absolute value in the array
 */

double rl_sum(double *in, int m);
/* ================================================================
 * double rl_sum(double *in, int m);
 *
 * computes the sum of the elements in the array pointed to by *in
 *
 * double *in == pointer to the array
 * in m == number of elements in the array
 *
 * returns S = sum(in);
 */

complex cmplx_sum(complex *in, int m);
/* ================================================================
 * double cmplx_sum(double *in, int m);
 *
 * computes the sum of the elements in the array pointed to by *in
 *
 * double *in == pointer to the array
 * in m == number of elements in the array
 *
 * returns S = sum(in);
 */

double unit_scale(double *in, int m);
/* ================================================================
 * double unit_scale(double *in, int pixels)
 *
 * scales the input array to have unit-average values and returns the
 * required scaling constant.
 *
 * double *in == pointer to the array
 * int m == number of elements in the array
 * double (return value) == multiplicative scaling constant used to
 *                          get unit-average intensity.
 */
```

Listing A.14: `mylibs.c`

```
#include "mylibs.h"
```

```c
/* functions to copy the contents of arrays */

void rl_copy(double *from, double *to, int m)
{
  int c0;

  for (c0=0; c0<m; c0++)
    *(to + c0) = *(from + c0);
  return;
}

void rl_to_cmplx_copy(double *from, complex *to, int m)
{
  int c0;
  complex tmp;

  for (c0=0; c0<m; c0++)
    {
      tmp = *(from+c0);
      *(to+c0) = tmp;
    }
  return;
}

void cmplx_to_cmplx_copy(complex *from, complex *to, int m)
{
  int c0;

  for (c0=0; c0<m; c0++)
    *(to + c0) = *(from + c0);
  return;
}
void cmplx_to_rl_copy(complex *from, double *to, int m)
{
  int c0;

  for (c0=0; c0<m; c0++)
    *(to + c0) = creal(*(from + c0));
  return;
}

void cmplx_abs_copy(complex *from, double *to, int m)
{
  int c0;
  for (c0=0; c0<m; c0++)
    *(to + c0) = cabs(*(from + c0));
  return;
}

/* function to compute the magnitude squared*/
void abssq(complex *array, int m)
{
  int c0;
  complex tmp;
  for (c0=0; c0<m; c0++)
    {
      tmp = *(array + c0);
      *(array + c0) = tmp * conj(tmp);
    }
  return;
}

/* scalar dot multiply functions for arrays */
```

```
void rl_sclr_mult(double *array, double c, int m)
{
  int c0;
  for (c0=0; c0<m; c0++)
    *(array + c0) = c * (*(array + c0));
  return;
}


void cmplx_sclr_mult(complex *array, complex c, int m)
{
  int c0;
  for (c0=0; c0<m; c0++)
     *(array + c0) = *(array+c0) * c;
   return;
}

/* routine to do the fft-shift (i.e. put dc at the center of the
    shift) */
void rl_fftshift(double *array, int m, int n)
{
  int c0, c1;
  double *tmp;

  tmp = malloc(m*n*sizeof(double));

  int x, y, xm,ym;
  x = m/2;
  y = m/2;
  for (c0=0; c0<m; c0++)
    {
      for(c1=0; c1<n; c1++)
        {
          xm = c0 - x; if (xm < 0) xm += m;
          ym = c1 - y; if (ym < 0) ym += n;

          *(tmp + c0 + m*c1) = *(array + xm + m*ym);
        }
    }
  rl_copy(tmp, array,m*n);

  free(tmp);
  return;
}
void cmplx_fftshift(complex *array, int n, int m)
{
  int c0, c1;
  complex *tmp;

  tmp = malloc(m*n*sizeof(complex));

  int x, y, xm,ym;
  x = m/2;
  y = m/2;
  for (c0=0; c0<m; c0++)
    {
      for(c1=0; c1<n; c1++)
        {
          xm = c0 - x; if (xm < 0) xm += m;
          ym = c1 - y; if (ym < 0) ym += n;

          *(tmp + c0 + m*c1) = *(array + xm + m*ym);
        }
    }
  cmplx_to_cmplx_copy(tmp, array,m*n);
  free(tmp);
  return;
}

void fft2(complex *in, complex *out, int m, int n)
{
  /* wrapper for the fftw to make 2-d fft's easier */
  /* for best performance make sure that the "in" and "out" arrays are
     allocated with the fftw_malloc() function. */

  fftw_plan pf;

  /* set up the fft routine */
  pf = fftw_plan_dft_2d(m,n,in,out,FFTW_FORWARD,FFTW_ESTIMATE);
  /* 19 Jan 2006... for some reason the fft routine called with FFTW_MEASURE
     gives inconsistent results...  doesnt' always work. FFTW_ESTIMATE seems
     to be OK*/

  fftw_execute(pf);  /* do the transform */
  //fftw_cleanup();
  fftw_destroy_plan(pf);
}

void ifft2(complex *in, complex *out, int m, int n)
{
  /* wrapper for the fftw to make inverse 2-d fft's easier */
  /* for best performance make sure that the "in" and "out" arrays are
     allocated with the fftw_malloc() function. This also implements the
     scaling required to make this a scaled transform pair when paired with
     fft2() above. */

  fftw_plan pf;

  /* set up the fft routine */
  pf = fftw_plan_dft_2d(m,n,in,out,FFTW_BACKWARD,FFTW_ESTIMATE);
  /* 19 Jan 2006... for some reason the fft routine called with FFTW_MEASURE
     gives inconsistent results...  doesnt' always work. FFTW_ESTIMATE seems
     to be OK*/

  fftw_execute(pf);  /* do the transform */
  cmplx_sclr_mult(out, 1/(double)(m*n), m*n);  /* fix the scaling. */
  //fftw_cleanup();  /* cleanup fftw */
  fftw_destroy_plan(pf);
}



void inplace_fft2(complex *ary, int m, int n)
{
  /* inplace ffts are slower than the others... */

  /* in-place 2-d fft, see fft2() above */
  fftw_plan pf;
  pf = fftw_plan_dft_2d(m,n,ary,ary,FFTW_FORWARD,FFTW_ESTIMATE);
  fftw_execute(pf);
  fftw_cleanup();
  fftw_destroy_plan(pf);

}

void inplace_ifft2(complex *ary, int m, int n)
{
  /* inplace ffts are slower than the others... */

  /* in-place 2-d inverse fft with normalization, see ifft2() above */
  fftw_plan pf;
  pf = fftw_plan_dft_2d(m,n,ary,ary,FFTW_BACKWARD,FFTW_ESTIMATE);
  fftw_execute(pf);
  cmplx_sclr_mult(ary,1/(double)(m*n), m*n); // fix scaling
  fftw_cleanup();
  fftw_destroy_plan(pf);


}

/* matlab like dot multiply */
void dotmultiply(double *out, double *in1, double *in2, int m)
{
  int c0;

  for (c0=0; c0<m; c0++)
     *(out+c0) = *(in1+c0) * (*(in2+c0));
}

void cmplx_dotmultiply(complex *out, complex *in1, complex *in2, int m)
{
  int c0;
  for (c0=0; c0<m; c0++)
    *(out+c0) = *(in1+c0) * (*(in2+c0));
}

void rl_cmplx_dotmultiply(complex *out, double *in1, complex *in2, int m)
{
  int c0;
  for (c0=0; c0<m; c0++)
    *(out+c0) = *(in1+c0) * (*(in2+c0));
}

double rl_sum(double *in, int m)
{
  int c0;
  double S=0;
  for (c0=0; c0<m; c0++)
    S += *(in+c0);
  return S;
}
complex cmplx_sum(complex *in, int m)
{
  int c0;
  complex S=0;
  for (c0=0; c0<m; c0++)
    S += *(in+c0);
  return S;
}

/* factorial function for ints */
double factorial(double n)
{
  if((int)n==0) return 1;
  else return (n*factorial(n-1));
}

/* scaled autocorrelation function for real data */
void rl_autocorr(double *in, double *out, int n)
{
  complex *tmp1, *tmp2;   // pointer to complex data type

  tmp1 = fftw_malloc(n*n*sizeof(complex));
  tmp2 = fftw_malloc(n*n*sizeof(complex));
  rl_to_cmplx_copy(in, tmp1, n*n);   //copy input to working array
  fft2(tmp1, tmp2, n, n);            // do the fft
  abssq(tmp2, n*n);                  // get mag squared
  ifft2(tmp2, tmp1, n, n);            // do inverse fft
  cmplx_to_rl_copy(tmp1, out, n*n);  // copy the real part to out
  rl_fftshift(out, n, n);             // fftshift it
  fftw_free(tmp1);                    // deallocate working arrays
  fftw_free(tmp2);
}

/* normalization function */
void rl_normalize(double *in, int m)
{
  int c0;
  double mx;

  mx = rl_max(in,m); // find the maximum
  for (c0=0; c0<m; c0++)
    *(in+c0) = *(in+c0)/mx;  // divide it out
}


double rl_max(double *in, int m)
{
  int c0;
  double mx;
  mx = *in;
  for (c0=1; c0<m; c0++)
    if (*(in+c0) > mx)
        mx = *(in+c0);
  return mx;
}

/* normalization function for compelx data */
void cmplx_normalize(complex *in, int m)
{
  int c0;
  double mx;

  mx = cmplx_max(in,m);      // find the maximum (magnitude)
  for (c0=0; c0<m; c0++)
    *(in+c0) = *(in+c0)/(complex)mx;  // divide it out
}

double cmplx_max(complex *in, int m)
{
  int c0;
  double mx, tmp=0;
  mx = cabs(*in);
  for (c0=1; c0<m; c0++)
    tmp = cabs(*(in+c0));
    if (tmp > mx)
        mx = tmp;
  return mx;
}

/* delta-removed speckle psd calculation function */
void spec_psd(double *in, double *rslt, int n)
{

  complex *tmp1, *tmp2;  // pointer to complex data type
  complex delta = n*n;

  tmp1 = fftw_malloc(n*n*sizeof(complex));  // allocate memory
  tmp2 = fftw_malloc(n*n*sizeof(complex));
  rl_to_cmplx_copy(in, tmp1, n*n);    // copy to the working array
  fft2(tmp1, tmp2, n, n);             // do the fft
  *tmp2 = *tmp2 - delta;              // remove the delta
  abssq(tmp2,n*n);                    // get the mag squared
  cmplx_to_rl_copy(tmp2,rslt,n*n);   // copy rslt to output array
  fftw_free(tmp1);                   // free memory up
  fftw_free(tmp2);
  rl_fftshift(rslt,n,n);             // fftshift the result

  return;
}
/* delta-removed speckle psd calculation for a circular aperture */
void circle_spec_psd(double *in, double *rslt, int n)
{
  complex *tmp1;
  complex *subarray;
  complex delta;
  int m, strt;
  int c0, c1;

  m = n/sqrt(2);   // dimension of the largest inscribed square
  strt = n/2-m/2;  // where the sub-array starts

  // allocate working arrays
  subarray = fftw_malloc(m*m*sizeof(complex));
  tmp1 = fftw_malloc(n*n*sizeof(complex));

  delta = m*m;                       // how big is the delta...
  for (c0=0; c0<m; c0++)             // copy out the sub-array w/in pupil
    for(c1=0; c1<m; c1++)
      *(subarray+c0+m*c1) = *(in + c0+strt + (c1+strt)*n);
  inplace_fft2(subarray, m, m);   // take the fft
  *subarray = *subarray - delta;  // take out the delta at zero freq
  abssq(subarray, m*m);           // get the absolute-value squared...
  cmplx_fftshift(subarray, m, m); // fft-shift it

  // now interpolate it back to the right size by fft-zeropad-ifft
  inplace_fft2(subarray, m, m);   // go back to fft-land
  cmplx_fftshift(subarray, m, m); // fft-shift
  for (c0=0; c0<n; c0++)          // zero-pad
    {
      for(c1=0; c1<n; c1++)
        {
          if(c0<strt || c0>=m+strt || c1<strt || c1>=m+strt)
            *(tmp1+c0+n*c1) = 0;
          else
            *(tmp1+c0+n*c1) = *(subarray + (c0-strt) + (c1-strt)*m);
        }
    }

  // copy the result out to the result array
  cmplx_abs_copy(tmp1, rslt, n*n);

  // free the working arrays
  fftw_free(subarray);
  fftw_free(tmp1);

}

// scale the input array to have unit-average values
double unit_scale(double *in, int pixels)
{
  double sum=0;
  int c0;

  for (c0=0; c0<pixels; c0++)
    sum += *(in+c0);
  sum = (double)pixels/sum;
  for (c0=0; c0<pixels; c0++)
    *(in+c0) = *(in+c0)*sum;
  return sum;
}
```

Listing A.15: zernikes.h

```
double zernike_val(double r, double theta, int P);
/*
   double val = zernike_val(double r, double theta, int, n, int m);
   evaluates Z_n^m(r,theta), the n,m order zernike polynomial at the radial
   point r, and the angular point theta.

   inputs:
   double r == radial position.   note: these are only defined for
                                        r<1, it will return a value,
                                        but it is not valid.
   double theta == angular position. 0<=theta<2pi
   int P == Noll-indexed zernike mode number

   returns:
   val == value of the (n, m) zernike mode at location (r, theta)
*/

void zernikemode(int P, int *n, int *m);
/* returnes the radial and frequency indices n and m for a given
   noll-indexed zernike mode P

   inputs:
   P == int noll-indexed mode number

   outputs:
   n == int ptr to radial zernike index
   m == int ptr to frequency zernike index */

void zernike_grid(double *arry, int *mask, int P, int m);
/* evaluate the P'th zernike mode across the m x m grid assuming the array
   extends from -1 to 1 in both directions, and mask the result outside of
   the unit-circle.

   inputs:
   array == double pointer to the allocated output array
   mask == int pointer to the mask array
   P == int noll index of the desired mode
   m == int number of pixles across the array
*/

void masked_zscreen(double *arry, double *mask, int P, int m);
/* evaluate the P'th zernike mode across the m x m grid assuming the array
   extends from -1 to 1 in both directions, and mask the result in
   accordance with the analog mask array *mask.  For the zernike screen to
   be valid, the mask must be zero for all pixels that lie completely
   outside the unit-circle.

   inputs:
   array == double pointer to the allocated output array
   mask == double pointer to the mask array
   P == int noll index of the desired mode
   m == int number of pixles across the array
 */

void zscreen(double *arry, int P, int m);
/* make a zernike mode screen that is valid across the entire grid.
   i.e. the unit-circle completely encompases the grid.

   this just like zernike_grid with the spatial limits such that -1/sqrt(2)
   <= (x,y) <= 1/sqrt(2) so that the entire grid fits inside the unit circle
```

```
    inputs:
    array == double pointer to the allocated output array
    P == int noll index of the desired mode
    m == int number of pixles across the array

*/
```

Listing A.16: `zernikes.c`

```
/* this file is for development and testing of zernike-mode generation
    code... */

#include "mylibs.h"
#include "zernikes.h"

/* see zernikes.h for details about calling these functions */


/********************************************************************/


double zernike_val(double r, double theta, int P)
{
  int n, m, s;
  int odd;
  double R, num, den, scl;
  double tmp1, tmp2;

  /* use hard-coded zerikes for p<45, otherwise compute with the sum */
  switch (P)
    {
    case 1: R = 1; break;
    case 2: R = 2*r*cos(theta); break;
    case 3: R =  2*r*sin(theta); break;
    case 4: R =  sqrt(3)*(2*r*r-1); break;
    case 5: R =  sqrt(6)*r*r*sin(2*theta); break;
    case 6: R =  sqrt(6)*r*r*cos(2*theta); break;
    case 7: R =  sqrt(8)*(3*r*r*r - 2*r)*sin(theta); break;
    case 8: R =  sqrt(8)*(3*r*r*r - 2*r)*cos(theta); break;
    case 9: R =  sqrt(8)*r*r*r*sin(3*theta); break;
    case 10: R =  sqrt(8)*r*r*r*cos(3*theta); break;
    case 11: R =  sqrt(5)*(6*r*r*r*r - 6*r*r +1); break;
    case 12: R =  sqrt(10)*(4*r*r*r*r - 3*r*r)*cos(2*theta); break;
    case 13: R =  sqrt(10)*(4*r*r*r*r - 3*r*r)*sin(2*theta); break;
    case 14: R =  sqrt(10)*r*r*r*r*cos(4*theta); break;
    case 15: R =  sqrt(10)*r*r*r*r*sin(4*theta); break;
    case 16: R =  sqrt(12)*(10*r*r*r*r*r-12*r*r*r+3*r)*cos(theta); break;
    case 17: R =  sqrt(12)*(10*r*r*r*r*r-12*r*r*r+3*r)*sin(theta); break;
    case 18: R =  sqrt(12)*(5*r*r*r*r*r - 4*r*r*r)*cos(3*theta); break;
    case 19: R =  sqrt(12)*(5*r*r*r*r*r - 4*r*r*r)*sin(3*theta); break;
    case 20: R =  sqrt(12)*r*r*r*r*r*cos(5*theta); break;
    case 21: R =  sqrt(12)*r*r*r*r*r*sin(5*theta); break;
    case 22: R =  sqrt(7)*(20*r*r*r*r*r*r-30*r*r*r*r+12*r*r-1); break;
    case 23: R =  sqrt(14)*(15*r*r*r*r*r*r - 20*r*r*r*r +
                            6*r*r)*sin(2*theta); break;
    case 24: R =  sqrt(14)*(15*r*r*r*r*r*r - 20*r*r*r*r +
                            6*r*r)*cos(2*theta); break;
    case 25: R =  sqrt(14)*(6*r*r*r*r*r*r - 5*r*r*r*r)*sin(4*theta); break;
    case 26: R =  sqrt(14)*(6*r*r*r*r*r*r - 5*r*r*r*r)*cos(4*theta); break;
    case 27: R =  sqrt(14)*r*r*r*r*r*r*sin(6*theta); break;
    case 28: R =  sqrt(14)*r*r*r*r*r*r*cos(6*theta); break;
    case 29: R =  4*(35*r*r*r*r*r*r*r - 60*r*r*r*r*r + 30*r*r*r -
                   4*r)*sin(theta); break;
    case 30: R =  4*(35*r*r*r*r*r*r*r - 60*r*r*r*r*r + 30*r*r*r -
                   4*r)*cos(theta); break;
    case 31: R =  4*(21*r*r*r*r*r*r*r - 30*r*r*r*r*r +
                       10*r*r*r)*sin(3*theta); break;
    case 32: R =  4*(21*r*r*r*r*r*r*r - 30*r*r*r*r*r +
                       10*r*r*r)*cos(3*theta); break;
    case 33: R =  4*(7*r*r*r*r*r*r*r - 6*r*r*r*r*r)*sin(5*theta);
    case 34: R =  4*(7*r*r*r*r*r*r*r - 6*r*r*r*r*r)*cos(5*theta);
    case 35: R =  4*r*r*r*r*r*r*r*sin(7*theta); break;
    case 36: R =  4*r*r*r*r*r*r*r*cos(7*theta); break;
    case 37: R =  3*(70*r*r*r*r*r*r*r*r - 140*r*r*r*r*r*r +
                       90*r*r*r*r - 20*r*r + 1); break;
    case 38: R =  sqrt(18)*(56*r*r*r*r*r*r*r*r - 105*r*r*r*r*r*r +
                             60*r*r*r*r - 10*r*r)*cos(2*theta); break;
    case 39: R =  sqrt(18)*(56*r*r*r*r*r*r*r*r - 105*r*r*r*r*r*r +
                             60*r*r*r*r - 10*r*r)*sin(2*theta); break;
    case 40: R =  sqrt(18)*(28*r*r*r*r*r*r*r*r - 42*r*r*r*r*r*r +
                             15*r*r*r*r)*cos(4*theta); break;
    case 41: R =  sqrt(18)*(28*r*r*r*r*r*r*r*r - 42*r*r*r*r*r*r +
                             15*r*r*r*r)*sin(4*theta); break;
    case 42: R =  sqrt(18)*(8*r*r*r*r*r*r*r*r -
                             7*r*r*r*r*r*r)*cos(6*theta); break;
    case 43: R =  sqrt(18)*(8*r*r*r*r*r*r*r*r -
                             7*r*r*r*r*r*r)*sin(6*theta); break;
    case 44: R =  sqrt(18)*(r*r*r*r*r*r*r*r)*cos(8*theta); break;
    case 45: R =  sqrt(18)*(r*r*r*r*r*r*r*r)*sin(8*theta); break;
    default:
      /* get the zernike indices and figure out if p is odd*/
      zernikemode(P, &n, &m);
      if (P%2==0)
        odd=0;
      else odd=1;

      /* use the equations from Noll to compute the zernike val */
      R = 0;
      for (s = 0; s<=(n-m)/2; s++)
        {
          if (s%2==0)  // even power...
            scl = 1;
          else  // odd power
            scl = -1;
          num = scl * factorial(n-s);// * pow(r,(n-(2*c0)));
          den = factorial(s) * factorial((double)(n-m)/2 - s) *
                               factorial((double)(n+m)/2 - s);
          R += (num/den)*pow(r,(n-(2*s)));
        }

      if (m==0)
        R *= sqrt(n+1);
      else if (odd == 1)
        R *= sqrt(2*(n+1))*sin(m*theta);
      else
        R *= sqrt(2*(n+1))*cos(m*theta);


    }
  return R;
}

void zernikemode(int P, int *n, int *m)
{
  int cntr=1, reuse=0;
  int n0=0, m0=0;
  while (cntr <P)
    {
      n0 +=1;
      if ((n0 - 2*(n0/2)) == 0)  /* if n is even start at m=0 */
        {
          m0 = 0;
          reuse = 0;
          cntr++;
          if (cntr >=P)
            { *n = n0; *m = m0; return;}
          while (m0<n0 || reuse==1)
            {
              if (reuse==1)
                {
                  reuse = 0;
                  cntr++;
                  if (cntr>=P)
                  { *n = n0; *m = m0; return;}
                }
              else
                {
                  reuse = 1;
                  m0 +=2;
                  cntr++;
                  if (cntr>=P)
                     { *n = n0; *m = m0; return;}
                }
            }
        }
      else    /* n is odd... */
        {
          reuse = 1;
          m0 = 1;
          cntr++;
          if (cntr >=P)
            { *n = n0; *m = m0; return;}
          while (m0<n0 || reuse ==1)
            {
              if (reuse ==1)
                {
                  reuse = 0;
                  cntr++;
                  if (cntr>=P)
                    { *n = n0; *m = m0; return;}
                }
              else
                {
                  reuse = 1;
                  m0+=2;
                  cntr++;
                  if (cntr>=P)
                    { *n = n0; *m = m0; return;}
                }
            }
        }
    }
  *n = n0;
  *m = m0;
}

void zernike_grid(double *arry, int *mask, int P, int m)
{
  int c0,c1;   // counters
  double dx;   // point spacing along the grid
  double x,y,r,theta=0;   // coordinate variables
  double pi = M_PI;       // shorthand for pi
  double *tmp = arry;     // output array
  int *tmp2=mask;         // mask array

  dx = (double)2/(m-1);   // dx

  x = -1;
  y = -1;
  for (c0=0; c0<m; c0++)
    {
      for (c1=0; c1<m; c1++)
        {
          r = sqrt(x*x + y*y);  // get the radius
          if (r>1)
            {
              *tmp = 0;  // zero outside unit-circle
              *tmp2 = 0; // make the mask
            }
          else
            {
              *tmp2 = 1;  // make the mask
              // figure out what quadrant we are in
              if (x == 0) // avoid divide by zero
                {
                  if (y>=0)
                    theta = pi/2;
                  else if (y<0)
                    theta = 3*pi/2;
                }
              else
                if (x>0)  // take care of 1st & 4th quadrants
                  theta = atan(y/x);
                else            // second and third quadrant
                  theta = pi + atan(y/x);
              *tmp = zernike_val(r, theta, P);  // value inside circle
            }
          x += dx;  // increment x
          tmp++;    // increment grid pointer
          tmp2++;   // increment mask pointer
        }
      x = -1;       // reset x
      y+=dx;        // increment y
    }
}

void masked_zscreen(double *arry, double *mask, int P, int m)
{
  int c0,c1;   // counters
  double dx;   // point spacing along the grid
  double x,y,r,theta=0;   // coordinate variables
  double pi = M_PI;       // shorthand for pi
  double *tmp = arry;     // output array
  double *tmp2=mask;          // mask array

  dx = (double)2/(m-1);   // dx

  x = -1;
  y = -1;
  for (c0=0; c0<m; c0++)
    {
      for (c1=0; c1<m; c1++)
        {
          r = sqrt(x*x + y*y);  // get the radius
          if (*tmp2 == 0)
            *tmp = 0;  // zero outside unit-circle
          else
            {
              // figure out what quadrant we are in
              if (x == 0) // avoid divide by zero
                {
                  if (y>=0)
                    theta = pi/2;
                  else if (y<0)
                    theta = 3*pi/2;
                }
              else
                if (x>0)  // take care of 1st & 4th quadrants
                  theta = atan(y/x);
                else             // second and third quadrant
                  theta = pi + atan(y/x);
              *tmp = *tmp2*zernike_val(r, theta, P);  // value inside circle
            }
          x += dx;  // increment x
          tmp++;    // increment grid pointer
          tmp2++;   // increment mask pointer
        }
      x = -1;       // reset x
      y+=dx;        // increment y
    }
}

void zscreen(double *arry, int P, int m)
{
  int c0,c1;   // counters
  double dx, xmax;   // point spacing along the grid
  double x,y,r,theta=0;   // coordinate variables
  double pi = M_PI;       // shorthand for pi
  double *tmp = arry;


  xmax = 2.0/sqrt(2.0); // twice the max x or y value
  if ((m-2*(m/2))==0)   // for an even sized grid
    dx = xmax/m;  // make sure we hit zero at the right spot
  else                  // for an odd-sized grid
    dx = xmax/(m-1);

  x = -1/sqrt(2);
  y = x;
  for (c0=0; c0<m; c0++)
    {
      for (c1=0; c1<m; c1++)
        {
          r = sqrt(x*x + y*y);
          // figure out where we are first
          if (x == 0) // avoid divide by zero
            {
              if (y>=0)
                theta = pi/2;
              else if (y<0)
                theta = 3*pi/2;
            }
          else if (x>0)  // talke care of 1st & 4th quadrants
            theta = atan(y/x);
          else             // second and third quadrant
            theta = pi + atan(y/x);

          *tmp = zernike_val(r, theta, P);
          x += dx;
          tmp++;
        }
      x = -1/sqrt(2);
      y+=dx;
    }
}
```

This appendix contains the code sections used to generate simulated data, run the APDI algorithm, and post-process the results. `Matlab`® code and associated mex files were compiled using `Matlab`® R14SP3, Gnu `make` and `gcc 3.4.6`. The FFTW3 library is required to build several of the `mex` files.

B.1 Simulation Code

Simulated data was generated using `Matlab`® . The uniform and normal random number generators were seeded using random bits read from the Linux /dev/urandom device file, or alternatively with the system time.

Listing B.1: `writePDdata.m`

```
function [] = writePDdata(snr,frms, det_sigma,varargin)
%==========================================================================
% Peter Johnson
% 10 Jul 2006
%
% writePDdata(snr,frms,det_sigma) generates simulated active phase-diversity
% data and a starting guess for the APD reconstruction algorithm.
%
% snr is the overall detection SNR encompasing poton and gaussian
% read noise, frms is the number of indpendent noise realizations to
% generate, det_sigma is the number of rms noise electrons/read for
% the gaussian read noise.
%
% writePDdata(snr,frms,det_sigma,'objfl.mat') will generate data using the
% object contained in objfl.mat
%==========================================================================

% compute the average number of photo-electrons required to get the desired
% snr. This comes from inverting snr = k/sqrt(k+sigma^2)
photons = max(roots([-1/snr^2 1 det_sigma^2]));

% randomize the random number generators...
% inlcude system time in case /dev/urandom is depleted of entropy
Rfile = fopen('/dev/urandom','r');          % pull from the linux random
r = fread(Rfile,2,'uint32');                % device for seeds. will only
fclose(Rfile);                              % work with linux
rand('state',sqrt(r(1)*sum(100*clock)));    % seed uniform-rand
randn('state',sqrt(r(2)*sum(100*clock)));   % seed normal-rand

% load the test object
if nargin>3
  sat = char(varargin(1));      % see if we were passed a truth object
else
  sat = 'medsat.mat';           % if not, load the default object
end
load(sat);
s = size(obj);
obj = obj/mean(mean(obj));

% set up some constants
modes = 15;       % number of zernike modes to compensate for
calcmodes = 200; % number of modes to blur data with
```

```
D_r0 = 15;         % D/r0 for the phase screen
div = 2;           % add waves of defocus to the div channel

% get indexes/make arrays for clear parts of aperture for the pp-psd
m = floor(s(1)/sqrt(2));
strt = fix(s(1)/2-m/2);
ind = strt+1:strt+m+1;
tmp1 = zeros(s);                 % working array
tmp2 = zeros(m);                 % temp array for clear part of pupil
tmp3 = zeros(s);                 % working array

% make working arrays
dta = zeros([s,2*frms]);         % storage array for the data...
wghts = zeros(1,modes*frms);     % storage for aberration parameters
calcwghts = zeros(1,calcmodes); % storage for aberration parameters
psd = zeros(s);                  % storage for averaged speckle psd

% make frames of data and put them in the output array
for c0=1:frms

  % make the focal and diversity phase screen
  inds = [1:modes]+(c0-1)*modes;        % index to aberr parameters
  % make the FP phase screen
  [scrn,msk,calcwghts] = circle_phasescreen(D_r0, s(1), calcmodes);
  wghts(inds) = calcwghts(1:modes);     % save the aberr parameters
  divwghts = calcwghts;                 % copy aberr for div screen
  divwghts(3) = divwghts(3) + div*pi./sqrt(3);  % add defocus
  % build up the DP phase screen
  scrn2 = zeros(s);
  for c1=3:length(divwghts)
    [tmp, msk] = zernike_grid(c1+1,s(1));
    scrn2 = scrn2 + divwghts(c1)*tmp;
  end

  % get the pupil-plane field
  fld = fieldsim(obj);

  % get the PP intensity, add noise, throw out negatives and normalize
  I_pp = abssq(fld(ind,ind));       % get the field inside the aperture
  if snr<100
    I_pp = poissrnd(I_pp*photons);                    % photon noise
    I_pp = I_pp + det_sigma*randn(size(I_pp));        % read noise
    I_pp = I_pp/photons;                              % normalize
    I_pp(find(I_pp<0)) = 0;                           % kill negative
  end

  % compute and average the pupil-plane speckle psd
  tmp2 = fftshift(fft2(spec_psd(I_pp)));
  tmp3(ind,ind) = tmp2;
  psd = psd + real(ifft2(ifftshift(tmp3)));

  % make the in-focus data frames, add noise and normalize
  I_fp = prod(s)*abssq(ifft2(msk.*fld.*exp(j*scrn)));
  if snr<100
    I_fp = poissrnd(I_fp*photons);                    % photon noise
    I_fp = I_fp + det_sigma*randn(size(I_fp));        % read noise
    I_fp = I_fp/photons;                              % normalize
    I_fp(find(I_fp<0)) = 0;                           % kill negative
  end
  dta(:,:,2*c0-1) = I_fp;

  % make the diversity data frames, add noise and normailze
  I_dp = prod(s)*abssq(ifft2(msk.*fld.*exp(j*scrn2)));
  if snr<100
    I_dp = poissrnd(I_dp*photons);                    % photon noise
    I_dp = I_dp + det_sigma*randn(size(I_dp));        % read noise
    I_dp = I_dp/photons;                               % normalize
    I_dp(find(I_dp<0)) = 0;                            % kill negative
  end
  dta(:,:,2*c0) = I_dp;
end

% make an initial guess array...
guess_obj = .05*ones(s);
guess_aber = 0.5*randn(1,length(wghts));
for c0=0:frms-1
  guess_aber(modes*c0+1:modes*c0+2) = 0;
end

% open the output files
guess = fopen('guess.dat','w','native');
data = fopen('data.dat','w','native');
truth = fopen('truth.dat','w','native');

% The data.dat file contains the data in the follwing configuration the first
% element is m, the dimension of the array, the second is f, the number of
% frames of data, the third is the number of aberration parameters, k, used
% to generate the phase  screen, the fourth is the number of waves of defocus
% used to generate the diversity images.  The remainder of the file contains
% the data values in the following format...  The contiguous block of f*m^2
% values are the focal and diversity intensities, and the final m^2 values
% are the averaged delta-removed and unit-average normalized pupil-plane
% speckle psd.
%
% The contiguous block of focal and diversity plane data are broken into 2m^2
% blocks, each block containing the data from one realization/frame.  The
% first m^2 values are the focal-plane data for this frame and the second m^2
% values are the diversity plane data.
%
% The guess and truth data files are identical with the exception that the
% truth data file contains the true object and aberration, and the guess file
% contains the initial guess. The first element in the guess file is m, the
% dimension of the object, the second element is the number of frames f, the
% third the number of aberration parameters k, the next m^2 elements are the
% object data values, and the last f*k elements are the aberration parameters.

% write the initial guess file...
m = s(1);
fwrite(guess, m, 'double');
fwrite(guess, frms, 'double');
fwrite(guess, modes, 'double');
fwrite(guess, D_r0, 'double');
fwrite(guess, div, 'double');
cnt = fwrite(guess, guess_obj, 'double');
cnt = fwrite(guess, guess_aber, 'double');

% write the truth data file...
fwrite(truth, m, 'double');
fwrite(truth, frms, 'double');
fwrite(truth, modes, 'double');
fwrite(truth, D_r0, 'double');
fwrite(truth, div, 'double');
cnt = fwrite(truth, obj, 'double');
cnt = fwrite(truth, wghts, 'double');

% write the data file...
fwrite(data, m, 'double');
fwrite(data, frms, 'double');
fwrite(data, modes, 'double');
fwrite(data, D_r0, 'double');
fwrite(data, div, 'double');
cnt = fwrite(data, dta, 'double');
cnt = cnt+fwrite(data, psd, 'double');

% output status and close the files
fprintf('\nwrote %d %dx%d frames using %d modes\n',frms,m,m,modes);
fclose('all');
```

Listing B.2: `circle_phasescreen.m`

```
function [screen, mask, X]=circle_phasescreen(D_r0,n,k)
%============================================================================
% Peter Johnson
% 7 Mar 2006
%
% [screen, mask, X] = circle_phasescreen(D_r0,n,k)
%
% Generate an nxn tilt-removed circularly-obscured phase screen for a D/r_0
% ratio of D_r0 using the first k Zernike polynomials (the first 3 are
% actually ingored, but counted anyway...)  and the covariance (Noll) matrix
% for the Zernike coefficients
%
% the weights used for the zernike modes are returned in the vector X.  The
% first two elements are ignored as tip and tilt.  Piston is not calculated
% nor carried along as a term...
%============================================================================

% covariance matrix for the zerike coefficients
K = D_r0^(5/3)*nollmatrix(k);   % make the noll-matrix for modes 2 => k+1
K = chol(K)';                   % Choleky Factorize K
X = randn(k,1);                 % Generate gaussian random numbers
X = K*X;                        % "color" them with K
X(1:2) = 0;                     % throw out tip/tilt

% now buld up the phase-screen and mask
screen = zeros(n);
for c0=1:k
    if (X(c0)~=0)
      [tmp,mask] = zernike_grid(c0+1,n);
       screen = screen + X(c0)*tmp;
    end
end

return
```

Listing B.3: `fieldsim.m`

```
function obj = fieldsim(obj);
% Peter Johnson
% 16 Nov 2005
% obj = fieldsim(obj); Uses AFRL gaussspec style speckle simulation to
% generaterate the complex fields in the pupil plane assuming an optically
% rough object.  The object array must be square for the scaling to be
% correct.  The resulting field should have unit-average intensity.
%
% It uses the object to produce circular complex RV's with statistics
% consistent with the object, then propagates the RV's to the far-field to
% generate the pupil-plane complex field.  A speckle realization is found
% by taking the square modulus of the field.  This implementation is set to
% give a unit-average intenisy in the pupil plane.
%
% This is faster, and agrees better with theory than the original upsample
% - add phase - propagate - down select method I used before. Definitely the
% better option.

obj = obj/mean(mean(obj));  % scale to unit average intensity
sigma = sqrt(obj/2);        % standard deviation for RV's
s = size(sigma);            % size of the object array
```

```
rl = sigma.*randn(s);        % make circular complex gaussian RV's
im = sigma.*randn(s);
obj = rl + i*im;

obj = fftshift(fft2(obj));   % propagate to far-field

obj = obj./s(1);     % re-scale it to unit average intensity at this point
                     % this is a result of MATLAB's implementation of FFT's
return
```

Listing B.4: `Makefile` for `Matlab`® MEX files

```
# set up variables
HOME=~pjohnson
CC=gcc
MATLAB=/apps/Linux86/matlab14sp3
MEX=$(MATLAB)/bin/mex
MATLABINC=-I$(MATLAB)/extern/include
MATLIBS=$(MATLAB)/extern/lib
CCFLAGS=-O3 -march=i686 -fPIC -Wall
LIBPATH=
SEARCHPATH=-I$(HOME)/lib/ -I$(HOME)/lib/headers
INCLUDE=-lfftw3 -lm
MATLIB=./matlab
MYLIB=mylibs.o

# the mex files that can be built/cleaned/etc...
MEXFLS=spec_psd.mexglx zernike_grid.mexglx nollmatrix.mexglx\
        circle_spec_psd.mexglx
MXOBJS=psdmex.o zernike_gridmex.o nollmatrixmex.o circle_spec_psdmex.o
OBJECTS=zernikes.o mylibs.o

# options for the different kinds of files
MEXOPTS=$(LIBPATH) $(SEARCHPATH) $(INCLUDE) -output
OOPTS=-c $(SEARCHPATH) $(CCFLAGS) $(MATLABINC)

# rule to make all the mex files
.PHONY: mex
mex: $(MEXFLS)
        mv *.mexglx ./matlab/

# rule to make all object files
.PHONY: objs
objs: $(OBJECTS)

# stuff for making mylibs.o
mylibs.o: mylibs.c
        $(CC) -c mylibs.c -o $(MYLIB) $(OOPTS)

# stuff for making a circular-aperture zernike mode
ZGRIDOB=zernikes.o $(MYLIB) zernike_gridmex.o
zernike_grid.mexglx: $(ZGRIDOB)
        $(MEX) -cxx $(ZGRIDOB) $(MEXOPTS) zernike_grid
zernike_gridmex.o: $(MATLIB)/zernike_gridmex.c zernikes.o
        $(CC) $(MATLIB)/zernike_gridmex.c $(OOPTS)

# stuff for making the psd functions
PSDOBJ=psdmex.o mylibs.o
spec_psd.mexglx: $(PSDOBJ)
        $(MEX) -cxx $(PSDOBJ) $(MEXOPTS) spec_psd
psdmex.o: $(MATLIB)/psdmex.c
        $(CC) $(OOPTS) $(MATLIB)/psdmex.c

# stuff for making the circle_spec_psd mexfunction
CRCPSDOBJ=circle_spec_psdmex.o mylibs.o
```

```
circle_spec_psd.mexglx: $(CRCPSDOBJ)
        $(MEX) -cxx $(CRCPSDOBJ) $(MEXOPTS) circle_spec_psd
circle_spec_psdmex.o: $(MATLIB)/circle_spec_psdmex.c
        $(CC) $(OOPTS) $(MATLIB)/circle_spec_psdmex.c

# stuff for making the noll-covariance matrix
NOLLOBJ=nollmatrixmex.o zernikes.o mylibs.o
nollmatrix.mexglx: nollmatrixmex.o
        $(MEX) -cxx $(NOLLOBJ) $(MEXOPTS) nollmatrix
nollmatrixmex.o: $(MATLIB)/nollmatrixmex.c
        $(CC) $(OOPTS) -std=c99 $(MATLIB)/nollmatrixmex.c

# clean up stuff
.PHONY : clean
clean:
        rm $(OBJECTS) $(MEXOBJS)
        cd matlab; rm $(MEXFLS)
```

Listing B.5: zernike_gridmex.c

```
/*====================================================================
 * Wrapper function for matlab to run subroutines written in C.      */

#include "mex.h"
#define LOCAL
#include "mylibs.h"
#include "zernikes.h"

void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
  double *arry, *mask, *m, *P;
  int mi, p, *msk, c0;

  /* check for complex input field */
  if (mxIsComplex(prhs[0]))
    mexErrMsgTxt("Input must be real");

  /* error check the number of input and output arguments */
  if (nrhs != 2)
    mexErrMsgTxt("need 2 input args");
  if (nlhs != 2)
    mexErrMsgTxt("Must have 2 output args");

  /* get a pointer to the input arrays */
  P = mxGetPr(prhs[0]);
  m = mxGetPr(prhs[1]);
  mi = *m;
  p = *P;

  /* make the mask array */
  msk = malloc(mi*mi*sizeof(int));

  /* make the output arrays to go back to matlab and set the pointer */
  plhs[0] = mxCreateDoubleMatrix(mi,mi,mxREAL);
  plhs[1] = mxCreateDoubleMatrix(mi,mi,mxREAL);
  arry = mxGetPr(plhs[0]);
  mask = mxGetPr(plhs[1]);

  /* call the subroutine */
  zernike_grid(arry, msk, p, mi);

  /* copy the mask to the matlab array */
  for (c0=0; c0<mi*mi; c0++)
    *(mask+c0) = *(msk+c0);

  free(msk);
  return;

}
```

Listing B.6: psdmex.c

```
/*=====================================================================
 * Wrapper function for matlab to run subroutines written in C .     */
#define LOCAL
#include "mylibs.h"
#include "mex.h"
void psdtest(double *array, double *rslt, int n);

void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
  int    n,m;
  double *speckle, *psd;

  /* check for complex input field */
  if (mxIsComplex(prhs[0]))
    mexErrMsgTxt("Input speckle field must be real");

  /* error check the number of input and output arguments */
  if (nrhs != 1)
    mexErrMsgTxt("Only one input is allowed");
  if (nlhs !=1)
    mexErrMsgTxt("Must have one output argument");

  /* get the dimension of the field */
  n = mxGetN(prhs[0]);
  m = mxGetM(prhs[0]);
  if (n != m)
    mexErrMsgTxt("Input must be square");

  /* get a pointer to the input array */
  speckle = mxGetPr(prhs[0]);

  /* make the output array to go back to matlab and set the pointer */
  plhs[0] = mxCreateDoubleMatrix(m,n,mxREAL);
  psd = mxGetPr(plhs[0]);

  /* call the psd subroutine */
  spec_psd(speckle,psd,n);

  return;

}
```

Listing B.7: circle_spec_psdmex.c

```
/*=====================================================================
 * Wrapper function for matlab to run subroutines written in C.
 *
 * this mex file is for computing the delta-removed speckle psd for the case
 * when the pupil is circular. This is done by using the largest inscribed
 * rectangle of data available in the pupil, computing the delta-removed psd,
 * then interpolating using an fft-zeropad-ifft approach.
 */
#define LOCAL
#include "mylibs.h"
#include "mex.h"
void psdtest(double *array, double *rslt, int n);

void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
  int     n,m;
  double *speckle, *psd;

  /* check for complex input field */
  if (mxIsComplex(prhs[0]))
    mexErrMsgTxt("Input speckle field must be real");

  /* error check the number of input and output arguments */
  if (nrhs != 1)
    mexErrMsgTxt("Only one input is allowed");
  if (nlhs !=1)
    mexErrMsgTxt("Must have one output argument");

  /* get the dimension of the field */
  n = mxGetN(prhs[0]);
  m = mxGetM(prhs[0]);
  if (n != m)
    mexErrMsgTxt("Input must be square");

  /* get a pointer to the input array */
  speckle = mxGetPr(prhs[0]);

  /* make the output array to go back to matlab and set the pointer */
  plhs[0] = mxCreateDoubleMatrix(m,n,mxREAL);
  psd = mxGetPr(plhs[0]);

  /* call the psd subroutine */
  circle_spec_psd(speckle,psd,n);

  return;

}
```

Listing B.8: nollmatrixmex.c

```
/*=========================================================================
 * Peter Johnson
 * 9 March 2006
 *
 * Mex function for computing the noll-covariance matrix for atmospheric
 * turbulence.  This is approximately 55 times faster than the matlab code
 * Pete Crabtree wrote, and gives the same results to within machine
 * precision.
 *
 * Calling Sequence:
 * M = nollmatrix(n)
 *
 * this will produce the nxn noll-covariance matrix, which must be scaled by
 * (D/r0)^(5/3) before it represents the covariance of zernike terms in
 * atmospheric turbulence.
 */

#ifndef M_PI
#define M_PI        3.14159265358979323846   /* pi */
#endif
#include "mex.h"
#define LOCAL
#include "mylibs.h"
#include "zernikes.h"

double noll_covariance(int n, int m);
/* C = noll_covariance(int n, int m); compute the zernike-mode covariance
 * between the n'th and m'th Noll-indexed zernike modes and assuming
 * Kolmogorov turbulence.  This can be used to build the covariance matrix,
 * which in turn can be used to generate phase screens.  To get the true
 * covariance, this needs to be multiplied by (D/r0)^(5/3).
 *
 * inputs:
 * int n == noll-index of the first mode
 * int m == noll-index of the second mode
 *
 * output:
 * double C == covariance of the requested modes
 */


void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[]);
/* mex wrapper function */


double noll_covariance(int i, int j)
{
  double C, A, B, D;        // working doubles
  int ni, mi, nj, mj;    // radial and angular orders for the modes
  int tmp, one;

  // compute the radial and angular mode numbers
  zernikemode(i, &ni, &mi);
  zernikemode(j, &nj, &mj);

  // see if we even need to bother
  if (((i-j)%2==0) && (mi==mj))
    {
      // get the exponent on the negative 1 and decide if -1^m is negative
      tmp = (ni + nj - 2*mi)/2;
      if (tmp%2 == 0)
        one = 1;
      else
        one = -1;

      A = 0.0072*one*sqrt((ni+1)*(nj+1))*pow(M_PI,(double)8/3);
      B = tgamma((double)14/3)*tgamma((ni+nj-(double)5/3)/2);
      D = tgamma((ni-nj+(double)17/3)/2)*tgamma((nj-ni+(double)17/3)/2) *
        tgamma((ni+nj+(double)23/3)/2);
      C = A*B/D;
    }
  else
    C = 0;
  return C;
}

void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
  int    n, c0, c1;
  double C, *M;

  /* check for complex input data */
  if (mxIsComplex(prhs[0]))
    mexErrMsgTxt("Input must be real");

  /* error check the number of input and output arguments */
  if (nrhs != 1)
    mexErrMsgTxt("need 1 input arg: number of modes");
  if (nlhs != 1)
    mexErrMsgTxt("need 1 output args: noll-matrix");

  /* figure out how many modes we are processing */
  n = *(mxGetPr(prhs[0]));

  /* make the output array to go back to matlab and set the pointers */
  plhs[0] = mxCreateDoubleMatrix(n,n,mxREAL);
  M = mxGetPr(plhs[0]);
  /* build up the noll covariance matrix */
  for (c0=0; c0<n; c0++)
    {
      for(c1=c0; c1<n; c1++)
        {
          C = noll_covariance(c0+2,c1+2);
          *(M+c1+n*c0) = C;
          if (c0 != c1)
            *(M+c0+n*c1) = C;
        }
    }


  return;
}
```

B.2 Data Generation and Reduction Code

This section contains code used to generate and post-process the data used for the SNR vs. frames analysis shown in Fig. 6.22. The `snr_frms_test` and `batchsub.sh` scripts are typical of the scripts used to run all of the scenarios described in chapter VI. The data generation scripts were run on a 64 node dual Opteron 248 Linux cluster and a 45 node Athlon 3000+ Linux cluster using the `PBS` batch processing system. Additional runs were done on a dual 2.8 GHz Pentium 4 Linux workstation and a dual 3.02 GHz Pentium 4 Linux workstation using `screen` and `bash` scripting.

Listing B.9: `snr_frms_test` for `bash` or PBS

```
#!/bin/bash
#PBS -m ae
#PBS -M peter.johnson@afit.edu
#PBS -l nodes=1:ppn=1,walltime=40000:00
#PBS -S /bin/bash
#PBS -r n
#PBS -V
#PBS -e err
#PBS -o out
# this file is for running the APDI alg on a given dataset for several
# conditioning bias levels for comparison.  The resulting output files are
# gzip'd together and stored in a directory for later analysis

# unset the DISPLAY env var
unset DISPLAY;

# set up some other variables
export HOME=/home/afit6/engphd07/pjohnson
export PROC=`uname -p`;          # decide what arch we are running on...
DATE=`date +%G%m%d`;             # date the script was run for filename

# see if the number of iterations was defined...
if [ ! -n "$ITERS" ]; then
    ITERS=100;                       # number of realizations to generate
fi
```

```
# check to see that we got snr, frms, and sigma from the environment
if [ -n "$SNR" ]; then
    export SNR                   # get the SNR from the caller
else
    export SNR=7;                # otherwise set a default SNR
fi

if [ -n "$FRM" ]; then           # get number of frames from caller
    export FRM;
else
    export FRM=30;               # otherwise set a default number of frms
fi

if [ -n "$SGMA" ]; then          # get the detector sigma from the caller
    export SIGMA
else
    export SGMA=6;               # otherwise set a default detector Sigma
fi

# set the directory to save the results in
BASEDIR=$HOME/dynamicaberr_data/snr_frms_test
SAVEDIR=$BASEDIR/${SNR}_snr/${FRM}frm/${SGMA}sgma

# make sure the directory is there to save stuff in
if [ ! -d $BASEDIR/${SNR}_snr ]; then
    mkdir $BASEDIR/${SNR}_snr
fi
if [ ! -d $BASEDIR/${SNR}_snr/${FRM}frm ]; then
    mkdir $BASEDIR/${SNR}_snr/${FRM}frm
fi
if [ ! -d $SAVEDIR ]; then
    mkdir $SAVEDIR
fi


# figure out which directory to run in from the environment, devault to 1 if
# not set.  This is so we don't overwrite files...
if [ -n "$WORKDIR" ]; then
    WORK=$BASEDIR/runtmp/$WORKDIR
else
    WORKDIR=1
    WORK=$BASEDIR/runtmp/$WORKDIR
fi

# make the directory if it isn't there
if [ ! -d $WORK ]; then
    mkdir $WORK;
fi

# change directory to the working directory
cd $WORK;

# set up architecture dependent stuff...

if [[ $PROC = x86_64 ]]; then
    MATLAB="/apps/Linux86_64/matlab14sp3/bin/matlab -nodisplay >& /dev/null"
    APDI=$HOME/bin/APDIsolve_64
    echo "64-bit Processor"
else
    MATLAB="/apps/Linux86/matlab14sp3/bin/matlab -nodisplay >& /dev/null"
    if [[ $PROC == athlon ]]; then
        APDI=$HOME/bin/APDIsolve_athlon
        echo "AMD Processor"
    else
        APDI=$HOME/bin/APDIsolve
        echo "Intel Processor"
    fi
fi

# loop over the iterations
for ((i=0; i<$ITERS; i++)); do


    # call matlab to write the data files...
    $MATLAB << EOF
proc = getenv('PROC');   % get the arch
home = getenv('HOME');   % find home
if strcmp(proc,'x86_64')        % set proc depend stuff
addpath(genpath(strcat(home,'/lib64')));
fprintf('\nx86_64\n');
else
addpath(genpath(strcat(home,'/lib')));
fprintf('\ni686\n');
end
rand('state',sum(100*clock));  % initialize rand number generator
randn('state',sum(100*clock)); % same for gaussian random numbers
snr = str2num(getenv('SNR'));            % get the SNR
frms = str2num(getenv('FRM'));           % get # of frames
det_sigma = str2num(getenv('SGMA'));     % gaussian noise std
writePDdata(snr,frms,det_sigma);  % generate data and write output files
% remove the unneeded guess data
!rm guess.dat

exit;
EOF
    wait

    # call the APDI algorithm
    $APDI
    wait

    # set the filename for this realization
    FILENAME=${DATE}_${WORKDIR}_${SNR}snr_${FRM}frms_${i}.tgz

    # tar-gzip the data files together and move them
    tar -czf $FILENAME *.dat
    rm *.dat
    mv $FILENAME $SAVEDIR

done
```

Listing B.10: `batchsub.sh`

```
#!/bin/bash
# Peter Johnson
# this launches a bunch of runs to the pbs server for queing

SNRVALS=( 2 3 5 7 10 15 20 );
snrlen=7;
FRMVALS=( 10 20 30 40 50 );
frmlen=5
export SGMA=100;

for ((i=0;i<$snrlen; i++)); do
    export SNR=${SNRVALS[${i}]}
    for ((j=0;j<$frmlen; j++)); do
        export FRM=${FRMVALS[${j}]}
        export WORKDIR=${i}${j}
        qsub snr_frms_test
        wait
    done
done
```

Listing B.11: `datareduce.m`

```
% Peter Johnson
% 23 May 2006
% datareduce.m
%
% this file cycles through the results directory and grinds up the data to
% get the MSE, mean MSE, and standard_deviation

clear; clc; close all hidden;

% set this to to save the graphics to files...
prnt=1;   % set to 1 to save individual pics
subfgs = 0;  % set to 0 to have all the figures in individual windows

% set up parameters
fname = '64x64_mse.csv';       % file name for results
SNR = {'2', '3', '5', '7', '10', '15', '20'};  % snr values
snrval = [2,3,5,7,10,15,20];
frms = {'10', '20', '30', '40', '50'};  % the number of frames
frmval = [10,20,30,40,50];
sgma = {'6','100'};                     % detector variances
sgmaval = [6 100];
base = '../';                           % base dir
workdir = strcat(base,'tmp/');          % working directory
stordir = strcat(base,'analysis/');     % directory to store results in
ind = 1:length(snrval)-1;

% make some storage arrays
l1 = length(SNR);
l2 = length(frms);
l3 = length(sgma);

raw_mse = zeros(l1,l2,l3);
adj_mse = zeros(l1,l2,l3);
scl_fact = zeros(l1,l2,l3);

raw_mse_sigma = zeros(l1,l2,l3);
adj_mse_sigma = zeros(l1,l2,l3);
slc_fact_sigma = zeros(l1,l2,l3);

raw_mse_stderr = zeros(l1,l2,l3);
adj_mse_stderr = zeros(l1,l2,l3);
scl_fact_stderr = zeros(l1,l2,l3);



mean_time = zeros(l1,l2,l3);
mean_time_sigma = zeros(l1,l2,l3);
mean_time_stderr = zeros(l1,l2,l3);

% define the awk command to look for the run time int the iterate.dat file
cmd = ['echo `awk ',char(39),'/Total User time/{print $4}',...
    char(39),' iterate.dat`'];

% open the file to store the results in and write colum titles
resfile = fopen(strcat(stordir,fname),'w');
fprintf(resfile,...
    'File,Frms,SNR,SGMA,Modes,Div,Dr0,Unproc_MSE,Scl_Cnst,Adj_MSE,Time\n');

% change to the work directory
cd(workdir);

% cycle through the frame numbers and SNR's
for c0=1:length(SNR)
  for c1=1:length(frms)
    for c4=1:length(sgma)
      srcdir = char(strcat(base,SNR(c0),'_snr/',frms(c1),'frm/',...
                            sgma(c4),'sgma/'));  % src dir
      fls = dir(strcat(srcdir,'*.tgz'));          % file listing
      fls = {fls.name};         % strip out names

      nruns = length(fls);
      if nruns>0

        % make a temp storage array for the mse's
        %unproc_mse_tmp = zeros(1,nruns(fls));
        adj_mse_tmp = zeros(1,nruns);
        raw_mse_tmp = zeros(1,nruns);
        scl_fact_tmp = zeros(1,nruns);
        time_tmp = zeros(1,nruns);

        for c2=1:nruns
          file = char(fls(c2));
          fprintf(strcat(file,'\n'));
          copyfile(strcat(srcdir,file),'.'); % copy dta to wrking dir
          unix(strcat('tar -xzf ',file));     % untar the data
          delete(file);                       % delete the data file

          % now read in the data and compute the desired metrics...

          % open the files for reading
          fl1 = fopen('results.dat','r','ieee-le');
          fl2 = fopen('truth.dat','r','ieee-le');
          fl3 = fopen('data.dat','r','ieee-le');

          % read in the truth data
          m = fread(fl2, 1, 'double');
          f = fread(fl2, 1, 'double');
          k = fread(fl2, 1, 'double');
          Dr0 = fread(fl2, 1, 'double');
          div = fread(fl2, 1, 'double');
          trth = reshape(fread(fl2, m*m, 'double'),m,m);
          fclose(fl2);

          % read in the results
          m = fread(fl1, 1, 'double');     % frame dimension
          f = fread(fl1, 1, 'double');     % number of frames
          k = fread(fl1, 1, 'double');     % number of aberrations
          Dr0 = fread(fl1, 1, 'double');   % D/r0
          div = fread(fl1,1,'double');     % diversity
          obj = reshape(fread(fl1, m*m, 'double'),m,m);  %  resulting obj
          fclose(fl1);

          % read in the unprocessed data
          m = fread(fl3, 1, 'double');     % frame dimension
          f = fread(fl3, 1, 'double');     % number of frames
          k = fread(fl3, 1, 'double');     % number of aberrations
          Dr0 = fread(fl3, 1, 'double');   % D/r0
          div = fread(fl3,1,'double');     % diversity
          tmpdata = reshape(fread(fl3,2*m*m*f,'double'),m,m,2*f); % data
          fclose(fl3);
          focal = zeros(m);
          for frmcntr=1:f
            focal = focal + tmpdata(:,:,2*frmcntr-1);
          end
          focal = focal./f;
          clear tmpdata;

          % compute the raw MSE, scale factor, and adjusted MSE
          unproc_mse_tmp(c2) = mean(mean((trth-focal).^2)); % unprocessed mse
          raw_mse_tmp(c2)    = mean(mean((trth - obj).^2));     % raw mse
          scl_fact_tmp(c2) = sum(sum(trth.*obj))./sum(sum(obj.^2));
```

```
          obj = scl_fact_tmp(c2).*obj;                      % adj object
          adj_mse_tmp(c2) = mean(mean((trth-obj).^2));      % adjusted mse

          % get the computation time for this run
          [status,t] = unix(cmd);
          if (length(t)>1)
            time_tmp(c2) = str2num(t);
          else
            time_tmp(c2) = NaN;
          end

          % write the results to the output file
          fprintf(resfile,strcat(file,',%g,',char(SNR(c0)),',', ...
                                 char(sgma(c4)),',%g,%g,%g,%g,%g,%g,%g\n'),...
                  f,k,div,Dr0, unproc_mse_tmp(c2), scl_fact_tmp(c2), ...
                  adj_mse_tmp(c2),time_tmp(c2));

          % clean up tmp directory
          unix('rm *.dat');
        end

        % compute the statistics
        raw_mse(c0,c1,c4) = mean(raw_mse_tmp);
        raw_mse_sigma(c0,c1,c4) = std(raw_mse_tmp);
        raw_mse_stderr(c0,c1,c4) = raw_mse_sigma(c0,c1,c4)/sqrt(nruns);

        adj_mse(c0,c1,c4) = mean(adj_mse_tmp);
        adj_mse_sigma(c0,c1,c4) = std(adj_mse_tmp);
        adj_mse_stderr(c0,c1,c4) = adj_mse_sigma(c0,c1,c4)/sqrt(nruns);

        scl_fact(c0,c1,c4) = mean(scl_fact_tmp);
        scl_fact_sigma(c0,c1,c4) = std(scl_fact_tmp);
        scl_fact_stderr(c0,c1,c4) = scl_fact_sigma(c0,c1,c4)/sqrt(nruns);

        timeind = find(isnan(time_tmp)==0);
        mean_time(c0,c1,c4) = mean(time_tmp(timeind));
        mean_time_sigma(c0,c1,c4) = std(time_tmp(timeind));
        mean_time_stderr(c0,c1,c4) = mean_time_sigma(c0,c1,c4)/sqrt(nruns);

      else
        raw_mse(c0,c1,c4) = NaN;
        raw_mse_sigma(c0,c1,c4) = 0;
        raw_mse_stderr(c0,c1,c4) = 0;

        adj_mse(c0,c1,c4) = NaN;
        adj_mse_sigma(c0,c1,c4) = 0;
        adj_mse_stderr(c0,c1,c4) = 0;

        scl_fact(c0,c1,c4) = NaN;
        scl_fact_sigma(c0,c1,c4) = 0;
        slc_fact_stderr(c0,c1,c4) = 0;

        mean_time(c0,c1,c4) = NaN;
        mean_time_sigma(c0,c1,c4) = 0;
        mean_time_stderr(c0,c1,c4) = 0;
      end
    end
  end
end
% close the files.
fclose('all');  % just in case...

% save the results for later playing
save snr_frm_reduced_data

% return to the analysis directory
```

```
cd(stordir)

% plot some of the reduced results
set(0,'DefaultAxesLineStyleOrder',{'-^','-*','-o','-x','-+','-d'})
linestyles=['b-^';'g-*';'r-o';'k-p';'c-d';'m-+'];

% expand the snrval matrix
snrmat = repmat(snrval',1,length(frmval));

% make on plot with all the mean MSE's v.s. SNR and frames
for c33=1:length(sgma)
  h1 = figure;
  hold on
  s = size(snrmat);
  for c0=1:s(2)
    errorbar(snrmat(ind,c0),adj_mse(ind,c0,c33),...
             adj_mse_stderr(ind,c0,c33), linestyles(c0,:))
    %plot(snrmat(ind,c0),adj_mse(ind,c0,c33),linestyles(c0,:));
  end
  hold off

  xlabel('SNR','FontSize',14);
  ylabel('MSE','FontSize',14);
  title(['MSE v.s. Detection SNR, \sigma_d=',char(sgma(c33))],'FontSize',14)
  a = axis;
  a(2) = max(snrval(ind))+1;
  a(1) = min(snrval(ind))-1;
  axis(a)

  grid on;
  legend([char(frms(1)),' frms'],[char(frms(2)),' frms'],...
         [char(frms(3)),' frms'],[char(frms(4)),' frms'],...
         [char(frms(5)),' frms'],'Location','NorthEast');
  if prnt
    prntfname = strcat(char(sgma(c33)),'sgma_snr_frms_mse_dynamic.eps');
    print(h1,'-depsc',prntfname);
  end


  % make on plot with all the mean times v.s. SNR and frames
  h2 = figure;
  hold on
  s = size(snrmat);
  for c0=1:s(2)
    errorbar(snrmat(ind,c0),mean_time(ind,c0,c33),...
             adj_mse_stderr(ind,c0,c33), linestyles(c0,:))
    % plot(snrmat(ind,c0),mean_time(ind,c0,c33),linestyles(c0,:));
  end
  hold off
  xlabel('SNR','FontSize',14);
  ylabel('Computation Time (s)','FontSize',14);
  title(['Time v.s. SNR, \sigma_d=',char(sgma(c33))],'FontSize',14)
  a = axis;
  a(2) = max(snrval(ind))+1;
  a(1) = min(snrval(ind))-1;
  axis(a)
  grid on;
  legend([char(frms(1)),' frms'],[char(frms(2)),' frms'],...
         [char(frms(3)),' frms'],[char(frms(4)),' frms'],...
         [char(frms(5)),' frms'],'Location','ne')
  if prnt && ~subfgs
    prntfname = strcat(char(sgma(c33)),'sgma_snr_frms_time_dynamic.eps');
    print(h2,'-depsc',prntfname);
  end
end
```

```
% put out a plot with the computation time for the noiseless case v.s. frms
h3 = figure
errorbar(frmval,mean_time(end-1,:,1),mean_time_stderr(end-1,:,1))
xlabel('Frames','FontSize',14)
ylabel('Time (s)','FontSize',14)
title('Mean Computation Time vs. Number of Frames','FontSize',14)
grid on
if prnt
  prntfname = strcat(char(SNR(end-1)),'snr_frms_time_lineplot_dynamic.eps');
  print(h3,'-depsc',prntfname)
end
```

Bibliography

1. Andrews, H. C. and B. R. Hunt. *Digital Image Restoration.* Prentice Hall, 1977.

2. Andrews, Larry C. and Ronald C. Philips. *Laser Beam Propagation through Random Media.* SPIE Optical Engineering Press, 1998.

3. Bakut, Peter. *Theoretical studies of Fourier telescopy for deep space imaging.* Technical Report DTIC: ADA368728, International Informatization Academy, Laser and Information Technologies Department, Volokolamskoe sh., 95, Moscow, 123424 Russia, September 1999. Contract F61775-98-WE012.

4. Beckmann, Petr and André Spizzichino. *The Scattering of Electromagnetic Waves from Rough Surfaces.* Artech House, 1987.

5. Bilmes, Jeff A. *A gentle tutorial of the EM Algorithm and its application to parameter estimation for gaussian mixture and hidden markov models.* Technical Report TR-97-021, International Computer Science Institute, April 1998.

6. Boger, J. K. "Pupil-plane speckle imaging with a referenced polarization technique". *Optics Letters*, 24(9):611–613, May 1999.

7. Boger, J. K. and M. P. Fetrow. "Pupil-plane speckle imaging using referenced polarization for wavefront estimations". *Proceedings of the SPIE*, 3815:90–97, 1999.

8. Born, Max and Emil Wolf. *Principles of Optics.* Cambridge University Press, sixth edition, 2003.

9. ten Brummelaar, Theo. "Temporal power spectra of Zernike coefficients". *Proceedings of the SPIE*, 2200:418–421, February 1994.

10. ten Brummelaar, Theo. "The contribution of high order Zernike modes to wavefront tilt". *Optics Communications*, 115:417–424, 1995.

11. Byrd, Richard H., Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. *A limited memory algorithm for bound constrained optimization.* Technical Report NAM-08, Northwestern University Department of Electrical Engineering and Computer Science, May 1994.

12. Dainty, J. C. (editor). *Topics in Applied Physics, Laser Speckle and Related Phenomena*, volume 9. Springer-Verlag, second edition, 1984.

13. Dellaert, Frank. *The Expectation Maximization Algorithm.* Technical Report GIT-GVU-02-20, College of Computing, Georgia Institute of Technology, February 2002.

14. Demmel, James W. *Applied Numerical Linear Algebra.* Society for Industrial and Applied Mathematics, 1997.

15. Dempster, A. P., N. M. Laird, and D. B. Rubin. "Maximum-likelihood from incomplete data via the EM algorithm". *Journal of the Royal Statistical Soceity, Series B*, 39, 1977.

16. Dudney, Robert S. "The struggle for space". *Air Force Magazine*, 87(3):4, March 2004.

17. Fienup, J. R. "Reconstruction of an object from the modulus of its Fourier transform". *Optics Letters*, 3(1):27–29, July 1978.

18. Fletcher, R. *Practical Methods of Optimization*, volume 2: Constrained Optimization. John Wiley and Sons, 1981.

19. Frigo, Matteo and Steven G. Johnson. "The design and implementation of FFTW3". *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

20. Gamiz, Victor, Laura Ulibarri, and Scott Long. "DCBeam algorithm overview", April 2005. Personal Communication.

21. Gill, Philip E., Walter Murray, and Margaret H. Wright. *Practical Optimization*. Academic Press, 1981.

22. Gonsalves, Robert A. and Robert H. Chidlaw. *Analytical studies of phase estimation Techniques. Volume I.* Technical Report DTIC:ADA078006, Eikonix Corporation, 1979. DARPA Contract: F30602-77-C-0176.

23. Gonsalves, Robert A. and Robert H. Chidlaw. *Analytical studies of phase estimation techniques.* Technical Report DTIC:ADA086285, Eikonix Corporation, 1980. DARPA Contract: F30602-77-C-0176.

24. Goodman, Joseph W. *Statistical Optics.* Wiley Interscience, 1985.

25. Goodman, Joseph W. *Introduction to Fourier Optics.* McGraw-Hill, second edition, 1996.

26. Hecht, Eugene. *Optics.* Addison Wesley, fourth edition, 2002.

27. Herbert, Adam J. "Toward supremacy in space". *Air Force Magazine*, 88(1):22–28, January 2005.

28. Holmes, R. B. and Mikhail S. Belen'kii. "Investigation of the Cauchy-Riemann equations for one-dimensional images recovery in intensity interferometry". *Journal of the Optical Society of America A*, 21(5):697–706, May 2004.

29. Holmes, R. B. and T. Brinkley. "Reconstruction of images of deep space objects using Fourier telescopy". *Proceedings of the SPIE*, 3815:11–22, July 1999.

30. Holmes, R. B., K Hughes, P. Fairchild, B. Spivey, and A Smith. "Description and simulation of an active imaging technique utilizing two speckle fields: iterative reconstructors". *Journal of the Optical Society of America A*, 19(3):458–471, March 2002.

31. Holmes, R. B., K Hughes, P. Fairchild, B. Spivey, and A Smith. "Description and simulation of an active imaging technique utilizing two speckle fields: root reconstructors". *Journal of the Optical Society of America A*, 19(3):444–457, March 2002.

32. Holmes, R. B., S. Ma, A. Bhowmik, and C. Greninger. "Analysis and simulation of a synthetic-aperture technique for imaging through a turbulent medium". *Journal of the Optical Society of America A*, 13(2):351–364, February 1996.

33. Hu, Sze-Tsen. *Homotopy Theory.* Academic Press, 1959.

34. Hunt, B. R., T. L. Overman, and Peter Gough. "Image reconstruction from pairs of Fourier-transform magnitude". *Optics Letters*, 23(14):1123–1125, 1998.

35. Leon-Garcia, Alberto. *Probability and Random Processes for Electrical Engineering.* Addison Wesley, second edition, 1994.

36. McLachlan, Geoffrey J. and Thriyambakam Krishnan. *The EM Algorithm and Extensions.* Wiley-Interscience, 1996.

37. Noll, Robert J. "Zernike polynomials and atmospheric turbulence". *Journal of the Optical Society of America*, 66(3):207–211, March 1976.

38. Palik, Edward D. *Handbook of Optical Constants of Solids.* Academic Press, 1997.

39. Paxman, Richard G., Timothy J. Schulz, and James R. Fienup. "Joint estimation of object and aberrations by using phase diveristy". *Journal of the Optical Society of America A*, 9(7):1072–1085, July 1992.

40. Roddier, F., M. J. Northcott, J. E. Graves, and D. L. Mckenna. "One-dimensional spectra of turbulence-induced Zernike aberrations: Time-delay and isoplanicity error in partial adaptive compensation". *Journal of the Optical Society of America A*, 10(5):957–965, May 1993.

41. Roggemann, Michael C. and Byron M. Welsh. *Imaging Through Turbulence.* CRC Press, 1996.

42. Schulz, Timothy J. "Multiframe blind deconvolution of astronomical images". *Journal of the Optical Society of America A*, 10(5):1064–1073, May 1993.

43. Schulz, Timothy J. "A fast algorithm for maximum-likelihood imaging with coherent speckle measurements". *Proceedings of the International Conference on Image Processing*, volume 1, 679–682. 1997.

44. Schulz, Timothy J. and D. L. Snyder. "Image recovery from correlations". *Journal of the Optical Society of America A*, 9(8):1266–1272, August 1992.

45. Seldin, John H. and Richard G. Paxman. "Phase-diverse speckle reconstruction of solar data". *Proceedings of the SPIE*, 2302:268–280, 1994.

46. Stremler, Ferrel G. *Introduction to Communication Systems.* Addison Wesley, third edition, 1992.

47. Szeto, Roque Kwok-Hung. "Analysis of unit-magnitude constrained phasor reconstruction problems". *Journal of the Optical Society of America A*, 14(7):1412–1420, July 1997.

48. Thelen, Brian J., Richard G. Paxman, David A. Carrara, and John H. Seldin. "Maximum *a posteriori* estimation of fixed aberrations, dynamic aberrations, and the object from phase-diverse speckle data". *Journal of the Optical Society of America A*, 16(5):1016–1025, May 1999.

49. van Trees, Harry L. *Detection, Estimation, and Modulation Theory, part I.* John Wiley and Sons, 1968.

50. Voelz, David G., John F. Belsher, Laura Ulibarri, and Victor Gamiz. "Ground-to-space laser imaging: review 2001". *Proceedings of the SPIE*, 4489:35–47, 2002.

51. Yariv, Amnon and Pochi Yeh. *Optical Waves in Crystals.* Wiley Interscience, 1984.

Index

The index is conceptual and does not designate every occurrence of a keyword.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704–0188*

1. REPORT DATE *(DD–MM–YYYY)* 22–03–2007	**2. REPORT TYPE** PhD Dissertation	**3. DATES COVERED** *(From — To)* Mar 2005 — Mar 2007

4. TITLE AND SUBTITLE Phase Diversity and Polariztion Augmented Techniques for Active Imaging	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) PETER M. JOHNSON, Capt, USAF	**5d. PROJECT NUMBER** ENG 06–244
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/DS/ENG/07-05
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Dr. Victor L. Gamiz, Active Track Program Manager AFRL/DESA, Air Force Materiel Command 3550 Aberdeen Ave SE Kirtland Air Force Base, NM, 87117 (505)846-4846, victor.gamiz@kirtland.af.mil	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

A multi-frame active phase diversity imaging (APDI) algorithm is derived for coherent light statistics and demonstrated. In addition to conventional focal-plane and diversity-plane data, a statistical description for pupil-plane (PP) intensity is formed and included in the derivation. The algorithm is implemented and characterized via Monte Carlo simulation. Analysis shows that it's robust, insensitive to detection noise for $\text{SNR} \geq 7$, performs well for SNR's as low as 2, and that the effect of system configuration on optimal parameters is minimal. Furthermore, introduction of PP data results in a 60% better reconstruction from dynamically aberrated data than obtained using only focal-plane and diversity-plane data. Both an EM-algorithm and a lensless-APDI approach are presented for generating imagery directly from PP polarization measurements. However, both approaches are currently impractical. Suggestions for improvement are offered. Finally, the APDI algorithm is modified to use PP polarization data in place of PP intensities. An initial statistical model is offered, and suggestions for performance improvement are presented.

15. SUBJECT TERMS

active imaging, phase diversity, multiframe blind deconvolution, deconvolution, coherent light, turbulence, image reconstruction, laser speckle, speckle, active phase diversity, phase-diversity

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Dr. Richard K. Martin
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(include area code)*
U	U	U	UU	212	(937)255–3636, x4625; richard.martin@afit.edu