Investigation of the Noise Floor of the Standard PIV Cross-Correlation Algorithm

Kyle L. Jones
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/gradreports

Part of the Mechanical Engineering Commons

Recommended Citation
https://digitalcommons.usu.edu/gradreports/181

This Report is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Plan B and other Reports by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.
INVESTIGATION OF THE NOISE FLOOR OF THE STANDARD PIV CROSS-CORRELATION ALGORITHM

by

Kyle L. Jones

A report submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE in

Mechanical Engineering

Approved:

Dr. Barton L. Smith
Major Professor

Dr. Robert Spall
Committee Member

Dr. Aaron Katz
Committee Member

Dr. Heng Ban
Committee Member

UTAH STATE UNIVERSITY
Logan, Utah

2012
Abstract

Investigation of the Noise Floor of the Standard PIV Cross-Correlation Algorithm

by

Kyle L. Jones, Master of Science
Utah State University, 2012

Major Professor: Dr. Barton L. Smith
Department: Mechanical and Aerospace Engineering

Particle image velocimetry (PIV) is a powerful measurement technique used to acquire instantaneous measurements of entire flow fields at a given instant in time. Quantifying the uncertainty and error in PIV is a critical part of realizing the full potential of PIV as a flow measurement technique.

The noise floor of PIV is the minimum amount of random error that can be achieved for a particular standard cross-correlation (SCC) algorithm. The noise floor of the SCC used by DaVis in correlating image pairs is explored. Two methods for creating image pairs for correlation are compared, namely pseudo image pairs and artificial image pairs. A common PIV experimental setup with seeded water in a glass tank was used to acquire images at \( dt \approx 0 \) seconds between images. The aperture or \( f\# \) of the lens was varied in order to achieve a range of particle image diameters at two different magnifications. A Matlab code was written to upsample, shift and downsample the images by a prescribed, sub-pixel displacement. The shifted images were then imported into DaVis and correlated, resulting in displacement vector images. The random error of these images were calculated and each particle diameter is compared.
The random and bias errors of the DaVis and PRANA SCC algorithms were also compared for a fixed, optimum particle image diameter and multiple sub-pixel displacements between 0 and 1 pixel.
Public Abstract

Investigation of the Noise Floor of the Standard PIV Cross-Correlation Algorithm

Kyle L. Jones

Particle image velocimetry is a powerful flow measurement technique that allows one to measure entire velocity flow fields at a given instant in time. In general, tracer particles are added to the fluid to be studied and a camera is placed perpendicular to a laser sheet, which is used to illuminate the particles in the flow field. PIV measurements can be performed in any transparent liquid and in a wide range of flow conditions. A pair of images of the tracer particles are taken at successively and then using a statistical approach known as a cross-correlation a velocity field is calculated.

As PIV gains momentum as a flow measurement technique, new rise has been given to determining its accuracy and isolating potential error sources. The noise floor of PIV is the minimum amount of random error present in a particular flow measurement and is the subject of this report. Two methods for determining the noise floor are investigated for multiple particle diameters using digital PIV data known as pseudo image generation and artificial image generation. The former involves taking the first image and shifting it by a known amount, creating an image pair with a shifted version of itself, while the latter involves shifting the second image in a pair, taken at approximately 0 seconds apart and shifting it to create the image pair. The application of three shifting methods as explored, namely the linear interpolation method and the nearest neighbor interpolation method with two separate filters. The image pairs created from these methods are correlated using LaVision’s DaVis software and the noise floor is determined for a single sub-pixel displacement. Multiple sub-pixel displacements are also explored for a given particle diameter. The random and bias errors of these displacements are compared for DaVis and PRANA cross-correlation software.
To my wife and son, who’s smiling faces make the most difficult of days bearable.
Acknowledgments

Thanks to Dr. Barton Smith for always having my best interests in mind and guiding me through, Jeff Harris for help in developing the idea, Burke Andrus for taking data, Scott Warner for supplemental Matlab codes, my coworkers at the EFDL for helping me stay focused and positive, my parents Lenny and Mary Ann and my parents-in-law Jeff and Kathy for their love and encouraging support, and especially my wife Chelsi and son Asher for listening and reminding me what all this is for.

Kyle L. Jones
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Public Abstract</td>
<td>v</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>vii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>x</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xi</td>
</tr>
<tr>
<td>Notation</td>
<td>xiii</td>
</tr>
<tr>
<td>Acronyms</td>
<td>xiv</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Particle Image Velocimetry</td>
<td>2</td>
</tr>
<tr>
<td>1.1.1 Cross-Correlation</td>
<td>3</td>
</tr>
<tr>
<td>1.1.2 PIV Uncertainty</td>
<td>4</td>
</tr>
<tr>
<td>2 Objectives</td>
<td>6</td>
</tr>
<tr>
<td>3 Experimental Description</td>
<td>7</td>
</tr>
<tr>
<td>3.1 Experimental Setup</td>
<td>7</td>
</tr>
<tr>
<td>3.2 Data Acquisition</td>
<td>8</td>
</tr>
<tr>
<td>3.3 PIV Noise Floor</td>
<td>9</td>
</tr>
<tr>
<td>3.3.1 Pseudo Image Pair Generation</td>
<td>9</td>
</tr>
<tr>
<td>3.3.2 Shifted $\delta t$ Image Pair Generation</td>
<td>10</td>
</tr>
<tr>
<td>3.4 Image Resampling</td>
<td>10</td>
</tr>
<tr>
<td>3.4.1 Filter Application</td>
<td>12</td>
</tr>
<tr>
<td>3.4.2 Image Shifting and Downsampling</td>
<td>15</td>
</tr>
<tr>
<td>4 Results</td>
<td>18</td>
</tr>
<tr>
<td>4.1 Particle Image Diameter</td>
<td>18</td>
</tr>
<tr>
<td>4.2 Image Shifting</td>
<td>18</td>
</tr>
<tr>
<td>4.3 Noise Floor</td>
<td>23</td>
</tr>
<tr>
<td>4.4 Subpixel Displacement</td>
<td>24</td>
</tr>
<tr>
<td>5 Conclusion</td>
<td>27</td>
</tr>
<tr>
<td>References</td>
<td>29</td>
</tr>
</tbody>
</table>
Appendices

Appendix A  Matlab Image Shifting Code ........................................... 32
   A.1  Image Shifting Driver ....................................................... 32
   A.2  Image Shifting Functions .................................................. 38

Appendix B  Matlab Vector Analysis Code ................................. 47
   B.1  Get Processed Vectors Code ............................................. 47
   B.2  Compute Noise Floor Code ............................................... 48
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 The lens apertures or $f#$ and aperture diameters $D_a$ used in this study to vary the particle image diameter $d_\tau$. The estimated $d_\tau$ in pixels for the near-field ($M = 0.25$) and far-field ($M = 0.53$) cases are also shown.</td>
<td>9</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 A typical experimental setup for PIV in a wind tunnel from Raffel [7].</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Example of a correlation map.</td>
<td>4</td>
</tr>
<tr>
<td>3.1 Aquarium setup for PIV data acquisition.</td>
<td>8</td>
</tr>
<tr>
<td>3.2 An example of digital acquisition of an analog particle image in the form of a sinc function. (a) Particle as seen by the camera. (b) The intensity profile for the particle. (c) The particle image as output by the camera sensor.</td>
<td>11</td>
</tr>
<tr>
<td>3.3 An example of the nearest neighbor upsampling technique. The pixels marked with red dots • represent the original pixels in both the original image space and upsampled image space. The upsampled scale of this example is 5 pixels. The original image pixel space is represented by the dashed line.</td>
<td>13</td>
</tr>
<tr>
<td>3.4 Comparison of the sinc function and the Lanczos function with two side-lobes (Lanczos2) and three side-lobes (Lanczos3).</td>
<td>14</td>
</tr>
<tr>
<td>3.5 The 2-D Lanczos function with three side-lobes (Lanczos3).</td>
<td>15</td>
</tr>
<tr>
<td>3.6 Comparison of a single particle’s intensity profile for the (a) linear interpolation, (b) two side-lobed Lanczos filter, and (c) three side-lobed Lanczos filter.</td>
<td>17</td>
</tr>
<tr>
<td>4.1 Comparison of the measured particle image diameter $d_r$ to the theoretical diameter calculated by Eq. (4.1) for (a) near-field case ($M=0.53$) and (b) far-field case ($M=0.25$).</td>
<td>19</td>
</tr>
<tr>
<td>4.2 Pseudo image pair result of linear interpolation case. (a) A portion of the original image, (b) the same portion of the image after shifting.</td>
<td>20</td>
</tr>
<tr>
<td>4.3 Pseudo image pair result of two side-lobed Lanczos filter case. (a) A portion of the original image, (b) the same portion of the image after shifting.</td>
<td>20</td>
</tr>
<tr>
<td>4.4 Pseudo image pair result of three side-lobed Lanczos filter case. (a) A portion of the original image, (b) the same portion of the image after shifting.</td>
<td>21</td>
</tr>
<tr>
<td>4.5 Comparison of the FTs for each interpolation case to the FT of the original image.</td>
<td>22</td>
</tr>
</tbody>
</table>
4.6 Convergence of the random error for the Lanczos3 method for the near-field case ($M = 0.53$) and the far-field case ($M = 0.25$).

4.7 Random error for $U$ and $V$ velocities for far- and near-field cases with $\tilde{x} = 2/9$ pixels. (a) Random error for shifted 0dt image pairs. (b) Random error for pseudo image pairs.

4.8 Comparison of (a) random, (b) bias, and (c) total errors for a single $f^*$ and multiple displacements between 0 and 1 pixel for the DaVis and PRANA SCC algorithms.
Notation

\( \beta \) Bias error
\( D_a \) Aperture diameter
\( D_I \) Interrogation region width
\( d_P \) Particle diameter
\( d_r \) Particle image diameter
\( dt \) Time between images
\( \epsilon \) Random error of single measurement
\( f\# \) f-number of lens (aperture)
\( M \) Magnification factor
\( N \) Number of images
\( N_v \) Total number of velocity vectors
\( N_{v_i} \) Number of velocity vectors per image
\( S_\epsilon \) Random error
\( S_x \) Number of pixels shifted in the upsampled space
\( s \) Number of side lobes on Lanczos function
\( s_0 \) Object’s distance
\( U \) Total error
\( U_s \) Upsample scaling factor
\( x \) Single displacement measurement
\( \ddot{x} \) Prescribed displacement
\( z \) Particle’s distance from focal plane
## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2C PIV</td>
<td>Two-component particle image velocimetry</td>
</tr>
<tr>
<td>3C PIV</td>
<td>Three-component particle image velocimetry</td>
</tr>
<tr>
<td>FT</td>
<td>Fourier transform</td>
</tr>
<tr>
<td>Lanczos2</td>
<td>Two side-lobed Lanczos function</td>
</tr>
<tr>
<td>Lanczos3</td>
<td>Three side-lobed Lanczos function</td>
</tr>
<tr>
<td>PIV</td>
<td>Particle image velocimetry</td>
</tr>
<tr>
<td>PTV</td>
<td>Particle tracking velocimetry</td>
</tr>
<tr>
<td>RMS</td>
<td>Root mean square</td>
</tr>
<tr>
<td>RSS</td>
<td>Root sum square</td>
</tr>
<tr>
<td>RPC</td>
<td>Robust phase correlation</td>
</tr>
<tr>
<td>SCC</td>
<td>Standard cross correlation</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Quantifying uncertainty in Particle Image Velocimetry (PIV) is a critical part of realizing the full potential of PIV as a flow measurement technique. Many sources can induce error, including calibration, magnification, perspective error, resolution, and sub-pixel interpolation [1]. With so many sources of error, it is necessary to determine which of these sources has the largest impact and which sources can be neglected. The cross-correlation algorithm used to calculate the magnitude and direction of the flow is a large source of PIV error. Sub-pixel displacements, regions of shear in the flow, and particle image size can generate uncertainty in the correlation algorithm.

Much of the uncertainty of PIV measurements comes from the correlation algorithm. Random errors are introduced in the correlation algorithm through wide or multiple peaks in the correlation map. Noise from sub-pixel displacements can generate velocity fluctuations that are not present in the physical flow. With so many sources that impact the accuracy of PIV measurements, it is impossible to control all parameters. Isolation of each parameter in particular is the best way to estimate their impact on PIV uncertainty.

The purpose of this study is to determine the minimum noise level, or noise floor, of the correlation algorithm using various methods of generating particle images based on real data. The noise floor can be estimated by using a single image as acquired by the PIV data acquisition system and shifting the image by a known displacement, generating a pseudo image pair. A vector field is computed from this pair and compared to the specified displacement. The resulting error is the basis for the noise floor of the correlation algorithm. Other studies [2] have attempted to compare the noise floor of different correlation algorithms, this study will employ LaVision’s DaVis cross-correlation algorithm [3] for a single sub-pixel displacement and multiple particle image diameters and will incorporate DaVis
as well as PRANA in comparing multiple particle image displacements for a single particle image diameter.

This study focuses on the process of resampling, filtering and shifting of real image data for generation of shifted $0dt$ and pseudo image pairs, the method for determining the accuracy of the image generation process, and the noise floor resulting from each generation method. The resampling methods used in this study include the linear interpolation method as discussed in [4, 5] and the nearest neighbor technique coupled with two and three side-lobed Lanczos filters. For the context of this study, MATLAB® was used for all image processing techniques [6] except the image cross-correlation.

1.1 Particle Image Velocimetry

Over the past few decades, Particle Image Velocimetry (PIV) has come to dominate laboratory velocity measurements. PIV is used to “determine instantaneous fields of the vector velocity by measuring the displacements of numerous fine particles that accurately follow the motion of the fluid [1].” Unique strength comes from the ability of PIV to measure entire flow fields at a given instant in time where most other techniques measure at a single point.

Experiments using PIV consist of several subsystems [7]. Flow visualization requires that the flow contain particles that accurately follow the flow. In most cases tracer particles are added to the flow. However, it has been shown that bubbles such as those seen in nucleate boiling experiments can be used in PIV [8]. The particles must be illuminated and imaged along a plane. One camera coupled with a laser sheet can record the displacement of the particles. This can be performed on a single frame or with multiple frames, although multiple frame PIV has become more common with the use of digital cameras. A typical two velocity component PIV (2C PIV) experimental arrangement from Raffel [7] is shown in Fig. 1.1.

After acquiring the images, each image is divided into interrogation regions. A cross-correlation method is used to determine the displacement of the particles, resulting in a single vector. This method is applied across the entire image resulting in a vector field
Fig. 1.1: A typical experimental setup for PIV in a wind tunnel from Raffel [7].

representing the flow for a given instant in time.

In contrast to many other velocity measurement techniques, PIV is noninvasive, allowing one to acquire velocity measurements without disrupting the flow or in an enclosed facility such as a wind tunnel. “This allows the application of PIV even in high speed flows with shocks or in boundary layers close to the wall, where the flow may be disturbed by the presence of probes [7].”

“PIV is a technique synonymous with compromises – at least when it comes to the choice of parameters [6].” PIV has certain limits of application including: PIV is limited to transparent media, PIV has large uncertainty in regions of high shear, and uncertainty increases with very low seeding density. It is therefore important to determine exactly what parameters are to be measured and ensure that the experimental setup properly exhibits the desired variables.

1.1.1 Cross-Correlation
The methods by which an image can be converted into velocity measurements are the auto-correlation and cross-correlation [10]. The auto-correlation is used for single-frame, double-exposure PIV where the cross-correlation is applied to multiple-frame, single-exposure PIV. These correlations are used in signal processing [11, 12], where the cross-correlation can be considered the similarity between two signals and the auto-correlation is the similarity of one signal with itself. Each interrogation region in PIV is a two-dimensional intensity signal to which a correlation can be applied. A resulting correlation map contains a peak that represents the actual displacement of the particles in the interrogation area. Figure 1.2 shows an example of a correlation map with a single peak representing the particle displacement for the interrogation area.

1.1.2 PIV Uncertainty

Coleman and Steele [13] describe uncertainty as “the degree of goodness of a measurement, experimental result, or analytical (simulation) result,” and is “an estimate of a range within which we believe the actual (but unknown) value of an error lies.” Many studies have been performed in an attempt to quantify the accuracy of PIV measurements.
Nogueira et al. [14] has shown that removing spurious vectors from the processed PIV data can improve accuracy. Multiple methods for spurious vector detection are presented by Westerweel [15]. Griffin et al. [16] suggests a statistical-based approach for outlier detection using a multivariate approach assuming a large enough sample for accurate detection. Charonko and Vlachos [17] present a method for predicting the uncertainty of individual velocity measurements based on the ratio of primary to secondary peak heights from the standard cross-correlation (SCC) and robust phase correlation (RPC) methods. Kähler, Scharnowski, and Cierpka [18] presented methods for improving the accuracy of PIV in regions of high shear and near walls using a single-pixel ensemble-correlation and using particle tracking velocimetry (PTV). Timmins, Wilson, Smith, and Vlachos [19] present a method for automatically determining the uncertainty of local PIV measurements based on parameters that contribute error to the result. Harris, Smith, and Wilson [2] investigate the impact of different error sources on PIV measurements, including the noise floor of the SCC algorithm, indicating that all the investigated error sources contributed little to the overall uncertainty of PIV measurements.
Chapter 2

Objectives

This report has the following objectives:

1. Assemble test facility with a simple jet for acquiring PIV data.

2. Acquire digital PIV data with time between images \( dt \approx 0 \ \mu s \) for 8 particle image diameters by adjusting lens aperture diameter \( D_a \) or lens f-stop \( f^\# \) at two different magnifications (Near-field \( M = 0.53 \) and Far-field \( M = 0.25 \)).

3. Write a code in Matlab to shift the images by a prescribed displacement using pseudo image pair and artificial image pair methods.
   a) Apply the shifting method proposed by Petrie et al. [4].
   b) Use nearest neighbor technique for resampling.
   c) Apply various filters to upsampled images to simulate analog signals.
   d) Compare the results of these methods.

4. Import the image pairs into DaVis and correlate them.

5. Write Matlab code to import vector files and calculate the random error of the cross-correlation algorithm.

6. Using a single \( f^\# \), displace the images \( \tilde{x} = 0/9, 1/9, ..., 9/9 \) pixels.

7. Correlate the image pairs in DaVis and PRANA.

8. Modify the existing Matlab code to compile the displacement data and calculate the random and bias errors for each case.
Chapter 3
Experimental Description

Sub-pixel interpolation allows the accuracy of non-integer displacements to be achieved, greatly increasing the dynamic range of PIV measurements. One purpose of this project is to determine the noise floor for the DaVis correlation method [3] using various image shifting methods. To provide confidence in the results, the particle image diameter $d_r$ was changed by adjusting the aperture or $f#$ of the camera lens. It is well accepted that PIV noise increases with particle image diameter, and the results of this project should reflect this trend. Another purpose is to determine the minimum random and bias errors for DaVis and PRANA [20] for various sub-pixel displacements using a single particle image diameter from the data acquired.

3.1 Experimental Setup

Particle image data were acquired using typical PIV techniques. The experimental setup is shown in Fig. 3.1 with the camera field of view shown in red. A standard, 10-gallon aquarium was filled with water and Potters’ hollow glass microspheres with a particle diameter $d_P \approx 10 \mu m$ were added as seed. A DC utility pump was used to recirculate the seeded water through the aquarium. A garden hose was attached to the outlet of the pump. A jet was generated by adding a pipe elbow attached to the opposing end of the hose, threaded into an acrylic plate and fixed inside the aquarium. A garden hose was attached to the inlet of the pump with a flow splitting pipe attachment at the opposing end to allow the flow to be removed symmetrically from the tank. This suction divider was placed at the opposite end of the aquarium and close to the free surface of the water. The jet and suction divider were aligned in the same plane at the center of the tank so as to avoid particle movement through the laser plane. The Imager Intense camera from LaVision with a 105
mm Nikkor lens with $f#s$ ranging from 2.8 to 32 was placed on a 1-D Velmex traverse and aligned perpendicular to the laser sheet. Data were acquired at two distances from the laser sheet. An Nd:YAG laser pair from New Wave Research was used to illuminate the particles and DaVis from LaVision controlled image acquisition.

It was desired to acquire image pairs illuminated by each laser and different exposures, minimizing the time $dt$ between images. Acquiring images at $dt = 0 \, \mu s$ is outside the camera’s specifications. Therefore $dt = 4 \, \mu s$ was used for acquiring each image pair, which was small enough that the particles did not move between images. The particle image diameter was varied by changing the camera lens’ aperture or $f#$.

### 3.2 Data Acquisition

For each of 8 lens apertures listed in Table 3.1, 1000 image pairs were acquired. Increasing the lens $f#$ increases $d_r$, but also decreases the amount of light seen by the camera’s sensor. Therefore, to acquire similar images while adjusting the $f#$ requires that the laser intensity be adjusted as well. To ensure that the image pairs were as similar as possible when varying $d_r$, the laser intensity was adjusted until the first frame’s mean intensities were similar for all apertures. After ensuring the first frames matched, the second laser intensity was adjusted until the intensity RMS of the second frame closely matched the first frame’s intensity RMS. Adjusting the RMS of the second image to match the first causes the particles to be the same intensity “height” above the background noise in the images.
Table 3.1: The lens apertures or \( f# \) and aperture diameters \( D_a \) used in this study to vary the particle image diameter \( d_\tau \). The estimated \( d_\tau \) in pixels for the near-field \((M = 0.25)\) and far-field \((M = 0.53)\) cases are also shown.

<table>
<thead>
<tr>
<th>( f# )</th>
<th>( D_a (mm) )</th>
<th>( d_\tau ) (pixels, ( M=0.25 ))</th>
<th>( d_\tau ) (pixels, ( M=0.53 ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.8</td>
<td>37.50</td>
<td>2.545</td>
<td>1.696</td>
</tr>
<tr>
<td>4</td>
<td>26.25</td>
<td>2.871</td>
<td>1.908</td>
</tr>
<tr>
<td>5.6</td>
<td>18.75</td>
<td>3.007</td>
<td>1.969</td>
</tr>
<tr>
<td>8</td>
<td>13.13</td>
<td>3.066</td>
<td>2.197</td>
</tr>
<tr>
<td>11</td>
<td>9.55</td>
<td>3.100</td>
<td>2.303</td>
</tr>
<tr>
<td>16</td>
<td>6.56</td>
<td>3.506</td>
<td>2.506</td>
</tr>
<tr>
<td>22</td>
<td>4.77</td>
<td>4.022</td>
<td>3.162</td>
</tr>
<tr>
<td>32</td>
<td>3.28</td>
<td>5.386</td>
<td>4.198</td>
</tr>
</tbody>
</table>

3.3 PIV Noise Floor

The noise floor represents the lowest possible amount of PIV random errors and is the minimum uncertainty of a single PIV measurement. It can be calculated by taking a single particle image and shifting it a prescribed displacement, applying a cross-correlation, and comparing the calculated displacement to the known displacement of the image.

After acquiring all image data, a Matlab code was written to artificially shift the images [6]. Shifting the images by a sub-pixel amount requires that the images be resampled to a higher resolution image space, shifted, and then resampled back to the original image space. This process is described in detail in Section 3.4.

There are three ways to create image pairs with a specified sub-pixel displacement, namely: synthetic image pairs, pseudo image pairs, and shifted 0\( dt \) image pairs. There are many different algorithms for synthetic image generation and these methods will not be explored here.

3.3.1 Pseudo Image Pair Generation

The process of pseudo image generation involves using an actual, digital PIV image as the first image and then shifting it a prescribed amount in either the \( x \)- or \( y \)-direction to form the second image. The procedure involves resampling the image to a finer grid (upsampling), shifting the image a prescribed distance, and downsampling the image back
to the original grid. There are many methods used for upsampling. For example: the
nearest neighbor method, the bilinear method, and the bicubic method [6]. The methods
used in this study include a linear interpolation method and a nearest neighbor method.

There are few attempts at pseudo image generation as applied to PIV [4, 5]. The
method in [4] describes using a Matlab linear interpolation method called interp2() to
interpolate the sub-pixel intensities when upsampling the images. It was found that this
method attenuates the amplitude of the image’s frequency spectrum causing the particles
to appear smeared after shifting, for small $d_\tau$. Thus, the two and three side-lobed Lanczos
filters have been explored in addition to the method described in [4].

3.3.2 Shifted $0dt$ Image Pair Generation

A shifted $0dt$ image pair was described by McPhail et al. [5]. This method involves
acquiring a digital PIV image pair at $dt \approx 0\mu s$ apart and using a similar method as with
creating a pseudo image pair to shift the second image of the acquired pair.

3.4 Image Resampling

The method of image resampling includes upsampling and downsampling. As men-
tioned previously, upsampling involves taking an image and converting it to a finer or
higher resolution image space. Downsampling, in contrast, converts a higher resolution
image back to a more coarse image space which, in this study, is the original image space.

When a digital image is acquired, light strikes the camera sensor and each pixel of the
sensor averages a portion of the signal, creating a single pixel in the image. A defraction-
limited particle will reflect light in the form of a sinc function [7]. The analog signal is
smooth and continuous. However, because the camera sensor has limited resolution, the
acquired signal appears as step changes in intensity, as shown in Fig. 3.2. Upsampling a
digital image should convert a step signal into a signal that appears more like the original
analog signal. Downsampling then simulates the acquisition of the slightly shifted image
by averaging a specified portion of the upsampled image for each pixel in downsampled
space. Figure 3.2(a) contains a schematic of a particle as seen by the camera sensor before
Fig. 3.2: An example of digital acquisition of an analog particle image in the form of a sinc function. (a) Particle as seen by the camera. (b) The intensity profile for the particle. (c) The particle image as output by the camera sensor.

acquisition, while Fig. 3.2(b) contains the intensity profile of the particle. The Analog case represents the smooth, continuous intensity profile that is produced by the illuminated particle. The Sampled case represents the averaged sample that the digital camera sensor produces upon acquiring the image. Figure 3.2(c) contains an example of the particle as output by the camera sensor.

**Linear Interpolation**

The linear interpolation method is a simple upsampling method proposed by [4]. The method is applied using Matlab’s `interp2()` function which applies a linear fit between each point and its surrounding points in the image to upsample. The images were upsampled
arbitrarily by a factor of 9. Figure 3.6(a) shows the application of this method to a particle which is upsampled using the linear interpolation method, shifted, and downsampled using the linear interpolation method. The Original case shows the original pixel intensities in the original image space. The Upsampled case shows result of the Linear Interpolation technique with higher resolution of the original image. Shifted is the same as the Upsampled case but shifted 2/9 pixels to the right. The Downsamped data shows the average of the surrounding pixels of the shifted image the upsampled image space, which has been shifted 0.5 pixels to the right for viewing purposes.

Nearest Neighbor

In contrast to the linear interpolation technique, the nearest neighbor upsampling method simply replicates the pixel in original image space to the surrounding nodes in the upsampled image space. This is shown in Fig. 3.3, where the original particle image is shown on the left and each pixel in the original space is spread to the surrounding pixels in the upsampled space on the right of the image. The original space pixels are marked with a red dot • in both the original image space and the upsampled space for visual comparison. The original image space for each pixel has been marked with a dashed line to indicate what region is averaged when downsampling the image (discussed further in Section 3.4.2).

Figure 3.6(b) and 3.6(c) show the application of the two side-lobed Lanczos filter and the three side-lobed Lanczos filter, respectively. The images were upsampled using the nearest neighbor technique, filtered, shifted, then downsampled using the nearest neighbor technique. The Original case shows the original pixel intensities in the original image space. The Upsampled case shows result of the nearest neighbor upsampling technique with higher resolution of the original image and the two side-lobed Lanczos filter. Shifted is the same as the Upsampled case but shifted 2/9 pixels to the right. The Downsamped data shows the average of the surrounding pixels of the shifted image the upsampled image space, which has been shifted 0.5 pixels to the right for viewing purposes.

3.4.1 Filter Application
Fig. 3.3: An example of the nearest neighbor upsampling technique. The pixels marked with red dots represent the original pixels in both the original image space and upsampled image space. The upsampled scale of this example is 5 pixels. The original image pixel space is represented by the dashed line.

The technique employed by [4] results in an inaccurate description of the particle in the upsampled image space for small particle image diameters, resulting in smearing of the particles in the image after shifting. The three side-lobed Lanczos filter applies a very smooth transition between step changes in intensity, which results in a more accurate method of shifting the images by a sub-pixel displacement. The linear interpolation method and two side-lobed Lanczos filter both have very sharp changes in intensity after shifting.

Many different image processing filters are available both in Matlab’s image processing toolbox and other image processing software packages. According to [21, 22], the “sinc function is the ideal low-pass filter” for signal processing. The sinc function is defined as:

\[ \text{sinc}(x) = \frac{\sin(x)}{x} \]  

(3.1)

and is presented in Fig. 3.4. The difficulty in applying a sinc function filter is that it requires an infinite number of side-lobes to perfectly filter the images. Therefore, it was desirous to implement a windowed form of the sinc function that transitions to zero more
**Fig. 3.4**: Comparison of the sinc function and the Lanczos function with two side-lobes (Lanczos2) and three side-lobes (Lanczos3).

smoothly after a fixed number of side-lobes.

\[
\text{Lanczos}(x) = \begin{cases} 
\frac{\sin(\pi x) \sin(\frac{\pi x}{s})}{\pi x} & , |x| < s \\
0 & , |x| \geq s
\end{cases}
\]  

(3.2)

The Lanczos filter achieves this effect and allows specification of the number of side-lobes to include. The Lanczos function is represented by Eq. (3.2) where \(s\) represents the number of side-lobes to implement (\(s = 2\) or \(s = 3\) for this study). This equation results in a function shown in Fig. 3.4 for \(s = 2\) (Lanczos2) and \(s = 3\) (Lanczos3), which are compared to the standard sinc function.

**Lanczos Filter**

Applying the Lanczos function to the image using Matlab’s 2-D filter, `filter2()`, required the creation of a filter window. The window width was defined as the particle diameter rounded up to the next highest integer so as to include the entire particle in the window. Using the \((x,y)\) integer coordinates in the window, the Lanczos function was
applied, using a technique similar to that described in [21]. A contour of the 2-D Lanczos function with three side-lobes (Lanczos3) is shown in Fig. 3.5. Each location value in the window acts as a weight factor when applied as an image filter. In order to not change the image intensity these weights must add to unity [23], therefore the window was multiplied by the inverse sum of the original window weights. After creating the filter window, the upsampled image was filtered using \texttt{filter2()}, which rotates the filter 180 degrees then applies a 2-D convolution, \texttt{conv2()}, to filter the image [6].

3.4.2 Image Shifting and Downsampling

Upsampling allows the image to be shifted a known sub-pixel displacement before applying the cross-correlation. The amount of sub-pixel displacement is defined by the Eq. (3.3) with \( \tilde{x} \) being the particle displacement, \( S_x \) the number of pixels shifted in the upsampled image space, and \( U_s \) being the upsample scaling factor.

\[
\tilde{x} = \frac{S_x}{U_s} \tag{3.3}
\]
For this study, a single particle image displacement ($\bar{x} = \frac{2}{9} = 0.22$ pixels) was used with various particle image diameters, shown in Table 3.1. The images were upsampled arbitrarily by a factor of 9 and shifted using Matlab’s `circshift()` function. Particle image diameters were calculated using the algorithm provided in [24].

After shifting, the images were downsampled back to the original image space. For the linear interpolation case, the `interp2()` function was applied in reverse. For the nearest neighbor cases, the images were downsampled by averaging the pixels in each of the nearest neighbor regions shown in Fig. 3.3. The nearest neighbor regions remain fixed during the image shift, therefore slightly different pixels of the upsampled image space lie in each region.
Fig. 3.6: Comparison of a single particle’s intensity profile for the (a) linear interpolation, (b) two side-lobed Lanczos filter, and (c) three side-lobed Lanczos filter.
Chapter 4

Results

4.1 Particle Image Diameter

The particle diameters for each $f^\#$ used was determined using the Matlab code provided in Appendix A.2. The resulting particle diameters for both magnifications are shown in Table 3.1. The theoretical particle image diameter $d_\tau$ as shown in [1] by

$$d_\tau = \sqrt{(M \cdot d_P)^2 + (2.44 \cdot f^\# \cdot \lambda \cdot (M + 1))^2 + \left(\frac{M \cdot z \cdot D_a}{s_0 + z}\right)^2}$$

(4.1)

where $M$ is the magnification, $d_P$ is the physical particle diameter, $\lambda$ is the laser wavelength, $z$ is the object’s distance from the focal plane, $D_a$ is the aperture diameter, and $s_0$ is the object distance. The third term of Eq. (4.1) represents the effect of aberrations on the particle diameter when the particles do not lie in the focal plane of the camera and is neglect because the particles were well focused. Equation (4.1) simplifies to

$$d_\tau = \sqrt{(M \cdot d_P)^2 + (2.44 \cdot f^\# \cdot \lambda \cdot (M + 1))^2}$$

(4.2)

The theoretical $d_\tau$ was compared to the measured $d_\tau$ according to Eq. (4.2) for both the near- and far-field cases and is shown in Figs. 4.1(a) and 4.1(b).

4.2 Image Shifting

According to Scarano [25], a “suitable image interpolation method should avoid loss of information in the re-sampling process.” Thus, the spatial Fourier Transform (FT) of each image was used to determine the accuracy of their resample and shift. A perfect case would result in identical FTs for both images as well as an identical mean and RMS. No resample
Fig. 4.1: Comparison of the measured particle image diameter $d_\tau$ to the theoretical diameter calculated by Eq. (4.1) for (a) near-field case ($M=0.53$) and (b) far-field case ($M=0.25$).

and sub-pixel shift is perfect. Losses are inherent in techniques of this nature. Scarano [25] suggests that resampling methods may range from simple linear interpolation methods to more refined schemes such as a windowed sinc function. Scarano uses a truncated-sinc function by multiplying the sinc function by a Hamming or a Blackman window which is said to improve the frequency response of the method. It should be noted that, in this study, the linear interpolation case experiences frequency attenuation over the majority of the spectrum and the least frequency attenuation occurs in the Lanczos3 case.

Comparing the images before and after shifting, it is difficult to determine if the method was accurate in shifting the images. Figures 4.2(a), 4.3(a), and 4.4(a) contain a section of the original image, while 4.2(b), 4.3(b), and 4.4(b) contain the same section of the images but shifted 2/9 pixels to the right for the linear, Lanczos2, and Lanczos3 cases, respectively. Some smearing is evident in the Linear and Lanczos2 cases, indicating a loss of frequency information after the shift. These losses become evident when comparing the FTs of the images. Figure 4.5 shows the loss in amplitude of the FT after shifting for the Linear and Lanczos2 cases, demonstrating the smearing of particles. The Lanczos2 filter allowed higher
Fig. 4.2: Pseudo image pair result of linear interpolation case. (a) A portion of the original image, (b) the same portion of the image after shifting.

Fig. 4.3: Pseudo image pair result of two side-lobed Lanczos filter case. (a) A portion of the original image, (b) the same portion of the image after shifting.
frequencies to be transmitted through the upsample and shift but resulted in a change in curvature of the FT, which caused more smearing of the particles than the Linear case. In contrast, the Lanczos3 filter allowed nearly all frequencies present in the original image to be transmitted through the resample and shift, resulting in nearly identical FTs. A high frequency peak is present in all cases in Fig. 4.5 at a frequency of approximately 0.45. It was found that this peak represents the frequency corresponding to the background noise of the image and its amplitude is reduced as the seeding density increases.

The FTs of the images for comparison were calculated by averaging the FT of each line of the image. To ensure that an FT was not repeated for the same particle, the number of lines corresponding to twice the particle diameter were skipped between each FT. This average FT per image was then averaged for 150 images and is presented in Fig. 4.5.

The linear interpolation and Lanczos2 filter cases cause large enough frequency attenuation after resampling to result in less than ideal images after shifting and will not be considered in the remainder of the study. The Lanczos3 case, however, provided the least frequency attenuation and will therefore be the only case presented.
Fig. 4.5: Comparison of the FTs for each interpolation case to the FT of the original image.
4.3 Noise Floor

Convergence of the random error is necessary to accurately predict the noise floor of the cross-correlation. Each image contained $43 \times 32$ vectors ($N_{vi} = 1376$) after processing. It was found that the `circshift()` function used to shift the images cause the edges to have vectors with increased random error. Two rows of vectors were removed from each image resulting in $39 \times 28$ vectors ($N_{vi} = 1092$) per image. To determine the number of images needed to converge the noise floor, $N = 400$ images of one dataset were correlated and the $U$ velocity random error was calculated for $N = 1, 2, 3, ..., 400$ images. The number of vectors needed to converge the result was dramatically reduced by using post-processing the vector fields. As shown in Fig. 4.6, it was necessary to process at least 100 images in order to converge the random error with post-processed images. One hundred fifty images ($N = 150$) were included in the noise floor calculations ($N_v = 163, 800$ total vectors), ensuring a converged result.

After correlating all the shifted $0dt$ image pairs and pseudo image pairs, the random
error noise floor for the SCC was calculated. According to Coleman and Steele [13], the random error can be calculated be the following equation:

\[
S_\epsilon = \sqrt{\frac{1}{N_v - 1} \sum_{i=1}^{N_v} \epsilon^2}
\]  
(4.3)

where \(N_v\) is the number of vectors and \(\epsilon\) are the random errors of the individual measurements and are defined as

\[
\epsilon = x - \tilde{x} - \beta
\]  
(4.4)

with \(x\) being an individual velocity vector, \(\tilde{x}\) the prescribed displacement, and \(\beta\) the bias error defined as

\[
\beta = \frac{1}{N_v} \sum_{i=1}^{N_v} (x - \tilde{x}).
\]  
(4.5)

The results of the noise floor for \(U\) and \(V\) velocities for the shifted 0\(dt\) and pseudo image pairs are shown in Figs. 4.7(a) and 4.7(b). A previous study [26] showed that the linear interpolation method resulted in decreasing random error for an increase in particle diameter, contrary to the well-known result presented by Raffel [7]. It was unexpected that the entire pseudo image pair process would grossly underestimate random error and not depend on \(d_\tau\) entirely, as shown in Fig. 4.7(b). Due to the erroneous results provided by the pseudo image pair method, it will be not be considered in the remainder of this report. The results for the shifted 0\(dt\) image pairs presented in Fig. 4.7(a) more closely follow the trend presented by Raffel [7].

### 4.4 Subpixel Displacement

Previously, only one sub-pixel displacement was used. It was found that the various shifting methods resulted in approximately the same random error. It was therefore desirous to determine the random and bias errors for multiple sub-pixel displacements for only on \(d_\tau\). Multiple, sub-pixel displacements were considered and correlated using both DaVis and PRANA. Displacements of \(\tilde{x} = 0/9, 1/9, ..., 9/9\) pixels were investigated using the Lanczos3 filtering method applied to the shifted 0\(dt\) image pairs and a single particle diameter of
Particle Image Diameter, $d_{\tau}$
Random Error, [pixels]
$U$, Far-Field $V$, Far-Field
$U$, Near-Field $V$, Near-Field

(a)

(b)

Fig. 4.7: Random error for $U$ and $V$ velocities for far- and near-field cases with $\tilde{x} = 2/9$ pixels. (a) Random error for shifted $0dt$ image pairs. (b) Random error for pseudo image pairs.

$d_{\tau} \approx 2.5$ pixels ($f^\# = 16$ and $M = 0.25$). Random error, bias error, and total error were considered in these cases and are compared in Figs. 4.8(a), 4.8(b), and 4.8(c), respectively. The scale for the random error in Fig. 4.8(a) is different from that of the bias error in Fig. 4.8(b) to allow the trend of the random error to be visible.

Figure 4.8(a) shows that the random errors for both the DaVis and PRANA cross-correlation methods are similar in magnitude and trend for all sub-pixel displacements. The bias error is a strong function of displacement as shown in Fig. 4.8(b) while the random error is not. These show that the error of sub-pixel displacements are strongly influenced by bias error rather than random error. The total error for each case is shown in Fig. 4.8(c) and was calculated by the root sum square of the random and bias errors. The elevated bias error for all cases dramatically increases the total error of both SCC algorithms.
Fig. 4.8: Comparison of (a) random, (b) bias, and (c) total errors for a single $f^\#$ and multiple displacements between 0 and 1 pixel for the DaVis and PRANA SCC algorithms.
Chapter 5

Conclusion

An investigation of the minimum random uncertainty or *noise floor* of the standard cross-correlation (SCC) algorithm used in PIV measurements has been presented. Two methods for creating image pairs were explored, pseudo image pairs and shifted \(0 dt\) image pairs. Creation of a pseudo image pair involves shifting an image by a known displacement and correlating the original image with the shifted version of itself. Meanwhile, a shifted \(0 dt\) image pair involves acquiring an image pair with \(dt \approx 0\) s, shifting the second image in the pair and then applying the correlation. Two filtering methods (Lanczos2 and Lanczos3) were therefore applied in conjunction with the nearest neighbor sampling method and a linear interpolation method was compared. The three side-lobed Lanczos filter provided the least frequency attenuation after shifting the images.

Convergence of the random error required, based on this experimental setup, \(N = 150\) post-processed images for the noise floor to be determined. It was found that the pseudo image pair method inaccurately underestimates the random error and should not be used for determining the uncertainty of PIV correlation algorithms. The shifted \(0 dt\) image pair method, however, provided a trend similar to the well-known result presented by Raffel [7] and therefore is a viable approach for determining the noise floor. The number of available particle image diameters used in this study was limited by the number of camera positions and lens aperture diameters.

Time restraints did not allow further processing all particle diameters for multiple sub-pixel displacements, and therefore only a single sub-pixel displacement was investigated. This range of displacements would allow for wider application of the noise floor results. It was unexpected that the random error for all shifting methods would be nearly identical in magnitude for a given sub-pixel displacement, thus motivating the use of a range of
sub-pixel displacements for a single particle diameter and comparing the random and bias errors for the DaVis and PRANA SCC algorithms. Bias error was found to be a strong function of sub-pixel displacement, while the random error was fairly constant for all sub-pixel displacements considered between 0 and 1 pixels.

No attempt was made to identify the error sources. Identification of possible error sources and their impacts on the noise floor could greatly reduce the error of cross-correlations for sub-pixel displacements, and in turn increase the accuracy of PIV measurements.

The \texttt{circshift()} function in Matlab used to shift the upsampled images was found to induce major errors along the edges of the images being shifted, negatively impacting the random error of displacements in those regions. To remove these errors, two vector rows along each edge of the images were removed from the noise floor calculations. This implication is an inherent problem in creating shifted $0dt$ and pseudo image pairs and cannot be avoided.

The small number of the particle image diameters available in this study greatly limited the possible data to be examined. Further investigation is needed for smaller and larger particle image diameters than those used in this study. Synthetic images are a possible solution to the lack of image data in the ranges below and above the range used to create the shifted $0dt$ image pairs. This study also focused on 1-D particle shift. Particle shifts in 2-D would be useful for calculating the noise floor for different cross-correlation algorithms, and therefore merits further investigation.

Digital signal processing and digital image processing have brought rise to many different types of filters, requiring further study on the impacts of other filters on the creation of shifted $0dt$ image pairs.
References


Appendices
Appendix A
Matlab Image Shifting Code

A.1 Image Shifting Driver

```matlab
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Driver_ImageShift.m
% % DESCRIPTION:
% %
% % DEPENDENCIES:
% % Image Processing Toolbox
% % Diameter_Check.m
% % LanczosFilter.m
% % LinearFilter.m
% %
% % DATA NEEDED:
% %
% % VERSION INFORMATION:
% % Number: Programmer: Changes:
% % v1.0 K. Jones Original
% %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all; close all;

startt = tic;
hours = 1/(60*60);

% Path Info
project = 'dt0_Wide_Align_Far';
noiseflr = [pwd,'\',project,'\'];
foldr = {'dt0_f028';
         'dt0_f04';
         'dt0_f056';
         'dt0_f08';
         'dt0_f11';
         'dt0_f16';
         'dt0_f22';
         'dt0_f32'};
subfold1 = 'Export';
subfold2 = 'Export_01';

% Contants
ext = 'bmp'; % Image extension to write to
extin = 'tif'; % Image extension of input images
nims = 1000; % Total number of images
```
nmin = 1; %Starting image number to analyze
nmax = 1; %Ending image number to analyze
nspace = 1; %Spacing of images to analyze
scale = 8; %Upsample size (final image is size width*(scale+1))
scale2 = scale/2; %Half upsample size
scale1 = scale+1; %Upsample size + 1
sd = 2; %Pixel displacement wrt the scale up

%Flags
send = 1; % 1=Sends an email when the analysis is complete. ...
  Prompts user for username and password (Gmail only)
shift1d = 0; %1d shift direction 0=xshift, 1=yshift
shift2d = 0; %2d shift 0=use 1d shift above, 1=symmetric shift in ...
  x and y
lncz2 = 1; %0=Don't use Lanczos2 filter, 1=Use Lanczos2 filter
lncz3 = 1; %0=Don't use Lanczos3 filter, 1=Use Lanczos3 filter
linear = 1; %0=Don't use Linear interpolation, 1=Use Linear ...
  Interpolation
FFTs = 0; %0=Don't calculate FFT's of images, 1=Calculate FFT's of ...
  images
parallel = 1; %Parallelize code
profiler = 0; %Flag to use profiler
pseudo = 1; %0=Artificially shift second image in pair, ...
  1=Artificially shift first image in pair

if pseudo==0
  outfold = 'ArtificialImagePair';
else
  outfold = 'PseudoImagePair';
end

%Input E-Mail Username and Password
if send
  [Username,Password,send] = EmailLogin;
end

if shift1d==0 && shift2d==0
  fprintf(
[images will be shifted ',num2str(sd/scall),...
  ' pixels in the x-direction \r']);
elseif shift1d==1 && shift2d==0
  fprintf([Images will be shifted ',num2str(sd/scall),...
  ' pixels in the y-direction \r']);
elseif shift1d==1
  fprintf([Images will be shifted ',num2str(sd/scall),...
  ' pixels in both x- and y-directions \r']);
end

if parallel; matlabpool local 12; end

%Folder Loop
for f=1:size(foldr,1)
  inner = tic;
  if pseudo==1
    fprintf([Starting pseudo shift of ',project,', ',foldr{f}', ' dataset ... \']);
    wd = charcat(noisefilr,foldr{f},'\'); %Path to current ...
    data folder 
  end
im = strcat(noiseflr,foldr{f,'\',subfold1,'\'}); % Path to images to analyze
wdpath = sprintf('%s',wd); % Convert cell to string
impath = sprintf('%s',im); % Convert cell to string
Diameter_Check(wdpath,impath,extin,0); % Calculate particle image diameter
dp = importdata([wdpath,'ParticleDiameter.dat']); % Get particle image diameter from file
winwd = round(dp*2); % Window width in pixels in original space

% Image Loop
m = nmin:nspac:2*nmax;
parfor n=nmin:nspac:nmax
  imname = [impath,'B',num2str(n,'%5.5d.'),extin];
  fprintf(['Image B',num2str(n,'%5.5d'), ' of dataset ',foldr{f,...
    ' Applied'});
  if profiler; profile on; end
  if lncz2==1
    type = 2;
    if shift1d==0; sx = sd; sy = 0;
      imwr1 = [wdpath,outfold,'\xshift\Lncz',num2str(type),...
        '\B',num2str((2*n-1),'%5.5d.'),ext];
      imwr2 = [wdpath,outfold,'\xshift\Lncz',num2str(type),...
        '\B',num2str((2*n),'%5.5d.'),ext];
    elseif shift1d==1; sx = 0; sy = sd;
      imwr1 = [wdpath,outfold,'\yshift\Lncz',num2str(type),...
        '\B',num2str((2*n-1),'%5.5d.'),ext];
      imwr2 = [wdpath,outfold,'\yshift\Lncz',num2str(type),...
        '\B',num2str((2*n),'%5.5d.'),ext];
    end
  elseif shift1d==1; sx = 0; sy = sd;
    imwr1 = [wdpath,outfold,'\xyshift\Lncz',num2str(type),...
      '\B',num2str((2*n-1),'%5.5d.'),ext];
    imwr2 = [wdpath,outfold,'\xyshift\Lncz',num2str(type),...
      '\B',num2str((2*n),'%5.5d.'),ext];
  end
  [~,~,~] = LanczosFilter(type,imname,imname,imwr1,imwr2,scale,...
    sx,sy,winwd,FFTs);
  fprintf(' Lanczos2');
  if profiler; profile off; end
  if lncz3==1
    type = 3;
    if shift1d==0; sx = sd; sy = 0;
      imwr1 = [wdpath,outfold,'\xshift\Lncz',num2str(type),...
        '\B',num2str((2*n-1),'%5.5d.'),ext];
      imwr2 = [wdpath,outfold,'\xshift\Lncz',num2str(type),...
        '\B',num2str((2*n),'%5.5d.'),ext];
    elseif shift1d==1; sx = 0; sy = sd;
      imwr1 = [wdpath,outfold,'\yshift\Lncz',num2str(type),...
        '\B',num2str((2*n-1),'%5.5d.'),ext];
      imwr2 = [wdpath,outfold,'\yshift\Lncz',num2str(type),...
        '\B',num2str((2*n),'%5.5d.'),ext];
    end
  elseif shift1d==1; sx = 0; sy = sd;
    imwr1 = [wdpath,outfold,'\xyshift\Lncz',num2str(type),...
      '\B',num2str((2*n-1),'%5.5d.'),ext];
    imwr2 = [wdpath,outfold,'\xyshift\Lncz',num2str(type),...
      '\B',num2str((2*n),'%5.5d.'),ext];
  end
if shift2d==1; sx = sd; sy = sd;
imwr1 = [wdpath,outfold,'\xyshift\Lncz',num2str(type),'
\B',num2str((2*n-1),'%5.5d.'),ext];
imwr2 = [wdpath,outfold,'\xyshift\Lncz',num2str(type),'
\B',num2str((2*n),'%5.5d.'),ext];
end
["","","] = LanczosFilter(type,imname,imname,imwr1,imwr2,scale,...
x,x,winwd,FFTs);
fprintf(', Lanczos3');

end
if lnear==1
  if shift1d==0; sx = sd; sy = 0;
    imwr1 = ...
      [wdpath,outfold,'\xshift\Lnear\B',num2str((2*n-1),...
      '%5.5d.'),ext];
imwr2 = [wdpath,outfold,'\xshift\Lnear\B',num2str((2*n),...
      '%5.5d.'),ext];
  elseif shift1d==1; sx = 0; sy = sd;
    imwr1 = ...
      [wdpath,outfold,'\yshift\Lnear\B',num2str((2*n-1),...
      '%5.5d.'),ext];
imwr2 = [wdpath,outfold,'\yshift\Lnear\B',num2str((2*n),...
      '%5.5d.'),ext];
  end
  if shift2d==1; sx = sd; sy = sd;
    imwr1 = ...
      [wdpath,outfold,'\xyshift\Lnear\B',num2str((2*n-1),...
      '%5.5d.'),ext];
imwr2 = [wdpath,outfold,'\xyshift\Lnear\B',num2str((2*n),...
      '%5.5d.'),ext];
end
["","","] = LinearFilter(type,imname,imname,imwr1,imwr2,scale,...
x,x,winwd,FFTs);
fprintf(', Linear');
end
fprintf(' \
');
else pseudo==0
  fprintf(['Starting artificial shift of ',project,', ',foldr{f},...
  'dataset \r']);
  wd = strcat(noiseflr,foldr{f},''); %Path to current ...
data folder
  im1 = strcat(noiseflr,foldr{f},'\',subfold1,''); %Path to images ...
to analyze
  im2 = strcat(noiseflr,foldr{f},'\',subfold2,''); %Path to images ...
to analyze
  wdpwth = sprintf('s',wd); %Convert cell to ...
  string
  impath1 = sprintf('s',im1); %Convert cell to ...
  string
  impath2 = sprintf('s',im2); %Convert cell to ...
  string
  Diameter_Che impeachment(path,impath1,extin,0); %Calculate ...
particle image diameter
  dp = importdata([wdpath,'\ParticleDiameter.dat']); %Get particle ...
  image diameter from file
```matlab
winwd = round(dp*2); % Window width in pixels in original space
%
% Image Loop
m = nmin:nspac:2*nmax;
parfor n=nmin:nspac:nmax
imname1 = [impath1,'B',num2str(n,'%5.5d.'),extin];
imname2 = [impath2,'B',num2str(n,'%5.5d.'),extin];
fprintf(['Image B',num2str(n,'%5.5d'),' of dataset ',foldrf{f}, ' Applied']);
if profiler; profile on; end
if lncz2==1
    type = 2;
    if shift1d==0; sx = sd; sy = 0;
        imwr1 = [wdpath,outfold,'\xshift\Lncz',num2str(type),... 
            '\B',num2str((2*n-1),'%5.5d.'),ext];
        imwr2 = [wdpath,outfold,'\xshift\Lncz',num2str(type),... 
            '\B',num2str((2*n),'%5.5d.'),ext];
    elseif shift1d==1; sx = 0; sy = sd;
        imwr1 = [wdpath,outfold,'\yshift\Lncz',num2str(type),... 
            '\B',num2str((2*n-1),'%5.5d.'),ext];
        imwr2 = [wdpath,outfold,'\yshift\Lncz',num2str(type),... 
            '\B',num2str((2*n),'%5.5d.'),ext];
    end
    if shift2d==1; sx = sd; sy = sd;
        imwr1 = [wdpath,outfold,'\xyshift\Lncz',num2str(type),... 
            '\B',num2str((2*n-1),'%5.5d.'),ext];
        imwr2 = [wdpath,outfold,'\xyshift\Lncz',num2str(type),... 
            '\B',num2str((2*n),'%5.5d.'),ext];
    end
end[~,~,~] = ... 
    LanczosFilter(type,imname1,imname2,imwr1,imwr2,scale,... 
        sx,sy,winwd,FFTs);
fprintf(' Lanczos2');
if profiler; profile off; end
end
if lncz3==1
    type = 3;
    if shift1d==0; sx = sd; sy = 0;
        imwr1 = [wdpath,outfold,'\xshift\Lncz',num2str(type),... 
            '\B',num2str((2*n-1),'%5.5d.'),ext];
        imwr2 = [wdpath,outfold,'\xshift\Lncz',num2str(type),... 
            '\B',num2str((2*n),'%5.5d.'),ext];
    elseif shift1d==1; sx = 0; sy = sd;
        imwr1 = [wdpath,outfold,'\yshift\Lncz',num2str(type),... 
            '\B',num2str((2*n-1),'%5.5d.'),ext];
        imwr2 = [wdpath,outfold,'\yshift\Lncz',num2str(type),... 
            '\B',num2str((2*n),'%5.5d.'),ext];
    end
    if shift2d==1; sx = sd; sy = sd;
        imwr1 = [wdpath,outfold,'\xyshift\Lncz',num2str(type),... 
            '\B',num2str((2*n-1),'%5.5d.'),ext];
        imwr2 = [wdpath,outfold,'\xyshift\Lncz',num2str(type),... 
            '\B',num2str((2*n),'%5.5d.'),ext];
    end
end[~,~,~] = ... 
    LanczosFilter(type,imname1,imname2,imwr1,imwr2,scale,...
```
sx,sy,winwd,FFTs);

fprintf(', Lanczos3');
end

if lnear==1
    if shift1d==0; sx = sd; sy = 0;
        imwr1 = ...
        [wdpath,outfold,'\xshift\Lnear\B',num2str((2*n-1),...
         ' %5.5d.'),ext];
        imwr2 = [wdpath,outfold,'\xshift\Lnear\B',num2str((2*n),...
         ' %5.5d.'),ext];
    elseif shift1d==1; sx = 0; sy = sd;
        imwr1 = ...
        [wdpath,outfold,'\yshift\Lnear\B',num2str((2*n-1),...
         ' %5.5d.'),ext];
        imwr2 = [wdpath,outfold,'\yshift\Lnear\B',num2str((2*n),...
         ' %5.5d.'),ext];
    end
    if shift2d==1; sx = sd; sy = sd;
        imwr1 = ...
        [wdpath,outfold,'\xyshift\Lnear\B',num2str((2*n-1),...
         ' %5.5d.'),ext];
        imwr2 = [wdpath,outfold,'\xyshift\Lnear\B',num2str((2*n),...
         ' %5.5d.'),ext];
    end
    [~,~] = ...
    LinearFilter(type,imname1,imname2,imwr1,imwr2,scale,...
    sx,sy,winwd,FFTs);
    fprintf(', Linear');
end
end

tloop = toc(inner); tloop = tloop*hours;
if send
    %Send Email
    setpref('Internet','E_mail',Username);
    setpref('Internet','SMTP_Server','smtp.gmail.com');
    setpref('Internet','SMTP_Username',Username);
    setpref('Internet','SMTP_Password',Password);
    props = java.lang.System.getProperties;
    props.setProperty('mail.smtp.auth','true');
    props.setProperty('mail.smtp.socketFactory.class', ...
        'javax.net.ssl.SSLSocketFactory');
    props.setProperty('mail.smtp.socketFactory.port','465');
    sendmail(Username, 'Dataset Complete',['Completed processing of ...
    dataset ','foldr{f},...
    ' of ',noiseflr,' in ',num2str(tloop),' hours']);
end
if profiler; profile viewer; end
if parallel; matlabpool close; end

diff = toc(startt);
diff = diff*hours;
fprintf(['Elapsed time is ',num2str(diff),' hours.']);
if send
    %Send Email
    setpref('Internet','E_mail',Username);
    setpref('Internet','SMTP_Server','smtp.gmail.com');
    setpref('Internet','SMTP_Username',Username);
    setpref('Internet','SMTP_Password',Password);
    props = java.lang.System.getProperties;
    props.setProperty('mail.smtp.auth','true');
    props.setProperty('mail.smtp.socketFactory.class', ...
                      'javax.net.ssl.SSLSocketFactory');
    props.setProperty('mail.smtp.socketFactory.port','465');
    sendmail(Username,'Processing Complete',
             ['Completed entire set of '...
              noiseflr,' in ',num2str(endt),' hours']);
    clear Username; clear Password;
end

A.2 Image Shifting Functions

Particle Diameter Function

function Diameter_Check(savepath,impath,ext,DaVis)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Diameter Estimation %
% − Enter in file name below %
% − Estimate at Center of Image %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% close all; clear all;

%DaVis File
if DaVis==1
davis=loadvec([impath,'B00001.im7']);
else
    %Any other image File type (png, jpg, bmp, etc.)
davis.w = double(imread([impath,'B00001.',ext]));
end
Nx=256;
x=zeros([DI DI]);
y=zeros([DI DI]);
for j=1:DI
    x(:,j)=linspace(-DI/2,DI/2-1,DI);
y(1:DI,j)=linspace(DI/2,DI/2-1,DI);
end
% (center)
img=davis.w(nx-DI/2+1:nx+DI/2,ny-DI/2+1:ny+DI/2);
X=nx-DI/2+1:nx+DI/2;
Y = ny - DI/2 + 1:ny + DI/2;
img = flipud(img');
img = img - min(min(img));

% Autocorrelate
A = fftn(img);
C2 = real(ifftn(conj(A) .* A));

% Correlation Peak at Center of image
C2(1:DI/2, 1:DI/2) = rot90(C2(1:DI/2, 1:DI/2), 2);
C2(1:DI/2, DI/2 + 1:DI) = rot90(C2(1:DI/2, DI/2 + 1:DI), 2);
C2(DI/2 + 1:DI, 1:DI/2) = rot90(C2(DI/2 + 1:DI, 1:DI/2), 2);
C2(DI/2 + 1:DI, DI/2 + 1:DI) = rot90(C2(DI/2 + 1:DI, DI/2 + 1:DI), 2);

% Locate Correlation Maximum
[ymaxval, yind] = max(C2, [], 1);
[~, xind] = max(ymaxval);
yind = yind(xind);
x = 0.5:size(C2, 1) - 0.5;

% Number of points used in Gauss Fit
val = 21;
xval = x(yind - val: yind + val);
Cval = C2(yind - val: yind + val, xind);

% In the Y direction
y = 0.5:size(C2, 2) - 0.5;
yval = y(xind - val: xind + val);
Cvaly = C2(yind, xind - val: xind + val);
Cvaly = Cvaly - min(Cvaly);

% [xData2, yData2] = prepareCurveData( yval, Cvaly ); % Matlab R2011+ only
xData2 = yval'; yData2 = Cvaly'; % Matlab R2010 and earlier
[fitresult2, ~] = fit( xData2, yData2, 'gauss1');
dia_y = 2*fitresult2.c1;

% Subtract lowest value
Cval = Cval - min(Cval);

% Gauss Fit
% [xData, yData] = prepareCurveData( xval, Cval ); % Matlab R2011+ only
xData = xval'; yData = Cval; % Matlab R2010 and earlier
[fitresult, ~] = fit( xData, yData, 'gauss1');
dia = 2*fitresult.c1;

fid = fopen([savepath, 'ParticleDiameter.dat'], 'w');
fprintf(fid, num2str(dia, '%e'));
fclose(fid);

Linear Interpolation Function

function [wave, fftmo, fftmd] = ...
    LinearFilter(type, imname1, imname2, imwr1, imwr2, scale, sx, sy, winwd, FFTs)
scale = scale+1; %Upsample size + 1

%Read in first image
im = imread(imname1);
w = size(im,2);
h = size(im,1);
if size(im,3)>1
    im = rgb2gray(im);
end
imwrite(uint8(im),imwr1);

if FFTs
    %FFT of original image
    mfft = nextpow2(w);
    if 2^mfft>w; mfft = mfft-1; end
    nfft = 2^mfft;
    wave = zeros(nfft/2,1);
    for j=2:nfft/2-1; wave(j) = wave(j-1)+1/nfft; end
    wave(1,1) = 10^floor(log10(wave(2)));
    m = 0;
    for j=1:round(dp):h
        m = m+1;
        ffto(m,:) = ...
            sffteu(real(double(im(j,:))),imag(double(im(j,:))),nfft,mfft,-1);
    end
    fftmo = mean(abs(ffto),1)';
else
    wave = 0;
    fftmo = 0;
end

%Read in second image
im = imread(imname2);
w = size(im,2);
h = size(im,1);
if size(im,3)>1
    im = rgb2gray(im);
end

if sx==0 && sy==0
    imd = im;
    imwrite(uint8(imd),imwr2);
    fftmd = 0;
else
    %Upsample the image
    xstep = 1/(scale+1);
ystep = 1/(scale+1);
    xi = 1:xstep:w;
yi = 1:ystep:h;
    [x,y] = meshgrid(1:w,1:h);
    [xia,yia] = meshgrid(xi,yi);
    imu = interp2(x,y,double(im),xia,yia);

    %Apply filter
    imf = imu;
Lanczos Filter Function

```matlab
function [wave,fftmo,fftmd] = LanczosFilter(type,imname1,imname2,imwr1,imwr2,scale,sx,sy,winwd,FFTs)
scal2 = scale/2; %Half upsample size
scal1 = scale+1; %Upsample size + 1
W = 2*type; %Window width
winsz = round(winwd*(scal1)); %Window size
if mod(winsz,2)==1
    winsz = winsz+1; %Make sure winsz is odd
end
%Read in first image
im = imread(imname1);
[w,h] = size(im,2);
if size(im,3)>1
    im = rgb2gray(im);
end
imwrite(uint8(im),imwr1);
if FFTs
    %FFT of original image
    mfft = nextpow2(w);
    if 2^mfft>w; mfft = mfft-1; end
    nfft = 2^mfft;
    wave = zeros(nfft/2,1);
    for j=2:nfft/2-1; wave(j) = wave(j-1)+1/nfft; end
    wave(1,1) = 10^floor(log10(wave(2))); h = m+1;
```
fft0(m,:) = ... 
    sffteu(real(double(im(j,:))),imag(double(im(j,:))),nfft,mfft,−1);
end
fft0 = mean(abs(fft0),1)';
else
    wave = 0;
    fft0 = 0;
end

%Read in second image
im = imread(imname2);
w = size(im,2);
h = size(im,1);
if size(im,3)>1
    im = rgb2gray(im);
end

if sx==0 && sy==0
    imd = im;
imwrite(uint8(imd),imwr2);
    fftmd = 0;
else
    %Upsample image
    imu = uint8(zeros(size(im,1)*(scal1),size(im,2)*(scal1)));
    imu(scal2+1:scal1:end,scal2+1:scal1:end) = im;
    for j=1:h
        for i=1:w
            m = i*scale+i−scal2;
            n = j*scale+j−scal2;
            imu(n−scal2:n+scal2,m−scal2:m+scal2) = im(j,i);
        end
    end

    x = −type:W/winsz:type;
    y = −type:W/winsz:type;

    %Create Lanczos filter
    for j=1:length(y)
        for i=1:length(x)
            mag = sqrt(x(i)^2+y(j)^2);
            if mag<type
                lnczf(i,j) = sin(pi*mag)*sin(pi*mag/type)/(pi^2*mag^2/type);
            else
                lnczf(i,j) = 0;
            end
        end
    end
    lnczf((length(x)+1)/2,(length(y)+1)/2) = 1;
    lnczf = 1/sum(sum(lnczf)).*lnczf;

    %Apply Lanczos filter
    imf = filter2(lnczf,imu,'same');

    %Shift image
    ims = circshift(imf,[sy,sx]);

    %Downsample image
for j=1:h
    for i=1:w
        m = i*scale+i-scal2;
        n = j*scale+j-scal2;
        imd(j,i) = mean2(ims(n-scal2:n+scal2,m-scal2:m+scal2));
    end
end

%Write Shifted Image
imwrite(uint8(imd),imwr2);
end

if FFTs
    m = 0;
    for j=1:round(dp):h
        m = m+1;
        fftd(m,:) = sffteu(real(imd(j,:)),imag(imd(j,:)),nfft,mfft,-1);
    end
    fftmd(:,1) = mean(abs(fftd),1);
else
    fftmd = 0;
end

Fast Fourier Transform Function

function [x,y] = sffteu(x,y,n,m,itype)
%
% Subroutine sffteu( x, y, n, m, itype )
% This routine is a slight modification of a complex split
% radix FFT routine presented by C.S. Burrus. The original
% program header is shown below.
%
% Arguments:
% x - real array containing real parts of transform
% sequence (in/out)
% y - real array containing imag parts of transform
% sequence (in/out)
% n - integer length of transform (in)
% m - integer such that n = 2**m (in)
% itype - integer job specifier (in)
%   itype .ne. -1 --> forward transform
%   itype .eq. -1 --> backward transform
%
% The forward transform computes
% X(k) = sum_{j=0}^{N-1} x(j)*exp(-2*ijk*pi/N)
% The backward transform computes
% x(j) = (1/N) * sum_{k=0}^{N-1} X(k)*exp(2*ijk*pi/N)

if n==1
  return
end

if itype==−1
  for i=1:n
    y(i) = −y(i);
  end
end

n2 = 2*n;
for k=1:n−1
  n2 = n2/2;
  n4 = n2/4;
  e = 2*pi/n2;
  a = 0.0;
  for j=1:n4
    a3 = 3*a;
    cc1 = cos(a);
    ss1 = sin(a);
    cc3 = cos(a3);
    ss3 = sin(a3);
    a = j*e;
    is = j;
    id = 2*n2;
    flag1 = 1;
    while flag1
      i0 = is:id:n−1
      i1 = i0 + n4;
      i2 = i1 + n4;
      i3 = i2 + n4;
      r1 = x(i0) − x(i2);
      x(i0) = x(i0) + x(i2);
      r2 = x(i1) − x(i3);
      x(i1) = x(i1) + x(i3);
      s1 = y(i0) − y(i2);
      y(i0) = y(i0) + y(i2);
      s2 = y(i1) − y(i3);
      y(i1) = y(i1) + y(i3);
      s3 = r1 − s2;
    end
  end
end
91  end
92  is = 2*id-n2+j;
93  id = 4*id;
94  if is>n; flag1 = 0; end
95  end
96  end
97
98
99  %------Last Stage, Length-2 Butterfly
100  is = 1;
101  id = 4;
102  flag2 = 1;
103  while flag2
104    for i0=is:id:n
105      i1 = i0 + 1;
106      r1 = x(i0);
107      x(i0) = r1 + x(i1);
108      x(i1) = r1 - x(i1);
109      r1 = y(i0);
110      y(i0) = r1 + y(i1);
111      y(i1) = r1 - y(i1);
112    end
113    is = 2*id-1;
114    id = 4*id;
115    if is>n; flag2 = 0; end
116  end
117
118  %------Bit Reverse Counter
119  j=1;
120  n1 = n-1;
121  for i=1:n1
122    if i<j
123      xt = x(j);
124      x(j) = x(i);
125      x(i) = xt;
126      xt = y(j);
127      y(j) = y(i);
128      y(i) = xt;
129    end
130    k = n/2;
131    flag3 = 1;
132    while flag3
133      if k>=j; break; end
134      j = j-k;
135      k = k/2;
136    end
137    j = j+k;
138  end
if itype==−1
    x = x./n;
    y = −y./n;
end
Appendix B

Matlab Vector Analysis Code

B.1 Get Processed Vectors Code

```matlab
% MATLAB Code for processing vectors

% Initialize variables
clear all; close all;
warning off all;
startt = tic;

% Set parameters
nims = 150;
camera = 'Far';
datapath = [pwd,'\dt0_Wide_Align\',camera, '_Proc_01'];
partdat = importdata([pwd,'\dt0_Wide_Align\',camera, 'Particle_FNumber.txt']);
fstop = importdata([pwd,'\dt0_Wide_Align\',camera, 'F_Stops.txt']);
pseudo = {'AIP','PIP'};
filt = {'Lncz2';'Lncz3';'Linear'};
subfold = 'CreateMultiframe\PIV_MP(64x64_50%ov)\PostProc';
imname = 'B00001.VC7';

% Load vector files
fprintf('Begin loading processed vector data...\r');
for i=1:length(fstop)
    for p=1:length(pseudo)
        foldr = [fstop{i},'.',pseudo{p},'.'];
        f = 1;
        %Read Lanczos2 Images
        for n=1:nims
            imname = ['B',num2str(n,'%5.5d'),'.VC7'];
            path = sprintf('%s',[datapath,'\',foldr,filt{f},...
                               '\',subfold,imname]);
            particle(i,p).lncz2(n) = loadvec(path);
            fprintf(['Dataset ',foldr,filt{f},...
                     ' Image ',imname,' loaded... \r']);
        end
        f = 2;
        %Read Lanczos3 Images
        for n=1:nims
            imname = ['B',num2str(n,'%5.5d'),'.VC7'];
            path = sprintf('%s',[datapath,'\',foldr,filt{f},...
                               '\',subfold,imname]);
            particle(i,p).lncz3(n) = loadvec(path);
            fprintf(['Dataset ',foldr,filt{f},...
                     ' Image ',imname,' loaded... \r']);
        end
        f = 3;
        %Read Linear Images
```
for n=1:nims
    imname = ['B',num2str(n,'%5.5d'),'.VC7'];
    path = sprintf('%s', [datapath,'\',foldr,filt{f},...
    '\\',subfold,imname]);
    particle(i,p).lnear(n) = loadvec(path);
    fprintf(['Dataset ',foldr,filt{f},...
    ' Image ',imname,' loaded...
']);
end
end
end
fprintf('Completed loading processed vector data...
');
%% %Save data
savest = tic;
savedir = [datapath, '\ProcessedData\'];
savefile = ['Processed',num2str(nims),'.Images.',num2str(length(filt)),...
'Filters'];
save([savedir,savefile]);
savetend = toc(savest)/60;
fprintf(['Data saved in ',num2str(savetend),' minutes!
']);
endt = toc(startt)/60;
fprintf(['Elapsed time is ',num2str(endt),' minutes.
']);

B.2 Compute Noise Floor Code

clear all; close all;
warning off all;
% startt = tic;

improc = 150;
sigma = 1.96;
shift = 2;
upsamp = 9;
nims = 400;
camera = 'Far';
datpath = [pwd,'\dt0_Wide_Alg',camera];
varpath = [pwd,'\dt0_Wide_Alg',camera,'.Proc'];
%Load mat file with vector data
loadt = tic;
load([varpath,'\ProcessedData\Processed',num2str(nims),'.Images,3Filters.mat']);
loadtime = toc(loadt);
fprintf(['Loaded *.mat file in ',num2str(loadtime),' seconds.
']);

%Calculate the number of vectors per image
rmvec = 2;
xmin = rmvec+1;
xmax = size(particle(1,1).lnear(1).vx,1)-rmvec;
 ymin = rmvec+1;
ymax = size(particle(1,1).lnear(1).vx,2)-rmvec;
Nvi = (xmax-xmin+1)*(ymax-ymin+1);
Nv = Nvi*improc;
%
for p=1:length(pseudo)
pt = tic;
displx = shift/upsamp;
disply = 0;
for i=1:length(fstop)
%Collect Lanczos2 Data
for n=1:improc
if n==1
    vx2 = abs(particle(i,p).lncz2(n).vx(xmin:xmax,ymin:ymax));
    vy2 = particle(i,p).lncz2(n).vy(xmin:xmax,ymin:ymax);
else
    vx2 = [vx2,abs(particle(i,p).lncz2(n).vx(xmin:xmax,ymin:ymax))];
    vy2 = [vy2,particle(i,p).lncz2(n).vy(xmin:xmax,ymin:ymax)];
end
Meanx2(i,p) = mean2(vx2−displx);
Meany2(i,p) = mean2(vy2−disply);
Biasx2(i,p) = sum(sum(vx2−displx))/Nv;
Biasy2(i,p) = sum(sum(vy2−disply))/Nv;
epsx2 = vx2−displx−Biasx2(i,p);
epsy2 = vy2−disply−Biasy2(i,p);
RMSx2(i,p) = sqrt(sum(sum(epsx2.ˆ2))/Nv);
RMSy2(i,p) = sqrt(sum(sum(epsy2.ˆ2))/Nv);
fprintf(['Calculated all vectors for ',fstop{i},' Lanczos2 Filter, ...
',pseud{p},'']);
end
%Collect Lanczos3 Data
for n=1:improc
if n==1
    vx3 = abs(particle(i,p).lncz3(n).vx(xmin:xmax,ymin:ymax));
    vy3 = particle(i,p).lncz3(n).vy(xmin:xmax,ymin:ymax);
else
    vx3 = [vx3,abs(particle(i,p).lncz3(n).vx(xmin:xmax,ymin:ymax))];
    vy3 = [vy3,particle(i,p).lncz3(n).vy(xmin:xmax,ymin:ymax)];
end
Meanx3(i,p) = mean2(vx3−displx);
Meany3(i,p) = mean2(vy3−disply);
Biasx3(i,p) = sum(sum(vx3−displx))/Nv;
Biasy3(i,p) = sum(sum(vy3−disply))/Nv;
epsx3 = vx3−displx−Biasx3(i,p);
epsy3 = vy3−disply−Biasy3(i,p);
RMSx3(i,p) = sqrt(sum(sum(epsx3.ˆ2))/Nv);
RMSy3(i,p) = sqrt(sum(sum(epsy3.ˆ2))/Nv);
fprintf(['Calculated all vectors for ',fstop{i},' Lanczos3 Filter, ...
',pseud{p},'']);
end
%Collect Linear Data
for n=1:improc
if n==1
    vx1 = abs(particle(i,p).lnear(n).vx(xmin:xmax,ymin:ymax));
    vy1 = particle(i,p).lnear(n).vy(xmin:xmax,ymin:ymax);
else
    vx1 = [vx1,abs(particle(i,p).lnear(n).vx(xmin:xmax,ymin:ymax))];
    vy1 = [vy1,particle(i,p).lnear(n).vy(xmin:xmax,ymin:ymax)];
end
Meanxl(i,p) = mean2(vxl-displx);
Meanyl(i,p) = mean2(vyl-disply);
Biasxl(i,p) = sum(sum(vxl-displx))/Nv;
Biasyl(i,p) = sum(sum(vyl-disply))/Nv;
epsxl = vxl-displx-Biasxl(i,p);
epsyl = vyl-disply-Biasyl(i,p);
RMSxl(i,p) = sqrt(sum(sum(epsxl.^2))/Nv);
RMSyl(i,p) = sqrt(sum(sum(epsyl.^2))/Nv);
fprintf(['Calculated all vectors for ',fstop{i},' Linear Filter, ...
',pseudo{p},'r']);

end
ptime = toc(pt);
fprintf(['Completed compiling data for ',pseudo{p},' images in ...
',num2str(ptime),' seconds.\r']);
end
strng = varpath; strng(strng=='\') = '/';
fprintf(['Completed compiling data for ',strng,'\r']);

%%
dtau = partdat.data(:,2);
figure,plot(dtau,RMSx2(:,1),'bo-',dtau,RMSx3(:,1),'gs--',dtau,RMSxl(:,1),'rd-.');
title('Artificial Image Pairs RMS Uncertainty');
xlabel('d{\tau}');
ylabel('RMS Uncertainty, [pixels]');

for p=1:length(pseudo)
    fid = fopen([varpath,'\ProcessedData\RMS\RMSx',pseudo{p},'.dat'],'w');
fprintf(fid,'dtau Lanczos2 Lanczos3 Linear\r');
    for i=1:length(dtau)
        fprintf(fid,'%e %e %e %e \r',dtau(i),RMSx2(i,p),RMSx3(i,p),RMSxl(i,p));
    end
fclose(fid);
    fid = fopen([varpath,'\ProcessedData\RMS\RMSy',pseudo{p},'.dat'],'w');
fprintf(fid,'dtau Lanczos2 Lanczos3 Linear\r');
    for i=1:length(dtau)
        fprintf(fid,'%e %e %e %e \r',dtau(i),RMSy2(i,p),RMSy3(i,p),RMSyl(i,p));
    end
fclose(fid);
end