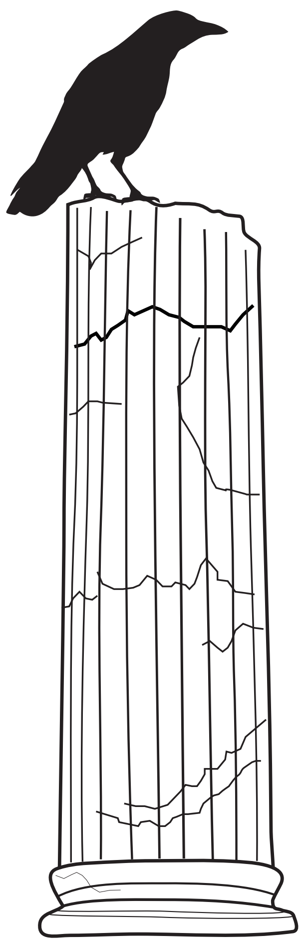
# User's Guide to OSIV 2.1

James Strother



Open

Source

Image

Velocimetry

# Chapter 1

# Installation

The OSIV suite may be installed either from pre-built binaries or by compiling the source code. In general, novices will find installation from pre-built binaries more convenient, while more experienced users may appreciate the ability to run on platforms for which no pre-built packages exist or to customize the OSIV build. Regardless of the installation method which you choose, you will need to have the TIFF, PNG, and fftw3 libraries installed on you system. If your distribution does not provide pre-built packages for these libraries (most do), then the TIFF library may be downloaded at *www.libtiff.org*, the PNG library may be downloaded at *www.libpng.org*, and the fftw3 library may be downloaded at *www.fftw.org*. If you attempt to install without these libraries, the installer will inform you which of the libraries are missing.

## Section 1: Installing from a binary RPM

RPM packages can be used with several common linux distributions. All available RPMs can be downloaded from the OSIV website. You should take care to download an RPM which is appropriate for your system. When you have found the right RPM, you can install with a command of the form

```
$ rpm -i osiv-2.1.2-opensuse110.i386.rpm
```

If an RPM for your system is not available, then you can still install from the source code.

## Section 2: Installing from a Debian package

Debian packages are available for the most recent Debian releases, and can be downloaded from the OSIV website. To install the package to your system, you should run a command of the form

```
$ dpkg -i osiv_2.1.2_lenny_i386.deb
```

## Section 3: Source Installation

Compiling OSIV from the source code is relatively simple, and provides the fastest code possible. Before you begin, you must acquire the development files for the TIFF, PNG, and fftw3 libraries. If you installed these libraries from the source code, then the headers have already been installed. If you installed these libraries as pre-built binaries, then you may also need to install additional development packages (typically with names sufficed with "-devel" or "dev").

After you have installed the necessary library headers, you may download the source code from the OSIV website into a convenient directory on your hard drive. The source code can be decompressed and unpackaged with this command

```
$ cat osiv-2.1.2.tar.gz | gunzip | tar xvf -
```

The `configure` script in the top directory allows you to customize the build for your system. In most cases the configuration script can be run without any options

```
$ cd osiv-2.1.2
$ ./configure
```

If you would prefer to customize the build, the OSIV suite may be built with a number of options. The suite may be built with a non-default compiler, with more aggressive optimization flags, using alternative libraries, or by directing installation to a non-default location. The list of available options may be viewed with the command

```
$ ./configure --help
```

When you have come to a satisfactory configuration, you may begin compilation with the command

```
$ make
```

If the build completes without error, you may now install the programs onto your system. As a user with write permissions for the installation direction, run the following command

```
$ make install
```

If you intend to leave the source code on your system, you should probably clean up the temporary files that were generated during the build. No harm results from leaving them on your system, but they needlessly occupy disk space. Running the following command will clean up the source tree

```
$ make clean
```

# Chapter 2

# Getting Started

OSIV is a set of programs for performing particle image velocimetry (PIV) analysis, a ubiquitous technique in modern fluid mechanics for visualizing flow fields. This chapter will give you a very brief introduction to the package so that you will be able to starting using the software as soon as possible. Later chapters will explain each of the algorithms that are provided in the package, and will provide references to papers in the literature with detailed explanations of each algorithm's advantages and disadvantages. Please visit the website if you have any questions or comments, or if you would like to file a bug report.

## Section 1: Provided Programs

The OSIV suite contains four programs to perform particle image velocimetry. The most important program provided by the suite, `osiv_corr`, accepts images and calculates flow velocity vectors. This program performs all the real work of PIV. The second program, `osiv_draw`, accepts the output generated by `osiv_corr` and generates publication quality flow field figures. The third program, `osiv_synth`, is used to generate synthetic images to test the accuracy of the various algorithms provided by `osiv_corr`. The last program, `osiv_dump`, is used to extract the contents of the binary format produced by `osiv_corr` into an often convenient text format.

## Section 2: Performing cross-correlation

For all of the following examples, you will need to have a pair of images to perform PIV calculations. If you do not yet have images of your own, images may be downloaded from the OSIV website. The examples that follow will assume that are are working on the files *flap0.tif* and *flap1.tif*, though the names of any TIFF image pair may be substituted.

Each of the programs provided with OSIV have default values for all of the parameters required to perform PIV. Though it is highly recommended that a user invest the time necessary to become familiar with the function of each parameter, the default values should provide satisfactory initial results. Thus, you may perform PIV cross-correlation of the images *flap0.tif* and *flap1.tif* with the following simple command

```
$ osiv_corr flap0.tif flap1.tif
```

This will produce a single file *osiv.out* which contains the resulting displacement vectors. You should probably not try to view the contents of this file directly as the data is stored in a binary format that will likely cause your terminal undue difficulty. A binary format was chosen as it is much faster, smaller, and extensible than an equivalent text file. The same file can be read on any platform, and is compatible with all previous and future releases. You can, however, always using the `osiv_dump` command to extract the data to more convenient text format. To view all of the parameters that were selected by default, run the command

```
$ osiv_dump -a osiv.out
```

The exact meaning of each of the parameters may be found in the "manpage" for `osiv_corr`. To view the calculated displacement vectors, run the command

```
$ osiv_dump -v osiv.out | more
```

## Section 3: Generating vector plots

The results calculated above may be visualized using vector plots generated by the program `osiv_draw`. The command `osiv_draw` can be used to generate highly configurable vector plots, but like `osiv_corr`, will always choose reasonable default values. To generate a plot of the flow field that was just calculated run the following command

```
$ osiv_draw osiv.out
```

This command will generate the file *osiv.eps*, an Encapsulated Postscript file that can be viewed using a program such as Ghostview, Adobe Illustrator, Apple Preview, etc.

## Section 4: Modifying default parameters

Upon examining the flow field in the above example, you may notice that there are a few outliers in the data set. By default `osiv_corr` implements the very common Fourier cross-correlation algorithm, which is exceedingly fast though less accurate than other available methods. To re-calculate the flow field vectors using a direct cross correlation algorithm and then write results to the file *direct.osiv* run the following command

```
$ osiv_corr --xcorr-alg=2 -o direct.osiv flap0.tif flap1.tif
```

This algorithm is quite a bit slower than the default algorithm, and may take up to a minute to complete. When it finishes, you may view the resulting flow field using the same graphing commands you used with the previous example. If you are still not satisfied with the outliers in the data, you may re-run the correlation using larger windows. Using larger windows has the effect of removing some outliers at the cost of decreased spatial resolution of the resulting flow field. The default subwindow size is 32x32 pixels, to use a 64x64 window run the following command

```
$ osiv_corr --xcorr-alg=2 --wind-size=64 \
   -o direct.osiv flap0.tif flap1.tif
```

This command should produce a smooth flow field. At the same time, you can easily imagine that as you set more parameters, that passing all parameters on the command line could become very tedious. For this reason, `osiv_corr` can also accept parameters passed to it from a parameter file. The parameter file has a simple text format that parallels the command line options. For example, the above example could also performed saving the following lines to the file *direct64.txt*

```
xcorr_alg = 2
wind_size = 64
```

then running the following command

```
$ osiv_corr -p direct64.txt -o flap.osiv flap0.tif flap1.tif
```

You might notice that format of the input parameter file is very similar to the arguments list you had previously printed with `osiv_dump`. In fact, the output any `osiv_dump -a` command is almost a valid input parameter file. This file is only "almost" a valid input file as the the `movie_one`, `movie_two`, and `output_file` parameters may be specified only on the command line.

## Section 5: Conclusion

Hopefully, you now have a good idea of how OSIV works in general, and how the various programs work with each other. The following chapters will describe each of these programs in detail.

# Cross-Correlation

# Chapter 3

The heart of PIV is calculating particle displacements by performing cross-correlation on windows taken from an image pair. OSIV provides the program `osiv_corr` to calculate these displacements. The following sections will describe the interface to `osiv_corr`, and then will describe the algorithms in more detail.

## Section 1: Passing Parameters

Correlation parameters can be passed to `osiv_corr` either through the command line or via a parameter file. Parameters may not be specified more than once in either the parameter file or the command line, but may be specified both in the command line and parameter file. Parameters passed on the command line will take precedence over parameters in the parameter file.

### Parameter File

One of the command line options that `osiv_corr` accepts is the name of a text file which contains additional parameters. This allows the most common parameters to be saved in a file, and those that change for each run to be modified on the command line. The name of the file is passed using the `-p` option.

The format of this parameter file is simple. The name of the variable is specified on the left of an equals sign with the value of the parameter on the right. Whitespace is allowed anywhere in the assignment. Quotes around the right hand value will be treated literally. Lines may be commented out using the pound sign ('#'). Comments may begin anywhere on the line, and may follow an assignment. Lines containing no text or only whitespace have no effect. The following is a typical parameter file

```
# parameters from JAS
xcorr_alg = 1
wind_size = 32 # TODO: too big?
disp_xoffset = 3
disp_yoffset = 0

# my own modifications
grid_xspacing=1
grid_yspacing=1
```

The variable names in the parameter file are nearly identical to the options that may be specified on the command line. The difference is only that variables on the command line use a hyphen to separate words, whereas variables in the parameter file use underscores.

### Invalid Parameters

It is certainly possible to specify unusable combinations of parameters. For example, giving incorrect image dimensions, vector grids that extend past the actual image, and so on. For the most part these restrictions are simply common sense, but a set of inequalities which the parameter values must satisfy is given in the Appendix. If parameters are passed which don't satisfy these inequalities then `osiv_corr` reports the error and exits.

## Section 2: Inputing Image Files

### Image Types

Currently, `osiv_corr` supports only two image formats: TIFF and a raw binary stream. TIFF file are recognized as files that end with the suffix *.tif* or *.tiff*, and binary files are recognized as files that end with the suffix *.dat*. It should be noted that accepting images through a binary stream is included largely for future expansion, and should not find common use. Support for TIFF is provided by libtiff, and so the vast majority of TIFF formats are accepted.

### Image Pairs

If you simply wish to perform correlation on a set of two images, then the command line call is simple. For example, if you had an image pair saved as *flap0.tif* and *flap1.tif* then the call would be

```
$ osiv_corr flap0.tif flap1.tif
```

### Image Sequences

If you would like to perform correlation on a sequence of images, then the calling convention is a little more complex. To fully specify the sequence, you must specify a start frame, a finishing frame, the frame skipping number, and the frame set number. This is probably simpler than it sounds. Assume that you have two arrays of images, called 'A(i)' and 'B(i).' The start frame specifies the first 'A' frame on which to perform correlation. The frame set number specifies the frame number of the matching 'B' frame in relation to the 'A' frame number. The frame skipping number specifies the number of frames to skip between each pair of images. Lastly, the movie finishing number specifies the last 'A' frame on which to perform correlation. Some examples are definitely in order. Let's say that you wanted to correlate 'A(1)' with 'B(1)', 'A(2)' with 'B(2), and 'A(3)' with 'B(3)'. You would use the following parameter file

```
movie_start = 1
movie_finish = 3
movie_skip = 1
movie_set = 0
```

If you wanted to perform correlation of 'A(2)' with 'B(5)' and 'A(4)' with 'B(7)', then the parameter file would be

```
movie_start = 2
movie_finish = 4
movie_skip = 2
movie_set = 3
```

### Image Sequences in Single Files

TIFF files are capable of storing multiple images in a single file. If you have stored you images in this way, then calling `osiv_corr` with an image sequence is identical to calling `osiv_corr` with a single image pair. The name of the first frame in the file is always called '0' by convention. For example, to perform correlation using the first three frames from the movies stored in the files 'alpha.tiff' and 'beta.tiff', you would use the command

```
$ osiv_corr --movie-finish=2 alpha.tiff beta.tiff
```

You should note that since the frame number starts at zero that, the last frame should be a two and not a three.

### Image Sequences in Multiple Files

If the sequence of images is stored in multiple files then you must somehow indicate to `osiv_corr` the name of the file that holds the $i^{th}$ frame. To do this, you may pass `osiv_corr` a image filename that closely resembles an ANSI printf format string. If you aren't familiar with the printf function, the examples which follow should provide you with most of the information which you would need. Basically, `osiv_corr` scans over the filename replacing every '%d' with the current frame number. If you were to pass it the filename 'flap%d.tif' then it will expect to find the $0^{th}$ frame in the file 'flap0.tif.' So, the example from the above section *Image Pairs* could just have easily have been made with the command

```
$ osiv_corr --movie-set=1 'flap%d.tif' 'flap%d.tif'
```

In fact, it is unnecessary to pass the name of the same movie twice. If you specify a single filename, then `osiv_corr` assumes that you intend for both images to be taken from the same sequence. So the following call would accomplish exactly the same thing as that above

```
$ osiv_corr --movie-set=1 'flap%d.tif'
```

This printf convention allows a rather versatile way of manipulating the format of the frame number. If, for example, you had a sequence of files named *camel0001.tif*, *camel0002.tif*, and so one. Then then you could specify the following command

```
$ osiv_corr --movie-start=1 --movie-set=1 'camel%04d.tif'
```

Within the filename string, the '4' specifies that the resulting number should fill four characters, while the '0' indicates that the number should be padded to four characters using zeros rather than the default of spaces. Lastly, it is suggested that the printf-style filenames which are passed on the command line be placed within single quotes (as in the above examples). This prevents the shell from performing unexpected expansions of the character '%' that could result in invalid filenames.

## Section 3: FFT Wisdom

The `osiv_corr` uses the fftw library to perform Fast Fourier transforms. This library is extremely fast and very versatile. It is also capable of self-tuning, using performance data gathered on previous runs to improve cross-correlation performance. By default, `osiv_corr` turns off most self-tuning to avoid the startup overhead. However, if you would like to make use of this then you can generate 'FFT wisdom' yourself with the utility `osiv_tune` and then pass generated file to `osiv_corr` using the option `--fft-wisdom`. This information is highly system specific, so you should try to use wisdom that is generated on the same machine as you intend to run `osiv_corr`. Also, the wisdom that is generated is specific to the window size. While passing wisdom for a different window size is unlikely to hurt, it is also unlikely to help. As an example, the following would generate a wisdom file for and run `osiv_corr` with a 32x32 window

```
$ osiv_tune --wind-size 32 -o wise32.dat
$ osiv_corr --wind-size 32 --fft-wisdom wise32.dat ...
```

## Section 4: Pre-processing Algorithms

There are currently five pre-processing methods supported. The algorithm which is used is specified with the `pproc-alg` option. Possible values are

**0** No Pre-processing
**1** Subtract to Minima
**2** Add to Maxima
**3** Stretch to Limits
**4** Stretch to Variance
**5** Stretch to Average

The purpose of these algorithms is to normalize for imperfections in the lighting conditions or image capture that would cause some pixels to collect biased intensities. The algorithms seek to find trends in long image sequences, and then normalize the images to provide an unbiased representation. It should be noted that these algorithms should never be used on a single image pair or a small set of image pairs where this kind of normalization is likely to do more harm then good. Unfortunately, relatively few studies have examined the effect of this normalization on the accuracy of the correlation (but see [5, 2]), and so concrete statements regarding the accuracy of each method are difficult. Nonetheless, when the images are of good quality, it is likely that pre-processing can omitted with little ill-effect. But when the images contain excessive background data, the "subtract to minima" algorithm is likely to improve displacement quality.

### Subtract to Minima

This algorithm finds the minimum pixel value for a particular pixel location over the set of all frames, then adds the difference between the smallest representable pixel value and this value to the the same pixel location in every frame. This has the effect of shifting the intensity of the entire image down, such that the lowest intensity pixels have the lowest representable values.

$$\xi^\star(\vec{x},t) = \xi(\vec{x},t) + \vartheta_{min} - \xi_{min}(\vec{x})$$

where $\xi^\star(\vec{x},t)$ is the normalized image, $\xi(\vec{x},t)$ is the unnormalized image, $\vartheta_{min}$ is the smallest representable pixel intensity, and $\xi_{min}(\vec{x})$ is the minimum pixel intensity from the set of all frames.

### Add to Maxima

This algorithm finds the maximum pixel value for a particular pixel location over the set of all frames, then adds the difference between the largest representable pixel value and this value to the same pixel location in every frame. This has the effect of shifting the intensity of the entire image up, such that the highest intensity pixels have the highest representable values.

$$\xi^{\star}(\vec{x},t) = \xi(\vec{x},t) + \vartheta_{max} - \xi_{max}(\vec{x})$$

where $\xi^{\star}(\vec{x},t)$ is the normalized image, $\xi(\vec{x},t)$ is the unnormalized image, $\vartheta_{max}$ is the largest representable pixel intensity, and $\xi_{max}(\vec{x})$ is the maximum pixel intensity from the set of all frames.

### Stretch to Limits

This algorithm causes the minimum pixel intensity to be placed at the smallest representable pixel value, and the maximum intensity to be placed at the largest representable pixel value. This has the effect of stretching the range of the intensity values in the image to match the range of representable values.

$$\xi^{\star}(\vec{x},t) = \frac{\vartheta_{max} - \vartheta_{min}}{\xi_{max}(\vec{x}) - \xi_{min}(\vec{x})}(\xi(\vec{x},t) - \xi_{min}(\vec{x})) + \vartheta_{min}$$

where variables are defined as above.

### Stetch to Variance

This algorithm causes the center 95% of the pixels values to be stretched over the representable domain, while the remaining 5% are saturated at the limits. This is similar to the *Stretch to Limits* algorithm, except that it uses a more statistical representation of the intensity range.

$$\xi^{\star}(\vec{x},t) = \frac{1}{4\,\xi_{std}}(\vartheta_{max} - \vartheta_{min})\,(\xi(\vec{x},t) - \xi_{avg}(\vec{x}) + 2\,\xi_{std}(\vec{x})) + \vartheta_{min}$$

where $\xi_{std}(\vec{x})$ is the standard deviation of the pixel intensity, and $\xi_{avg}(\vec{x})$ is the mean pixel intensity.

### Stretch to Average

This assumes that a zero pixel value should correspond to the smallest representable pixel value, then scales the all the pixel values such that the average observed intensity has the pixel value in the middle of the representable range.

$$\xi^{\star}(\vec{x},t) = \left(\frac{\vartheta_{max} - \vartheta_{min}}{2\ \xi_{avg}(\vec{x})}\right)\xi(\vec{x},t) + \vartheta_{min}$$

Normalized values which exceed the variable range are rounded to the nearest representable value.

## Section 5: Correlation Algorithms

There are currently ten correlation algorithms supported. The algorithm which is used is specified with the `xcorr-alg` option. Possible values are

**1** Direct Least Squares
**2** Fast Direct Least Squares
**3** Direct Correlation
**4** Fast Direct Correlation
**5** Fourier Correlation
**6** Fourier Least Squares
**7** Iterative Fourier Correlation
**8** Direct Normalized Correlation
**9** Fast Direct Normalized Correlation
**10** Fourier Normalized Correlation

For example, to perform correlation using Direct Least Squares, you would use a command of the form

```
$ osiv_corr --xcorr-alg=2 ...
```

The following sections will describe each of the algorithms in detail. It may be useful to skip these sections your first time through this document.

## Direct Least Squares

This algorithm minimizes the least squares difference between a window in the present frame and the window shifted by some displacement in the next frame. This direct calculation is generally slower than methods using Fourier-based cross-correlation, but often yields less RMS error compared to both direct cross-correlation and Fourier-based cross-correlation methods [3, 4, 9].

Formally, we define the the error function, $\varepsilon(x, y, \delta x, \delta y)$, to be

$$\varepsilon(x, y, \delta x, \delta y) = \sum_{i=0}^{W_x-1} \sum_{j=0}^{W_y-1} (\xi_1(x+i, y+j) - \xi_2(x+i+\delta x, y+j+\delta y)^2$$

where $(W_x, W_y)$ are the dimensions of the evaluation window, $(x, y)$ is the position of the upper-left corner of the window, $(\delta x, \delta y)$ is the displacement vector, $\xi_1$ is the first image, and $\xi_2$ is the second image. The displacement $(\delta x, \delta y)$ for which $\varepsilon(x, y, \delta x, \delta y)$ is minimized is considered the best approximation to the true displacement.

## Fast Direct Least Squares

This is mathematically identical to the Direct Least squares algorithm, but manages to avoid the computational redundancy associated with overlapping window portions. This is accomplished by first calculating an indefinite sum for the whole image, then evaluating the sum at specific points to calculate the sum for individual windows. It has all of the benefits of a direct least squares (it yields *exactly* the same displacements), but is generally faster.

Formally, we first define the indefinite sum, $\zeta(x, y, \delta x, \delta y)$, be given by

$$\zeta(x, y, \delta x, \delta y) = \sum_{i=0}^{x} \sum_{j=0}^{y} \left(\xi_1(i, j) - \xi_2(i+\delta x, j+\delta y)\right)^2,$$

where $(\delta x, \delta y)$ is the displacement vector, $\xi_1$ is the first image, and $\xi_2$ is the second image.
Then the error function, $\varepsilon(x, y, \delta x, \delta y)$, is given by

$$\begin{aligned}\varepsilon(x, y, \delta x, \delta y) \;=\; & \zeta(x-1, y-1, \delta x, \delta y) + \zeta(x+W_x-1, y+W_y-1, \delta x, \delta y) - \\ & \zeta(x-1, y+W_y-1, \delta x, \delta y) - \zeta(x+W_x-1, y-1, \delta x, \delta y)\end{aligned}$$

As before, the displacement $(\delta x, \delta y)$ for which $\varepsilon(x, y, \delta x, \delta y)$ is minimized is considered the best approximation to the true displacement.

## Direct Correlation

This algorithm performs an unnormalized direct cross-correlation, and has been described as an approximation to the Direct Least Squares algorithm [4]. This algorithm typically results in RMS errors that are greater than those from the Direct Least Squares algorithm [4] and, as such, is included only for comparative purposes.

Formally, consider an expansion of the least squares error function

$$\left(\xi_1(x, y) - \xi_2(x+\delta x, y+\delta y)\right)^2 = \xi_1^2(x, y) + \xi_2^2(x+\delta x, y+\delta y) - 2\,\xi_1(x, y)\,\xi_2(x+\delta x, y+\delta y)$$

The first term is not a function of $(\delta x, \delta y)$ and thus does not affect the location of the minimum. The second term *is* a function of $(\delta x, \delta y)$; however, as it is the sum of an arbitrary window in the second image, it is expected to remain constant for relatively large windows taken from a homogenous image $\xi_2(x, y)$. Thus, we take the correlation function $\hat{\varepsilon}(x, y, \delta x, \delta y)$ as

$$\hat{\varepsilon}(x, y, \delta x, \delta y) = \sum_{i=0}^{W_x-1} \sum_{j=0}^{W_y-1} \xi_1 (x+i, y+j)\, \xi_2 (x+i+\delta x, y+j+\delta y)\,,$$

whose maxima correspond approximately to minima of the function $\varepsilon(x, y, \delta x, \delta y)$.

### Fast Direct Correlation

This algorithm is mathematically identical to the Direct Correlation algorithm but, similarly to the Fast Direct Least Squares algorithm, eliminates redundancy associated with overlapping windows to achieve improved performance.

The mathematical basis of this algorithm closely mirrors that of the Fast Direct Least Squares, and will consequently be omitted. The difference between the two algorithm lies only in that here the indefinite sum is given by

$$\zeta(x, y, \delta x, \delta y) = \sum_{i=0}^{x} \sum_{j=0}^{y} \xi_1(i, j) \times \xi_2(i+\delta x, j+\delta y)$$

### Fourier Correlation

This algorithm makes use of the Faltung Theorem in order to make a fast approximation of an unnormalized direct cross correlation. Due to the high performance of this algorithm it is frequently preferred, though it is known to produce greater RMS errors [4] and peak-locking artifacts [6] then corresponding direct methods.

Formally, for large arrays one finds that

$$\hat{\varepsilon}(x, y, \delta x, \delta y) \approx \breve{\varepsilon}(x, y, \delta x, \delta y) = F^{-1}[F[\xi_1] \cdot F^*[\xi_2]],$$

where $F[\,]$ is the Fourier transform, $F^{-1}[\,]$ is the inverse Fourier transform, and $()^*$ denotes the complex conjugate.

### Fourier Least Squares

This method could be appropriately called a "Fourier-accelerated" Direct Least Squares. The location of the correlation peak is calculated as in the "Fourier Correlation" method, however, the direct least squares error is then calculated around this peak and then used for subpixel interpolation. This algorithm results in performance that approximates the Fourier Correlation algorithm with accuracy that approaches the Direct Least Squares. [4]

### Iterative Fourier Correlation

Due to assumptions inherent in Fourier cross correlation, the RMS error of calculated displacements tends to increase at greater true particle displacements. The Iterative Fourier Correlation algorithm attempts to compensate for this effect by iteratively shifting the correlation window toward the calculated displacement, such that the final subpixel interpolation occurs around a cross-correlation map whose peak occurs at a zero displacement. This algorithm has improved performance compared with a direct calculation, and has improved accuracy compared with typical Fourier Correlation. [12]

### Direct Normalized Correlation

This algorithm performs a normalized direct cross-correlation. This algorithm differs from the direct correlation in that it is normalized to compensate for variation in the local image intensity. This algorithm typically results in very low bias and RMS error [10], but with greater computational cost than FFT-accelerated algorithms.

Formally, we take the normalized cross-correlation function $\hat{\varepsilon}(x, y, \delta x, \delta y)$ as

$$\hat{\varepsilon}(x, y, \delta x, \delta y) = \frac{\sum_{i=0}^{W_x-1} \sum_{j=0}^{W_y-1} \xi_1' (x+i, y+j)\, \xi_2' (x+i+\delta x, y+j+\delta y)}{\sqrt{\left(\sum_{i=0}^{W_x-1} \sum_{j=0}^{W_y-1} \xi_1' (x+i, y+j)^2\right) \left(\sum_{i=0}^{W_x-1} \sum_{j=0}^{W_y-1} \xi_2' (x+i+\delta x, y+j+\delta y)^2\right)}}$$

where

$$\xi_1'(x+i,y+j) = \xi_1(x+i,y+j) - \frac{1}{W_xW_y}\sum_{m=0}^{W_x-1}\sum_{n=0}^{W_y-1}\xi_1(x+m,y+n)$$

$$\xi_2'(x+i,y+j) = \xi_2(x+i,y+j) - \frac{1}{W_xW_y}\sum_{m=0}^{W_x-1}\sum_{n=0}^{W_y-1}\xi_2(x+m,y+n)$$

### Fast Direct Normalized Correlation

This algorithm is mathematically identical to the Normalized Correlation algorithm but, similarly to the Fast Direct Least Squares algorithm, eliminates redundancy associated with overlapping windows to achieve improved performance.

The mathematical basis of this algorithm closely mirrors that of the Fast Direct Least Squares, and will consequently be omitted.

### Fourier Normalized Correlation

This method is entirely analogous to the Fourier Least Squares method. The location of the correlation peak is calculated as in the "Fourier Correlation" method, however, the direct normalized correlation is then calculated around this peak and then used for subpixel interpolation. This algorithm results in performance that approximates the Fourier Correlation algorithm with accuracy that approaches the Direct Normalized Correlation.

## Section 6: Signal-to-Noise Ratio

Each correlation algorithm will provide a signal-to-noise ratio for each calculated displacement. For algorithms in which the correlation between two windows is maximized, the signal-to-noise ratio is taken as the peak correlation divided by the second highest correlation value. For algorithms in which the error between two windows is minimized, the signal-to-noise ratio is taken as the second smallest error divided by the minimum error. In both cases, the displacements immediately adjacent to the signal peak/minimum will excluded when searching for the noise peak/minimum. The size of the excluded region can be controlled with the `snr-xexcl` and `snr-yexcl` options. In very rare circumstances it is not possible to calculate the signal-to-noise ratio for a displacement. For example, when the noise peak is exactly equal to zero the signal-to-noise ratio is mathematically undefined. Such cases are unlikely to occur with real images, but may occur with synthetic images. When such a data set is encountered the signal-to-noise ratio is set to zero.

## Section 7: Interpolation Algorithms

Here each of the interpolation algorithms included in OSIV is presented first qualitatively and then mathematically. Available algorithms include

**0** No Interpolation
**1** Gaussian Interpolation
**2** Parabolic Interpolation
**3** Paraboloidal Interpolation
**4** Centroid Interpolation

### Gaussian Interpolation

This method fits the peak of the cross-correlation map to a Gaussian function using the value at the peak and the values at the peak's two nearest neighbors. Two independent fits are performed to approximate the subpixel displacement in the $x$ and $y$ directions. This method shows little bias, minimal peak-locking, and is the preferred peak interpolation method. [8, 7]

Formally, we calculate the subpixel offset as

$$x - x_\circ = \frac{1}{2}\,\frac{\ln\xi(x_\circ - 1) - \ln\xi(x_\circ + 1)}{\ln\xi(x_\circ - 1) + \ln\xi(x_\circ + 1) - \ln\xi(x_\circ)}$$

where $x_\circ$ is the location of the extreme value and $\xi(x)$ is the value of the correlation map at a displacement of $x$.

### Parabolic Interpolation

This method fits the peak of the cross-correlation map to a parabolic function using the value at the peak and the values at the peak's two nearest neighbors. Two independent fits are performed to approximate the subpixel displacement in the $x$ and $y$ directions. This method shows greater peak-locking than Gaussian interpolation [7], and is included only for comparative purposes.

Formally, we calculate the subpixel offset as

$$x - x_\circ = \frac{1}{2}\,\frac{\xi(x_\circ + 1) - \xi(x_\circ - 1)}{\xi(x_\circ - 1) + \xi(x_\circ + 1) - 2\,\xi(x_\circ)}$$

where $x_\circ$ is the location of the extreme value and $\xi(x)$ is the value of the correlation map at a displacement of $x$.

### Paraboloidal Interpolation

This method fits the peak of the cross-correlation map to an elliptic paraboloidal function using the value at the peak and the values at the peak's eight nearest neighbors. Two fits are performed to approximate the subpixel displacement in the $x$ and $y$ directions. To the best of this author's knowledge, the accuracy of such interpolation is unevaluated within the literature.

Formally, we calculate the subpixel offset as

$$x - x_\circ = \frac{1}{2}\,\frac{\bar{\xi}(x_\circ + 1) - \bar{\xi}(x_\circ - 1)}{\bar{\xi}(x_\circ - 1) + \bar{\xi}(x_\circ + 1) - 2\bar{\xi}(x_\circ)}$$

where,

$$\bar{\xi}(x) = \xi(x, y_\circ - 1) + \xi(x, y_\circ) + \xi(x, y_\circ + 1).$$

and $x_\circ$ is the location of the extreme value and $\xi(x, y)$ is the value of the correlation map at a displacement of $(x, y)$.

### Centroid Interpolation

This method calculates subpixel displacement based on the centroid of the peak. This method tends to result in peak-locking [13] and, as such, is included only for comparative purposes.

Formally, we calculate the subpixel offset as

$$x - x_\circ = \frac{\xi(x_\circ + 1) - \xi(x_\circ - 1)}{\xi(x_\circ - 1) + \xi(x_\circ) + \xi(x_\circ + 1)}$$

and $x_\circ$ is the location of the extreme value and $\xi(x)$ is the value of the correlation map at a displacement of $(x)$.

## Section 8: References

[1] R. Adrian. Particle-imaging techniques for experimental fluid mechanics. *Ann. Rev. Fluid Mech.*, 23:261–304, 1991.

[2] A.M. Fincham and G.R. Spedding. Low cost, high resolution DPIV for measurement of turbulent fluid flow. *Exp. Fluids*, 23:449–462, 1997.

[3] L. Gui and W. Merzkirch. A method of tracking ensembles of particle images. *Exp. Fluids*, 28:465–468, 1996.

[4] L. Gui and W. Merzkirch. A comparative study of the MQD method and several correlation-based piv evaluation algorithms. *Exp. Fluids*, 28:36–44, 2000.

[5] M. Honkanen and H. Nobach. Background extraction from double-frame PIV images. *Exp. Fluids*, 2005:349–362, 2005.

[6] H. Huang, D. Dabiri, and M. Gharib. On errors of digital particle image velocimetry. *Meas. Sci. Tech.*, 8:1427–1440, 1997.

[7] L. Lourenco and A. Krothpalli. On the accuracy of velocity and vorticity measurements with PIV. *Exp. Fluids*, 13:421–428, 1995.

[8] M. Marxen, P. E. Sullivan, M. R. Loewen, and B. Jähne. Comparison of gaussian particle center estimators and the achievable measurement density for particle tracking velocimetry. *Exp. Fluids*, 29:145–153, 2000.

[9] S. P. McKenna and W. R. McGillis. Performance of digital image velocimetry processing techniques. *Exp. Fluids*, 32:106–115, 2002.

[10] M. Piirto, H. Eloranta, P. Saarenenrinne, and R. Karvinen. A comparative study of five different piv interrogation algorithms. *Exp. Fluids*, 39:571–588, 2005.

[11] J. Westerweel. Fundamentals of digital image velocimetry. *Meas. Sci. Tech.*, 8:1379–1392, 1997.

[12] J. Westerweel, D. Dabiri, and M. Gharib. The effect of a discrete window offset on the accuracy of cross-correlation analysis of digital PIV recordings. *Exp. Fluids*, 23:20–28, 1997.

[13] C. Willert. Digitial particle image velocimetry. *Exp. Fluids*, 10:181–193, 1991.