Short Training Report 2020-05

IMPLEMENTATION OF ITERATIVE MULTIGRID AND WINDOW DEFORMATION SCHEMES IN THE OPENPIV PYTHON PACKAGE

T. Käufer

Supervisor: M.A. Mendez

December 2019

Short Training Report 2020-05

IMPLEMENTATION OF ITERATIVE MULTIGRID AND WINDOW DEFORMATION SCHEMES IN THE OPENPIV PYTHON PACKAGE

T. Käufer

Supervisor: M.A. Mendez

December 2019

Acknowledgments

I want to thank Prof. Miguel Mendez for his excellent advice as well as his outstanding mentorship during my Short Training Program, Prof. Alex Liberzon and Dr. William Thielicke for their great open-source PIV-tools and the fruitful discussion on PIV algorithms. Furthermore, all the colleagues that made my stay at the VKI as great as it was and the VKI itself for granting me a scholarship. Finally, Prof. Christian Cierpka for recommending me the VKI and supporting my application.

Abstract

Particle Image Velocimetry has become one of the most used flow measurement techniques, but even though its widely used in science, its application is still related to significant costs. To surpass the limitations due to the high license cost of proprietary software, in this work Iterative Multigrid and Window Deformation Schemes are implemented and added to the open-source OpenPIV Python package. The fundamentals of PIV interrogation, as well as advanced interrogation techniques like the Multipass interrogation, the Grid Refinement, and the Window Deformation, are recapitulated and investigated. Particular focus is given to the cross-correlation-based displacement estimation as the core function of the displacement analysis. The relationship between convolution and cross-correlation, its variants (circular and linear), as well as its fast calculation in the frequency domain, are extensively described on a one-dimensional example. The Python code for those examples is also provided.
An overview of how to set up a PIV analysis using the Iterative Multigrid and Window Deformation Schemes algorithm of the OpenPIV package is given, and the different parameters are explained.
This work concludes with a comparison between PIVlab and OpenPIV, using the Iterative Multigrid and Window Deformation Schemes, based on the test case A of the 2003 International PIV challenge. It is shown that the implemented interrogation schemes provide accurate and robust results.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Since its origin in the 1980's the Particle Image Velocimetry (PIV) became a well-established velocity measurement technique that significantly augment flow visualization techniques in the study of fluid flows [1]. The standard PIV allows the measurement of an instantaneous two-dimensional (2D) and two-component (2C) velocity field. More sophisticated extensions of the method, using additional camera, such as Stereo-PIV, allow for measuring also out-of-plane velocity component, providing 2D-3C field. Three-dimensional and three-components of velocity (3D-3C) measurements are possible in a volume with the methods such as Tomographic-PIV or light-field cameras and enable to determine indirect quantities such as pressure and forces. An extensive review is provided by the monograph [9].

To obtain a velocity field of the flow, the flow must be seeded with particles. Those particles must be of such a kind that their motion faithfully follows that of the flow. These particles are made visible by illuminating them in the measurement plane using a light source combined with appropriate optics. As a light source, lasers are commonly used. Particles in the measurement plane are observed by one or more cameras to determine their position in the image. These images are divided into smaller areas, called the interrogation area, where the flow motion is expected to be approximately uniform. By cross-correlating the interrogation windows of successive images, obtained within a short time interval between them Δt, the particle displacement between those two can be obtained. This displacement, in terms of pixel/Δt, can be transformed into a velocity in meters/second using the time between two successive images and the relation between the camera sensor and the measurement plane.

A typical 2D-PIV setup is shown in figure 1.1.

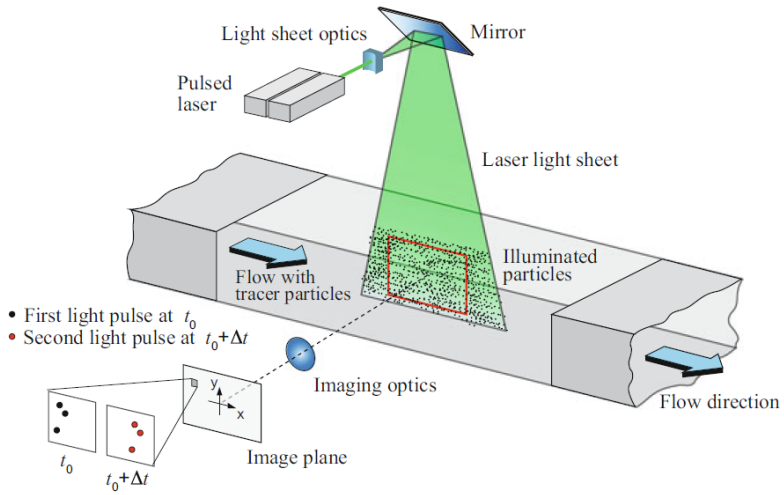


Figure 1.1: Experimetal arrangment of a typical 2D-PIV setup. Image taken from [9].

Even though PIV is a precise method for velocity measurements, the costs of a PIV equipment are high, therefore limiting its fields of application. There are two main cost factors related. At first, there are the costs for the laser, the cameras, the optic, and the evaluation hardware. At second, the costs of licenses for the evaluation software are worth mentioning.
To reduce the equipment costs, several low-cost setups using diodes as a light source and mobile phone cameras for image acquisition have been proposed [3]. However, their usability is limited by reason of the incoherence of the LED. A way to avoid the cost of a commercial evaluation software is the use of software available under an open-source license. There are several open-source software packages available for standard 2D-PIV evaluation. Among the most common ones is the MATLAB package PIVlab, developed and maintained by W. Thielicke [15], and the Python/C++ package OpenPIV developed and maintained by A. Liberzon [13] and the OpenPIV community. Although with several implementation differences, both codes offer advanced PIV-processing methods like multipass with deformable interrogation windows and subpixel estimation. However, since PIVlab requires MATLAB and several of its toolboxes, one depends on licenses that are quite expensive and therefore limit the open-source aspect. Therefore, this work focused on the extension of OpenPIV.

1.2 Aim of the project

This short training program aimed at obtaining a deep understanding of the main function and processes of a PIV interrogation. These include the cross-correlation and advanced interrogation techniques like the subpixel estimation and the Multipass processing. This knowledge is then be used to further improve the functionality of the available OpenPIV Python package to obtain a top-level and easy to use open-source PIV interrogation tool.

Chapter 2

Fundamentals of PIV interrogation

2.1 The cross-correlation and convolution in 1D

The displacement determination in PIV is often done by cross-correlation. This chapter gives a short overview of the cross-correlation in general and different methods to compute it.
In digital signal processing, the cross-correlation is a method to determine the similarity between two signals. These signals can be, for example, audio recordings or images. To limit the complexity, the cross-correlation is here explained on the example of 1D discrete signals which can be considered as audio signals. The Python code used to produce the signals and the plotted signals themselves can be seen in Program code 1 and figure 2.1.

The signals, denoted with I_1 and I_2, consists of 200 samples n and are obtained as the summation of two Gaussian curves in n. Even though the signals consist of discrete samples, these are displayed as continuous signals for plotting purposes.

The cross-correlation of discrete, one-dimensional, real valued signals of the same size can be defined as in equation (2.1). One might find other definitions of the cross-correlation, but in this work, the definition as in [6] is used.

$$R_{cor}[k] = \sum_{n=-\infty}^{\infty} I_1[n] I_2[n+k]. \tag{2.1}$$

In this equation R_{cor} is a vector containing the correlation values of the signals I_1

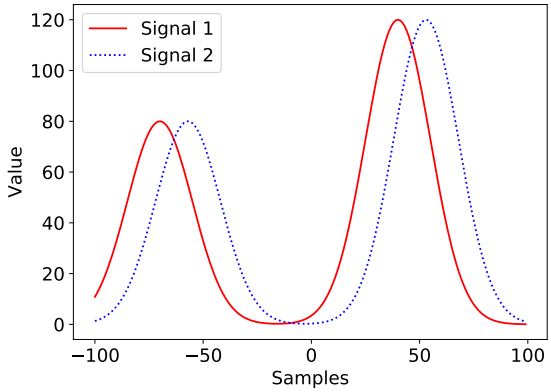
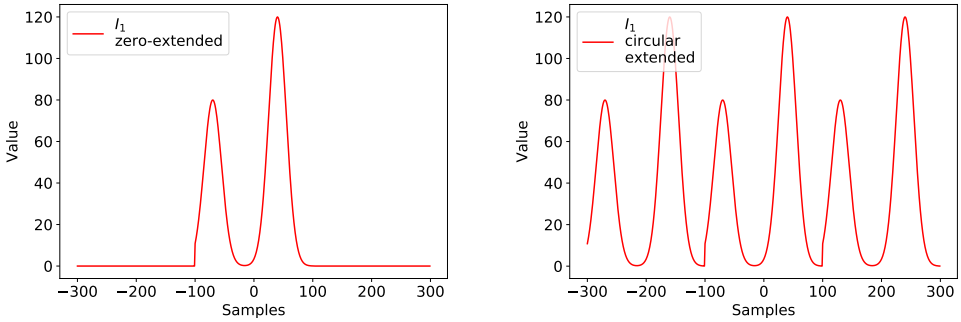


Figure 2.1: Example signals

```
import numpy as np
from matplotlib import pyplot as plt

n = np.arange(-100, 100)  # duration of the signal
sigma = 15  # sigma for gaus
P11 = -70  # first peak in first signal
P12 = 40  # second peak in first signal
P21 = -57  # first peak in second signal
P22 = 53  # second peak in second signal

I1 = 80 * np.exp(-((n - P11)**2) / (2 * sigma**2)) + \
        120 * np.exp(-((n - P12)**2) / (2 * sigma**2))

I2 = 80 * np.exp(-((n - P21)**2) / (2 * sigma**2)) + \
        120 * np.exp(-((n - P22)**2) / (2 * sigma**2))

plt.plot(n,I1)
plt.plot(n,I2)
```

Program code 1: Python code used to create the example signals



(a) Zero-extension of I_1 (b) Circular extension of I_1

Figure 2.2: Figure comparing the zero extension of I_1 with the circular extension.

and I_2 depending on the shift k. N is the signal size and n is the sample in the signal. The equation is defined for signals of infinite length, therefore, it is necessary to extend signals with finite length. This can be achieved in many ways, but the most common ways are the extension with zeros which leads to the so-called linear cross-correlation in equation (2.2), and the periodic extension of the signal itself which leads towards circular cross-correlation shown in equation (2.3). In addition to this equations, the zero extension as well as the circular extension of Signal I_1 is shown in figure 2.2.

$$R_{lincor}[k] = \sum_{n=0}^{N-1} I_1[n] I_2[n+k], \qquad \begin{aligned} &-(N-1) \leq k \leq N-1 \\ &I_{1,2}[n] = 0 \ ; \ \forall n \ \notin [0, N-1] \end{aligned} \tag{2.2}$$

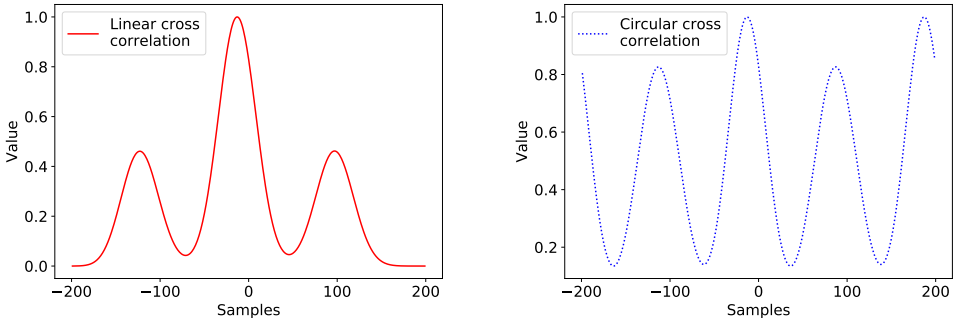
$$R_{circcor}[k] = \sum_{k=0}^{N-1} I_1[n] I_2[n+k], \qquad \begin{aligned} &0 \leq k \leq N-1 \\ &I_{1,2}[n] = I_{1,2}[N+n] \end{aligned} \tag{2.3}$$

The shift k can be limited to the sequence mentioned in the equations since for values outside of this sequence the correlation value would either be zero for the linear cross-correlation or repeat periodically for the circular cross-correlation. In figure 2.3 the linear and circular cross-correlation of the example signals is shown. Figure 2.3b is plotted over the same range as the linear cross-correlation. The code used to calculate the linear and circular cross-correlation is shown in Program code 2.

By comparing the results of the linear and circular cross-correlation, several differences become clear. The most obvious is the size of the correlation which is $2N$ - 1 for the linear cross-correlation while the curve of the circular cross-correlation repeats itself after a period of N. Furthermore, the shapes of the plotted curves differ. In the case of the linear cross-correlation the main peak is significantly higher than the secondary peaks

```
import numpy as np
import matplotlib.pyplot as plt

n=np.arange(-100,100) # period of the signal
sigma=15 #sigma for gaus
P11=-70  #first peak in first signal
P12=40 # second peak in first signal
P21=-57 # first peak in second signal
P22=53 #second peak in second signal

I1 = 80 * np.exp(-((n - P11)**2) / (2 * sigma**2)) + \
        120 * np.exp(-((n - P12)**2) / (2 * sigma**2))

I2 = 80 * np.exp(-((n - P21)**2) / (2 * sigma**2)) + \
        120 * np.exp(-((n - P22)**2) / (2 * sigma**2))

# Is you want to try with small vector for hand calculation
#x=np.array([1,0,3,2])
#y=np.array([0,1, 2,4])

#linearcorrelation of the signal
RLin=np.correlate(I1,I2, 'full'); # Linear correlation
Shifts=np.linspace(-len(I1)+1, len(I1)-1, 2*len(I1)-1) # Create the Shifts

#circular correlation of the signal
RCirc=(np.correlate(I1,np.hstack((I2[1:], I2)),mode='valid'))
# Here we assembly the full set, recalling periodicity.
# Observe: a -3 shift is equivalent to a +2 shift.
RCirc_Full=np.fft.fftshift(np.concatenate((RCirc,RCirc[1:len(RCirc)])))


#normalizing onto 1
RLin=RLin/RLin.max()
RCirc=RCirc/RCirc.max()
RCirc_Full=RCirc_Full/RCirc_Full.max()

# Circular Shifts
Ns=len(I1)-1

# Plot the results
plt.plot(Shifts,RLin)
plt.plot(Shifts,RCirc_Full)
```

Program code 2: Python code used to cross-correlate the example signals



(a) Linear cross-correlation of the example signals

(b) Circular cross-correlation of the example signals

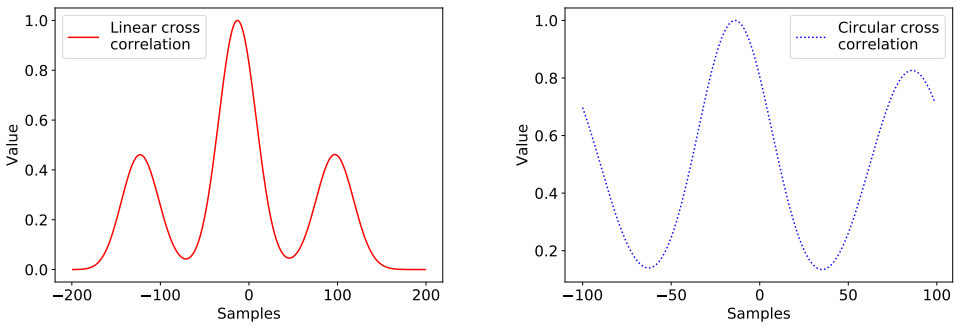
Figure 2.3: Figure showing the linear and circular cross-correlation of the example signals.

while the difference is not that significant for the circular cross-correlation. Also, the linear cross-correlation is approaching zero at the flanks.

Even though the linear cross-correlation in the signal domain provides the desired results, it comes with heavy computational costs. Therefore, an approach to determine the cross-correlation in the frequency domain, based on the Fast Fourier Transformation (FFT) and the convolution theorem, is commonly employed in practice [6]. To understand this, it is important to know the definition of convolution and its relation to cross-correlation. The convolution of discrete, one-dimensional, real-valued signals can be defined as in equation (2.4) [6].

$$R_{conv}[k] = \sum_{n=-\infty}^{\infty} I_1[n] I_2[k-n] \tag{2.4}$$

The variables are the same as in equation (2.1) and like for the cross-correlation, the equations for the linear and the circular convolution can be derived accordingly. Cross-correlation and convolution look quite similar and in fact, the only difference is the flipping of one signal. This means, that by flipping on signal one can switch between cross-correlation and convolution. The result for the linear and circular convolution with a flipped signal is visualized in figure 2.4 and the associated Python code is shown in Program code 3. As one can see, convolution with on flipped signal provides the same result as the cross-correlation regardless of the correlation type. Now that the relation between cross-correlation and convolution is explained, the next step is to have a look at



(a) Linear convolution of Signal 1 and flipped Signal 2

(b) Circular convolution of Signal 1 and flipped Signal 2

Figure 2.4: Figure showing the linear and circular convolution of Signal 1 and flipped Signal 2 .

```python
import numpy as np
from scipy import convolve
from scipy.fftpack import fftshift

n = np.arange(-100, 100)  # period of the signal
sigma = 15  # sigma for gaus
P11 = -70  # first peak in first signal
P12 = 40  # second peak in first signal
P21 = -57  # first peak in second signal
P22 = 53  # second peak in second signal

I1 = 80 * np.exp(-((n - P11)**2) / (2 * sigma**2)) + \
        120 * np.exp(-((n - P12)**2) / (2 * sigma**2))

I2 = 80 * np.exp(-((n - P21)**2) / (2 * sigma**2)) + \
        120 * np.exp(-((n - P22)**2) / (2 * sigma**2))

# flliping the signal
I2 = I2[::-1]
# linearcorrelation of the signal
RLin = convolve(I1, I2, 'full')
# circular correlation of the signal
RCirc = fftshift(convolve(I1, np.hstack((I2[1:], I2)), mode='valid'))

# normalizing onto 1
RLin = RLin / RLin.max()
RCirc = RCirc / RCirc.max()
```

Program code 3: Python code used to cross correlate the example signals by convolution with a flipped signal

the convolution in the frequency domain and the convolution theorem. The convolution theorem states that the convolution of a periodic signal in the signal domain e.g. time or space is equivalent to a multiplication in the Fourier Transformed domain e.g. frequency or wave numbers [16]. Therefore, the equation (2.5) is established.

$$\mathcal{F}^{-1}(\mathcal{F}[I_1]\mathcal{F}[I_2]) = I_1 * I_2 \tag{2.5}$$

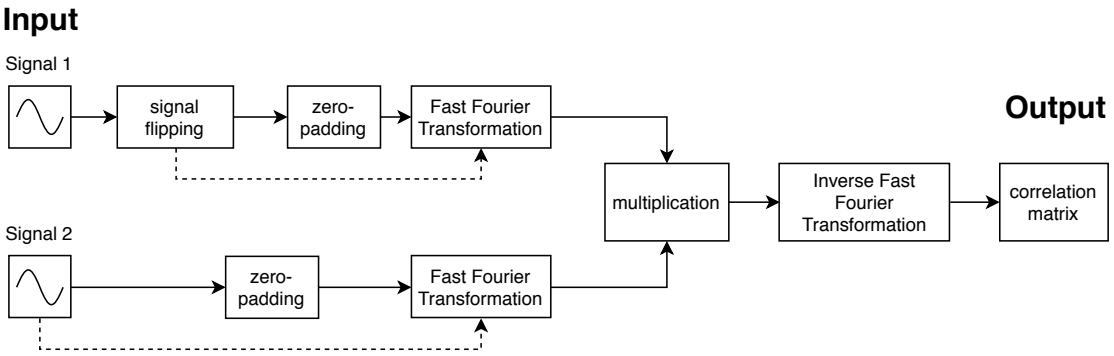
The variables I_1 and I_2 represent the signals, F and F^{-1} are the Fourier Transform and the inverse Fourier Transform and $*$ is the convolution operator. This means that one can easily calculate the circular cross-correlation and convolution by transforming the signals into the frequency domain, perform the multiplication and transforming the result back into the signal domain. Assuming a Fast Fourier Transformation algorithm is used, the computational cost is $\mathrm{O}\left[N \log_2 N\right]$ compared to $\mathrm{O}\left[N^2\right]$ for the direct calculation in the signal domain [9].

Since the convolution theorem is only valid for periodic signals, only the circular cross-correlation can be calculated in the frequency domain. Though, the linear cross-correlation is, in general, better suited to determine the shift between two signals because of the larger difference in the height between the main peak and the secondary peaks which results in a better signal to noise ratio. In order to benefit from the more efficient calculation in the frequency domain, a simple trick can be used: Before transforming the signals into the frequency domain, the signals are zero-padded to a length of (at least) $S = 2N - 1$. This zero extension of the signal eliminates the effects of the periodicity of the signal and one can obtain the linear cross-correlation using the FFT algorithm.

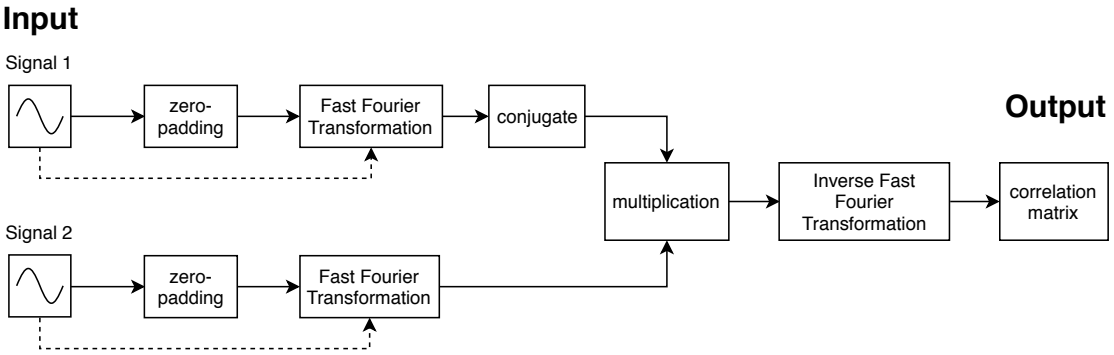
Following equation (2.5), the calculation consists in flipping one of the signals, zero-padding them adequately, transforming them into the frequency domain using the FFT, do the multiplication and transform the result back into the signal domain as shown in figure 2.5a. An example on how to implement the linear and circular cross-correlation in Python is shown in Program code 4 and the related results are shown in figure 2.6.

Again, by comparing the results of the FFT based linear and circular cross-correlation with the previous results one can see that they are the same.

As an alternative to the flipping of one signal at the beginning, one can multiply the conjugate-complex of one signal in the frequency domain since the flipping of a signal in the signal domain is equivalent to the conjugate-complex of the signal in the frequency domain. Figure 2.7 shows the comparison between the flipping and the conjugate-complex in the frequency domain. The related Python code is shown in Program code 5. Depending on the FFT algorithm , it might be necessary to rearrange

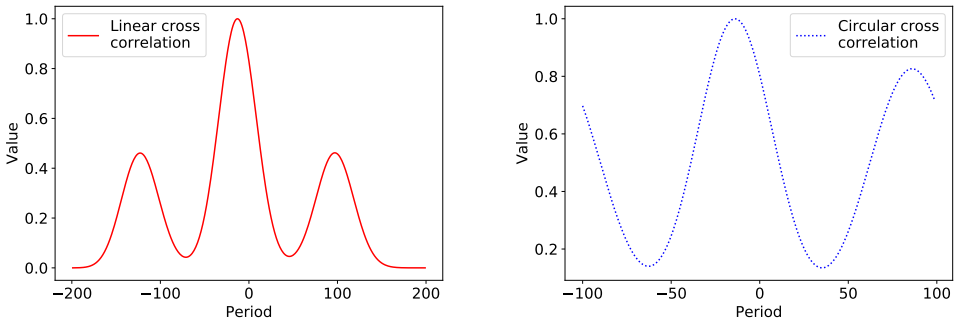


(a) Flowchart showing the principle of the FFT based cross-correlation with a flipped signal. The dotted line shows the procedure for the circular cross-correlation.



(b) Flowchart showing the principle of the FFT based cross-correlation with conjugate-complex multiplication. The dotted line shows the procedure for the circular cross-correlation.

Figure 2.5: Methods of signal manipulation to calculate the cross-correlation in the frequency domain



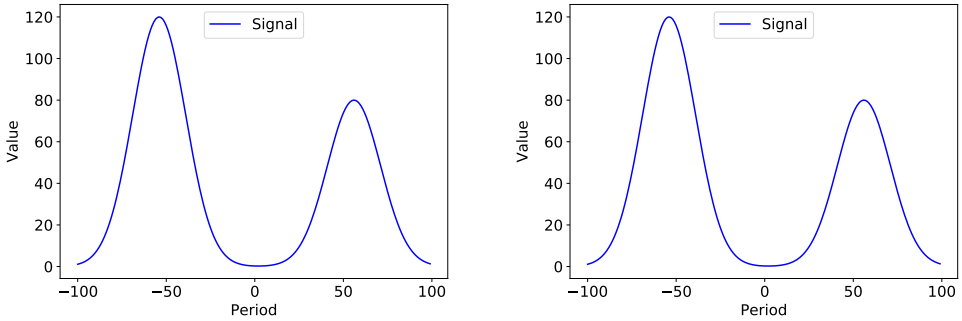
(a) FFT-based linear cross-correlation of Signal 1 and flipped Signal 2

(b) FFT-based circular cross-correlation of Signal 1 and flipped Signal 2

Figure 2.6: Figure showing the linear and circular-convolution of Signal 1 and flipped Signal 2 using the FFT .

```
import numpy as np
from numpy.fft import fft, ifft, fftshift

n = np.arange(-100, 100)  # period of the signal
sigma = 15  # sigma for gaus
P11 = -70  # first peak in first signal
P12 = 40  # second peak in first signal
P21 = -57  # first peak in second signal
P22 = 53  # second peak in second signal

# calculating the signal values
I1 = 80 * np.exp(-((n - P11)**2) / (2 * sigma**2)) + \
        120 * np.exp(-((n - P12)**2) / (2 * sigma**2))

I2 = 80 * np.exp(-((n - P21)**2) / (2 * sigma**2)) + \
        120 * np.exp(-((n - P22)**2) / (2 * sigma**2))

I1 = I1[::-1]
'linear cross correlation'
# calculate the range of the zero padding
padding_size = np.size(I1, 0) + np.size(I1, 0) - 1
# multiplication in the frequency domain and retransformation
# zero-padding is done within the fft
RLin = ifft(fft(I1, n=padding_size) * fft(I2, padding_size)).real

'circular cross correlation'
# multiplication in the frequency domain and retransformation
RCirc = fftshift(ifft(fft(I1) * fft(I2)).real)

# normalizing the result
RLin = (RLin / RLin.max())
RCirc = (RCirc / RCirc.max())
```

Program code 4: Python code used to cross-correlate the example signals in the frequency domain



(a) Flipped signal

(b) Inverse transformed conjugate-complex of the signal in the frequency domain

Figure 2.7: Comparison of a flipped signal with the inverse transformed complex-conjugate signal

the correlation vector. The procedure using the conjugate-complex multiplication in the frequency domain to calculate the cross-correlation is visualized as flowchart in figure 2.5b.

```
import numpy as np
from numpy.fft import rfft, irfft

n = np.arange(-100, 100)  # period of the signal
sigma = 15  # sigma for gaus
P11 = -70  # first peak in first signal
P12 = 40  # second peak in first signal
P21 = -57  # first peak in second signal
P22 = 53  # second peak in second signal

I1 = 80 * np.exp(-((n - P11)**2) / (2 * sigma**2)) + \
        120 * np.exp(-((n - P12)**2) / (2 * sigma**2))

# flipping signal I1
I1_flip = I1[::-1]

# flipping in fourier domain
I1_con = irfft(np.conj(rfft(I1))).real
I1_con = np.roll(I1_con, -1)
```

Program code 5: Python code used to compare the flipped signal with the inverse transformed complex-conjugate signal

2.2 Displacement evaluation

As mentioned before, the cross-correlation can be applied to determine the similarity of two signals. The highest similarity is determined by the maximum of the correlation matrix, the so-called correlation peak. This correlation peak can be used to determine the shift of the signals. For that, the position of the correlation peak in the correlation matrix is ascertained and the default peak position of the auto-correlation, which is the cross-correlation of the signal with itself and positioned in the middle of the correlation matrix, is subtracted. The result of this operation is the relative shift of the signals to each other which corresponds to the particle displacement in PIV.
Even though this seems straight forward, several limitations and properties must be taken into account when this procedure is applied for PIV [9].

- Since the convolution theorem is only defined for periodic signals, the signals must be modified as mentioned in section 2.1. But the mentioned zero padding cannot be used directly because the transition from the non-zero background of the image to the zero-padded extension induces high-frequency noise [9]. This can be solved by performing a background subtraction on the images or by using the circular

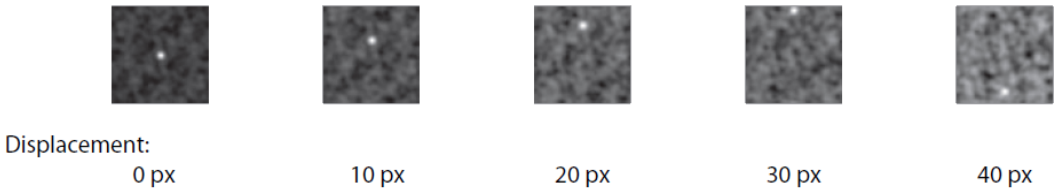


Figure 2.8: The peak position in for different displacement based on the circular cross-correlation matrix for interrogation window size $S = 64\,\text{px}$. For the displacement $d = 40\,\text{px}$ the aliasing effect occurs [14].

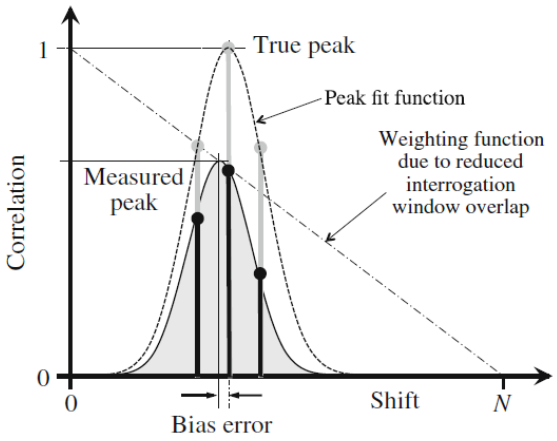


Figure 2.9: Bias effect due to less correlation data [9].

cross-correlation.

- In case the circular cross-correlation is used to determine the displacement, the maximum displacement must not exceed half of the interrogation window size S_w. Otherwise, the position of the correlation peak position is effected by aliasing as displayed in figure 2.8.

- Furthermore, the displacement should not exceed $\frac{1}{4}$ of the interrogation window size S_w since the measurement uncertainty drastically increases [7].

- Also due to the periodicity of the correlation data, the correlation peak position is biased towards a more central position because fewer samples contribute to the outside neighbor of the correlation peak than to the inner neighbor. The effect is shown in figure 2.9. This can be solved by applying a weighting function [9].

2.2.1 Sub pixel estimation

A way to surpass the integer pixel locking is the subpixel estimation which takes advantage of the shape of the correlation peak. For that, not only the correlation peak but also the neighboring pixels are used since their correlation value will most likely be significantly higher than the background value assuming that the size with what the particles are represented on the camera sensor is about two to three pixels or greater. These properties of the neighboring pixels in the correlation matrix allows to fit a function through the correlation values as shown in figure 2.9. This can be for example a Gaussian or a Centroid function. The position of the correlation peak can now be estimated by the maximum of the fitting function which is in many cases a closer estimation than the integer peak position and can be processed as described above [9].

2.3 Multi pass, Grid Refinement and Window Deformation

2.3.1 The Multipass algorithm

The Multipass algorithm is an advanced technique developed to increase the accuracy and and dynamic range of the PIV evaluation. This is achieved by performing multiple iterations of the PIV interrogation which allows the application of a Multigrid analysis as well as the usage of the window deformation technique.

2.3.2 Iterative Multigrid analysis

Because of the increasing loss-of-pair due to the in-plane movement of the particles – the signal to noise ratio decreases for larger displacements, which again increases the measurement uncertainty. Therefore, the displacement should be limited to $\frac{1}{4}$ of the interrogation window size [7]. This means, that for greater displacement larger interrogation windows are required to provide the dynamic range. On the other hand, larger interrogation windows result in a reduced spatial resolution of the vector field since each interrogation window provides just one displacement vector. Hence, one must deal with a trade-off between the spatial resolution of the vector field and the dynamic range [9].
This is the point where the Iterative Multigrid approach [10] comes into play (Fig. 2.10). The Iterative Multigrid approach performs multiple passes of PIV processing. For every pass the validated displacement of the previous pass is used as a predictor to offset

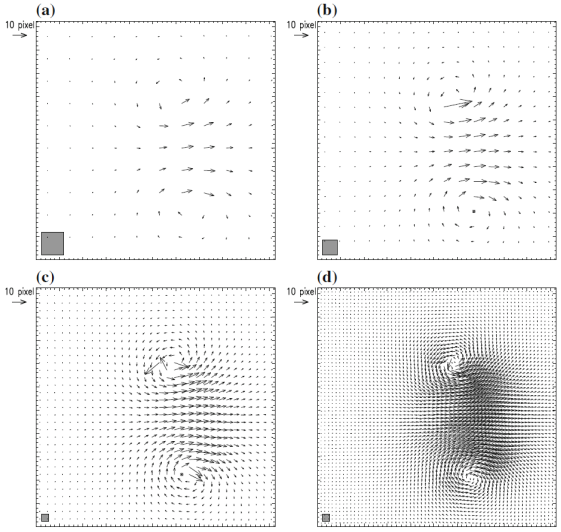


Figure 2.10: Increase of spatial resolution due to iterative grid refinement [9].

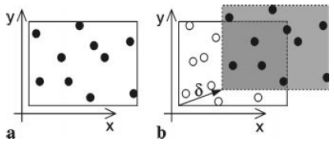


Figure 2.11: Visualization of the window displacement [10].

the interrogation windows of the second image accordingly as shown in figure 2.11 and equation (2.6) [10]. $\delta^k(x,y)$ is the resulting, $\delta_p^k(x,y)$ the predicted and $\delta_c^k(x,y)$ corrective displacement of pass k depending on the lateral dimensions x and y.

$$\begin{aligned} \delta^k &= \delta_p^k(x,y) + \delta_c^k(x,y) \\ \text{with } \delta_p^{k+1} &= \text{ integer } \left(\delta^k\right) \end{aligned} \tag{2.6}$$

Due to the offset of the interrogation windows, only the first pass, when there is no displacement prediction available, is limited by the one-quarter rule. Hence, the spatial resolution of the displacement vector field can be increased by choosing smaller interrogation windows for the succeeding pass.

2.3.3 Window Deformation

A basic assumption in PIV is that the velocity inside the interrogation windows must be close to uniform while most flows, especially vortices and turbulent flows, do not satisfy this condition. The high velocity gradients induced by these flows result in a decreased

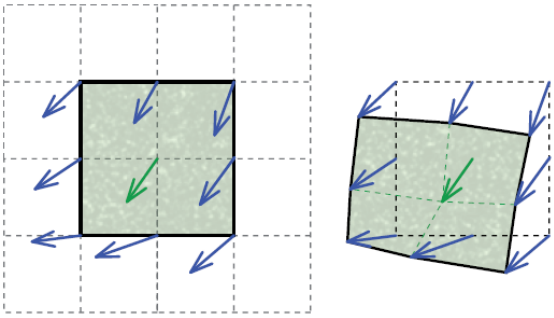


Figure 2.12: Window deformation based on the additional displacement information provided by the neighboring interrogation windows ([15]).

signal to noise ratio and can broaden the correlation peak, corrupting the measurement. To account for the negative effects caused by the in-plane velocity gradients, the window deformation technique was introduced [11]. The window deformation technique is about deforming the interrogation windows according to the velocity gradients in the window. In fact, the window deformation is often implemented by deforming the image instead of the interrogation windows [9]. As for the Multigrid approach, the displacement information of the previous pass is used to modify the PIV setup of the succeeding pass. For a better understanding, the principle of the window deformation is explained in an example. Assuming a window overlap of 50 percent, no image borders and constant window size, a total of nine displacement information, four in the corners, four in the middle of each edge and one at the center, are available for every interrogation window that can be used to deform the window for the next pass, as shown in figure 2.12. The deformation is done by interpolating the pixel values of the image on to a new grid that is defined by the old image grid combined with the displacement information. The interpolation method used has a huge impact on the measurement results. Therefore, it is important to use an adequate interpolation method. In practice, the B-spline interpolation has proven to be an appropriate choice for most cases [2].

In addition to better robustness in turbulent flows, the peak locking effect is drastically reduced due to the pixel value interpolation [11]. The whole process of the Multipass processing with Grid Refinement and Window Deformation is shown in figure 2.13.

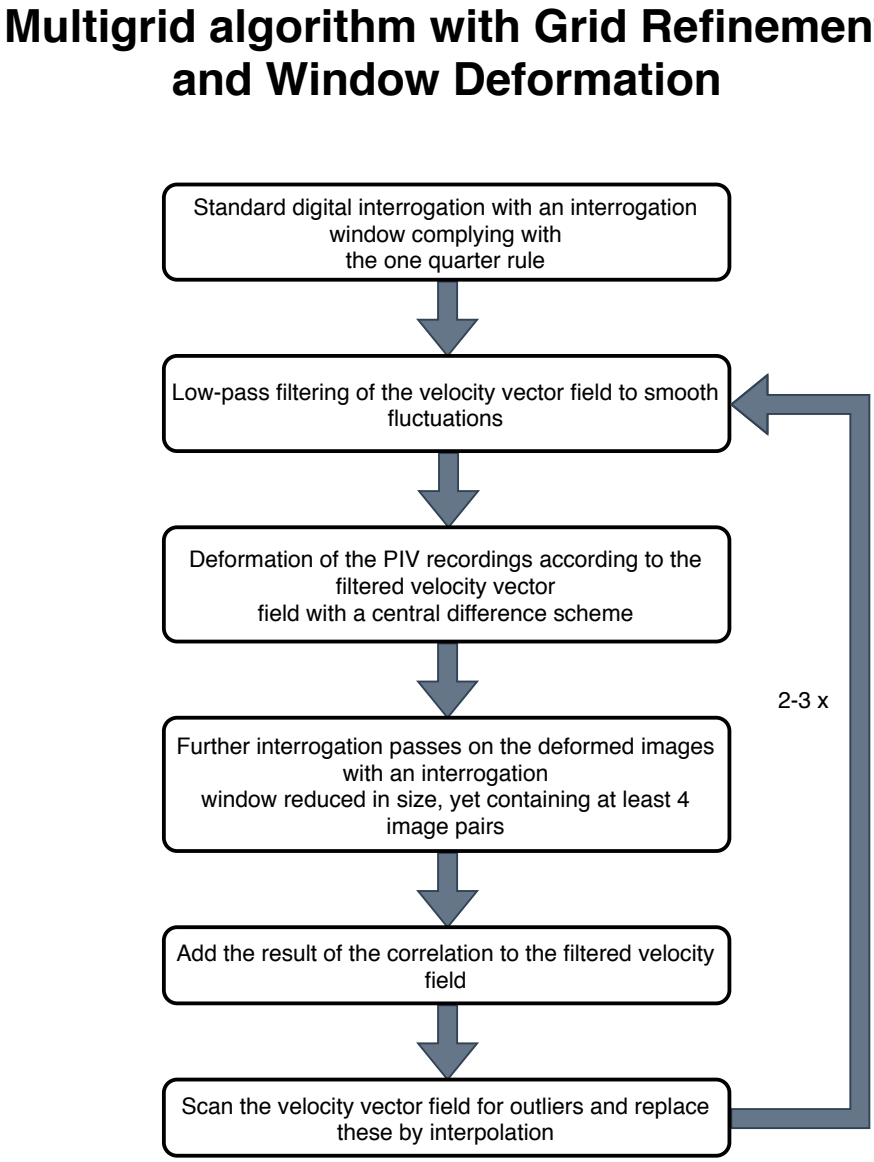


Figure 2.13: Flowchart of Multipass algorithm with Grid Refinement and Image Deformation based on [9].

Chapter 3

Advanced PIV interrogation using the OpenPIV Python package

The implemented advanced interrogation techniques became part of the OpenPIV Python package [8]. One can install it using pip or download it from GitHub here. The Iterative Multigrid and Window Deformation Schemes algorithm uses a Python version of the"smoothn" algorithm from D. Garcia [5]. The easiest way to set up and run a PIV analysis using advanced interrogation techniques is to open the example_windef_run.py file and enter the desired parameter.

3.1 The example_windef_run.py

The example_windef_run.py is a setup file for the PIV interrogation. After modifying the parameters according to one's needs one just runs the script and the results are returned in the specified folder. The available options are:

Data related settings

- **settings.filepath_images:** (type: string)
  Here the file path to the folder that contains the images must be added. Note: If one uses the absolute file path on a windows system ":\" can result in an error. This can be solved by adding a second backslash like this ":\\".

- **settings.save_path:** (type: string)
  Here the file path must be inserted at which a folder with the results should be created. Note: Be aware about the absolute file path issue mentioned above.

- **settings.save_folder_suffix:** (type: string)
  Here a suffix can be added to customize the name of the results folder. The save folder is named like:
  "OpenPIV_results_"+*window size of the last pass*+"_"+*settings.save_folder_suffix*

- **settings.frame_pattern_a and settings.frame_pattern_b:** (type: string)
  These parameters should contain the pattern of the image's names. An Example: Assuming 200 images structured as 100 double frames and the images names to be like A000a-A099a and A000b-A099b. In this case one would like to correlate the image A000a with the image A000b and so on until all the images are evaluated. This is automatized by using a pattern. For this example, the **settings.frame_pattern_a** would look like **A*a** and **settings.frame_pattern_b** would look like **A*b**. The asterisk is replacing the frame number.

Region Of Interest

- **settings.ROI:** (type: string or tuple with four int entries)
  This option allows to select a certain Region Of Interest of the images. The **ROI** is a square defined by the pixel coordinates like this **(xmin, xmax, ymin, ymax)** with the image origin in the upper left corner of and the x-axis going to the right and the y-axis going downwards. An example ROI could look like this (50, 300, 50, 300). Be aware that the ROI must not exceed the size of the image. In case one would like to evaluate the whole image, one can replace the tuple with the string 'full'.

Image preprocessing

- **settings.dynamic_masking_method:** (type: string) This option enables the image preprocessing. One can choose between 'None', 'edges' and 'intensity' If 'None' is selected no preprocessing is done. If 'edges' is selected a filter is applied which blurs out edges in the image. If 'intensity' is selected, larger image areas which intensity is above a threshold defined by **settings.dynamic_masking_filter_size:** are masked out. A description of this tool is provided here.

- **settings.dynamic_masking_filter_size:** (type: int) This parameter defines the size of the filter used for the image prepossessing.

- **settings.dynamic_masking_threshold:** (type: float) Here one can insert the threshold for the image prepossessing.

Processing parameters

- **settings.correlation_method:** (type: string) One can choose between to different cross-correlation options. The options are 'circular' and 'linear'. If 'circular' is select circular cross-correlation is done which is faster but slightly less accurate. If 'linear' is the displacement evaluation is done using linear cross-correlation which is slower.

- **setting.iterations:** (type: int)
  This parameter determines the number of PIV-iterations. The minimum is one iteration. Advanced interrogation techniques require at least two iterations.

- **settings.windowsizes:** (type: tuple of int)
  (settings.windowsizes) allows to define the window size for each PIV-iteration. The tuple needs at least as many entries as iterations performed. The first iteration of the PIV evaluation is done using the window size with the index 0 and so on. Only the first pass is affected by the one-quarter rule and the window size of the following passes can be reduced to increase the dynamic range and the spatial resolution. The window size should be a value with base two. Example: (settings.windowsizes)=(64,32,16). Observe that if the tuple introduced is longer than the number of iterations, the last values will be ignored.

- **settings.overlap:** (type: tuple of int)
  settings.overlap allows one to define the overlap of the interrogation windows for each PIV-iteration. The tuple needs at least as many entries as iterations performed. In general, an overlap of half the interrogation window size is a solid choice for most cases. They overlap should be a value with base two. Example: (settings.overlap)=(32,16,8).

- **settings.subpixel_method:** (type: string)
  This option allows one to select a fitting function for the subpixel estimation of the correlation peak. One can choose between 'gaussian', 'centroid' and 'parabolic'. By default, 'gaussian' is used.

- **settings.interpolation_order:** (type: int)
  settings.interpolation_order defines the order of interpolation used for the image deformation. It should be between three and five. Only used if more than one iteration is executed.

- **settings.scaling_factor:** (type: int)
  The scaling factor is used to transform the displacement from $\frac{pixel}{frame}$ to $\frac{meter}{frame}$. It describes the relationship between the measurement plane and the camera sensor. If no information is available, it is better to set this value to 1 and then leave the measurement scaling to the post-processing stage.

- **settings.dt:** (type: int)
  This parameter defines the time between the frames. It transforms the displacement from $\frac{pixel}{frame}$ to $\frac{pixel}{second}$. In combination with the scaling factor, this results in a velocity. If no information is available, as for the previous parameter, it is worth setting it to 1.

Signal to noise ratio options (only for the last pass)

- **setting.extract_sig2noise:** (type: bool)
  This setting lets one choose to extract the signal to noise ratio and export it. Observe that the signal to noise ratio is only calculated for the last pass. It must be enabled to do the signal to noise ratio validation. If this is set to false, the signal to noise ratio column in the output is filled with NaNs.

- **setting.sig2noise_method:** (type: string)
  Here one can choose between the method 'peak2peak' and 'peak2mean' to determine the signal to noise ratio. 'peak2peak' calculates the signal to noise ratio by dividing the correlation value of the largest peak by the correlation value of the second largest peak. 'peak2mean' calculates the signal to noise ratio by dividing the largest peak by the averaged correlation value of the interrogation window.

- **settings.sig2noise_mask:** (type: int)
  This parameter defines the size of the mask around the first peak. It only affects the calculation of the signal to noise ratio if the option 'peak2peak' is chosen. 'peak2peak' calculates the signal to noise ratio by dividing the value of the largest correlation peak with the value of the second largest peak. In the case of very noisy cross-correlation maps, this function avoids that the second correlation peak is too close the first. This is done by using a circled mask centered in the first peak. This parameter is the radius of such a mask. By default, two is a sufficient value.

Vector validation options

- **setting.validation_first_pass** (type: bool)
  In the multi-pass approach, the first pass is always validated before proceeding to the following ones. If Singlepass is chosen, the validation can be disabled by setting this parameter to false.

  Validation by minimal and maximal displacement

  - **settings.MinMax_U_disp and settings.MinMax_U_disp:** (type: tuple of float)
    This option allows one to set a limit for the u and v displacement. The values have to be inserted into a tuple like this (min, max). Displacement vectors that exceed these limits will be marked as invalid. In case this kind of validation is not desired, a very large value can be inserted. Important: the input for this validation is in $\frac{pixel}{frame}$.

  Validation by standard deviation

  - **settings.std_threshold:** (type: float)
    This parameter sets the threshold for the validation by the global standard deviation. Vectors are discarded if they differ from the global mean by a certain value, in either of the components, which is larger than the specified number of standard deviations. A sufficient value has to be chosen according to the circumstances. Also, this validation can be disabled by entering very large thresholds.

  Validation by local median

  - **settings.median_threshold:** (type: float)
    This parameter sets the threshold for the validation by the local median. For that, the local median of the u and v displacement is calculated independently. In case the difference between the actual value and its local median exceeds the threshold, the vector is masked as invalid. This validation is similar to the popular median test but has no normalization. Like the others, it can be disabled by choosing a large value.

  Validation by signal to noise ratio

  - **settings.do_sig2noise_validation:** (type: bool)
    This enables or disables the validation by the signal to noise ratio by setting it to True or False.

It is only available for the last pass and **setting.extract_sig2noise** has to be enabled.

- **settings.sig2noise_threshold:** (type: float)
  Sets the threshold for the signal to noise ratio. Displacement vectors that exceed this threshold are marked as invalid and removed.

Outlier replacement or Smoothing options

- **settings.replace_vectors:** (type: bool)
  This option enables or disables the outlier replacement by setting it to True or False. This is only valid for the last pass.

- **settings.smoothn:** (type: bool)
  This option enables or disables the smoothing of the velocity fields. This option uses the famous smoothing function from Garcia, also used in PIVlab. It can be of interest when spatial derivatives will be calculated on the instantaneous velocity fields.

- **settings.smoothn_p:** (type: float)
  This is the smoothing parameter from Garcia's function. Low values produce little smoothing, high values produce strong smoothing.

- **settings.filter_method:** (type: string)
  This parameter stipulates the method used to replace the outlier vectors. One can choose between 'localmean', 'disk' and 'distance'. 'localmean' replaces the invalid vector by the mean of a local square area around the invalid vector. 'disk' replaces the invalid vector by the mean of a local circle around the invalid vector. 'distance' uses also a circular area around the invalid vector but weights the values according to their distance from the center.

- **settings.max_filter_iteration:** (type: int)
  settings.max_filter_iteration defines the maximum number of filter iterations. This kind of filter, documented in the OpenPIV package, is important when a cluster of outliers is produced. In this case, the replacement starts from the boundaries of the field and proceed towards the center using a number of iterations defined by this parameter. If this value is set too high, the value used to replace the invalid vector can significantly diverge from the actual displacement and it might be worth considering different parameters for the analysis.

- **settings.filter_kernel_size** (type: int) Here one can define the size of the area used to calculate the vector replacement. Default: 2

Output options

- **settings.save_plot:** (type: bool) Decide whether one wants to save the plotted vector fields by choosing between True and False. They are stored in the same folder as the .txt files containing the results.
- **settings.show_plot:** (type: bool) Decide whether one wants to see the plotted vector fields by choosing between True and False. Depending on the interrogation setting the vector field may be instantly replaced by the following vector field.
- **settings.scale_plot:** Defines the arrow size of the plotted vector field. Larger values result in smaller arrows.

The code of the script is shown below:

```
from openpiv import windef


settings = windef.Settings()


'Data related settings'
# Folder with the images to process
settings.filepath_images = r'C:\Users\Theo\Desktop\VKI\Validaton\PIV_challenge_2003_A'
# Folder for the outputs
settings.save_path = r'C:\Users\Theo\Desktop\VKI\Validaton\Vectorfields_2003A'
# Root name of the output Folder for Result Files
settings.save_folder_suffix = 'test'
# Format and Image Sequence
settings.frame_pattern_a = 'A*a.tif'
settings.frame_pattern_b = 'A*b.tif'

'Region of interest'
# (50,300,50,300) #Region of interest: (xmin,xmax,ymin,ymax) or 'full' for full image
settings.ROI = 'full'

'Image preprocessing'
# 'None' for no masking, 'edges' for edges masking, 'intensity' for intensity masking
# WARNING: This part is under development so better not to use MASKS
settings.dynamic_masking_method = 'None'
settings.dynamic_masking_threshold = 0.005
settings.dynamic_masking_filter_size = 7
```

```
'Processing Parameters'
settings.correlation_method='circular'  # 'circular' or 'linear'
settings.iterations = 2  # select the number of PIV passes
# add the interroagtion window size for each pass.
# For the moment, it should be a power of 2
settings.windowsizes = (64, 32, 16) # if longer than n iteration the rest is ignored
# The overlap of the interroagtion window for each pass.
settings.overlap = (32, 16, 8) # This is 50% overlap
# Has to be a value with base two. In general window size/2 is a good choice.
# methode used for subpixel interpolation: 'gaussian','centroid','parabolic'
settings.subpixel_method = 'gaussian'
# order of the image interpolation for the window deformation
settings.interpolation_order = 3
settings.scaling_factor = 1  # scaling factor pixel/meter
settings.dt = 1  # time between to frames (in seconds)
'Signal to noise ratio options (only for the last pass)'
# It is possible to decide if the S/N should be computed (for the last pass) or not
settings.extract_sig2noise = False  # 'True' or 'False' (only for the last pass)
# method used to calculate the signal to noise ratio 'peak2peak' or 'peak2mean'
settings.sig2noise_method = 'peak2peak'
# select the width of the mask to mask out pixels next to the main peak
settings.sig2noise_mask = 2
# If extract_sig2noise==False the values in the signal to noise ratio
# output column are set to NaN
'vector validation options'
# choose wether you want to do validation of the first pass: True or False
settings.validation_first_pass = True
# only effecting the first pass of the interrogation the following passes
# in the multipass will be validated
'Validation Parameters'
# The validation is done at each iteration based on three filters.
# The first filter is based on the min/max ranges. Observe that these values are defined in
# terms of minimum and maximum displacement in pixel/frames.
settings.MinMax_U_disp = (-10, 5)
settings.MinMax_V_disp = (-5, 5)
# The second filter is based on the global STD threshold
settings.std_threshold = 7  # threshold of the std validation
# The third filter is the median test (not normalized at the moment)
settings.median_threshold = 5  # threshold of the median validation
# On the last iteration, an additional validation can be done based on the S/N.
settings.median_size=1 #defines the size of the local median
'Validation based on the signal to noise ratio'
# Note: only available when extract_sig2noise==True and only for the last
# pass of the interrogation
# Enable the signal to noise ratio validation. Options: True or False
settings.do_sig2noise_validation = False # This is time consuming
# minmum signal to noise ratio that is need for a valid vector
settings.sig2noise_threshold = 1.2
```

```
'Outlier replacement or Smoothing options'
# Replacment options for vectors which are masked as invalid by the validation
settings.replace_vectors = True # Enable the replacment. Chosse: True or False
settings.smoothn=False #Enables smoothing of the displacemenet field
settings.smoothn_p=0.1 # This is a smoothing parameter
# select a method to replace the outliers: 'localmean', 'disk', 'distance'
settings.filter_method = 'localmean'
# maximum iterations performed to replace the outliers
settings.max_filter_iteration = 4
settings.filter_kernel_size = 2  # kernel size for the localmean method
'Output options'
# Select if you want to save the plotted vectorfield: True or False
settings.save_plot = False
# Choose wether you want to see the vectorfield or not :True or False
settings.show_plot = True
settings.scale_plot = 500  # select a value to scale the quiver plot of the vectorfield
# run the script with the given settings

windef.piv(settings)
```

Chapter 4

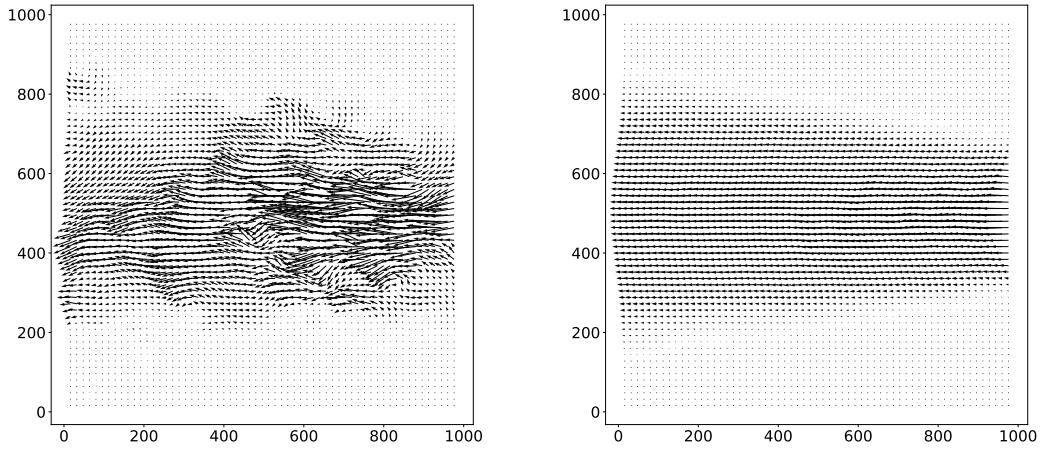
Validation of the developed code

In this chapter, the newly implemented Iterative Multigrid with Image Deformation Schemes is compared with PIVlab [15]. PIVlab has proven its accuracy and robustness extensively and was already used to provide measurement results for many scientific publications. Furthermore, PIVlab already uses the similar advanced interrogation techniques to those implemented in this work. Hence, it is well suited to be used as a reference for validation of the newly implemented advanced interrogation techniques. The comparison is based on image sets of the Second International PIV Challenge [12]. Since PIVlab and OpenPIV are not using the same grid, the comparison is performed based on velocity profiles. The difference of the grid results from the fact that PIVlab is centering the interrogation grid one the image while OpenPIV is not.

4.1 The Second International PIV Challenge Case A

The validation of the newly implemented iterative Multigrid and Window Deformation Schemes is performed by using images of the Test Case A of the Second International PIV Challenge. This case consists of 100 real images of a self-similar turbulent jet flow. An averaged and an instantaneous vector field, processed using the presented algorithm, is shown in figure 4.1 to give the reader a quick impression of the flow. For further details concerning the experimental setup, the reader is referred to [4].

The usage of experimental images is advised, since the aim of this validation is to prove the suitability of the implemented schemes to evaluated experimental data rather than running synthetic test cases. Using experimental data not only allows for evaluating the accuracy of the algorithm, but proof its robustness under real conditions including non-perfectly shaped particles, background noise, alignment errors of the light sheet,



(a) Instantaneous vector field (b) Time averaged vector field

Figure 4.1: Vector fields of the turbulent jet flow.

etc. Also, due to the well-known statistical properties of self-similar turbulent jet flows, it is simple to compare the measurement results of different PIV algorithms even though they are evaluating on different grids.
For the comparison, three passes of PIV interrogation were performed with window sizes of 64, 32 and 16 pixels. The overlap of the interrogation window remained 50 % throughout the whole iterations. The whole processing was done without any image preprocessing to test the robustness of only the PIV evaluation. The cross-correlation is done in the frequency domain using an FFT algorithm and the circular cross-correlation scheme to benefit from the significantly faster calculation of the correlation map. The used sub-pixel estimation is based on a 2 x 3-point Gauss approach. After the PIV interrogation, the vectors are validated using global limits, a standard deviation test, and a median test. After the validation, the invalid vectors are replaced by outliers replacing methods.

Figure 4.2 shows the self-similar averaged profile of the turbulent jet flow evaluated with the iterative grid refinement and window deformation method. The profiles are extracted on four different x-locations namely 200, 400, 600, 800 pixels with the x-origin on the left image side. The figure shows that the similiarized profiles fit decently which indicates that the implemented Iterative Multigrid and Window Deformation Schemes is capable of displaying the characteristics of the flow at various positions and therefore the

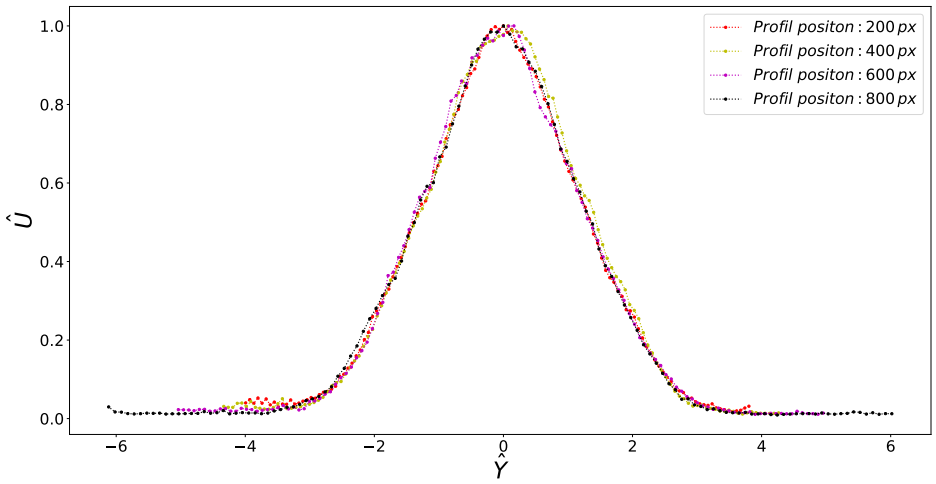


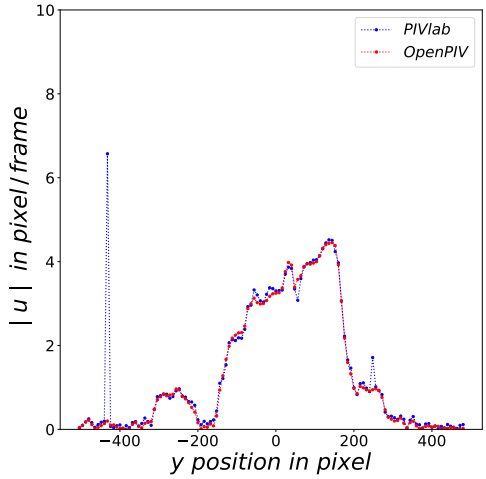
Figure 4.2: Self-similar profiles of the turbulent jet flow at different x-locations

comparison between the implemented algorithm and PIVlab can be reduced to significant positions.

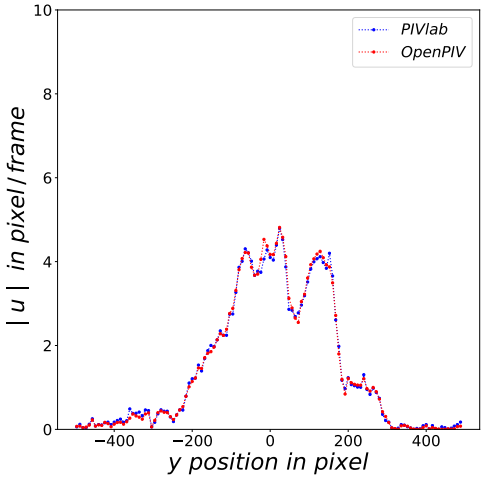
In figure 4.3 for pairs of instantaneous displacement profiles are shown. The absolute value of the horizontal displacement is plotted over the y-axis. The y-axis is centered around the y-coordinate of the maximum u displacement. The comparison between PIVlab and the Iterative Multigrid with Image Deformation schemes algorithm shows that the instantaneous displacement profiles match very well over the whole profile range.

In addition to the u-displacement profiles in figure 4.3, figure 4.4 compares the vertical displacement profiles of OpenPIV and PIVlab at different x-locations. Again, as before the y-axis is centered around the y-coordinate of the maximum displacement of the Gauss function of the averaged u-displacement. Comparing the profiles shows the consistency of results of PIVlab and OpenPIV apart from a few outlying vectors. The outliers might be reduced if image preprocessing is done beforehand the PIV processing. The image preprocessing was skipped on the purpose to just compare the results of the PIV algorithms itself.

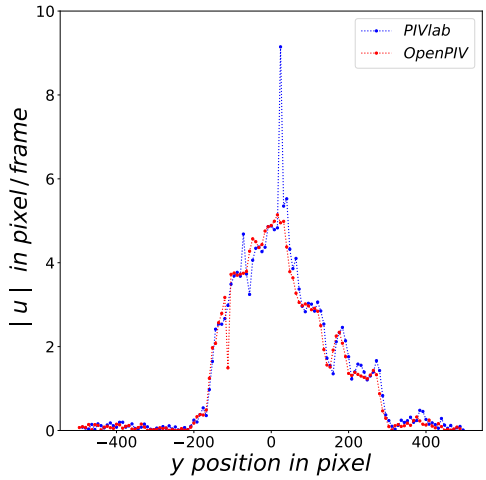
All this proves that the new OpenPIV package including the Iterative Multigrid and Window Deformation Schemes is performing satisfactorily and compares well with the established PIVlab tool, while not relying on a programming language that requires a commercial license.



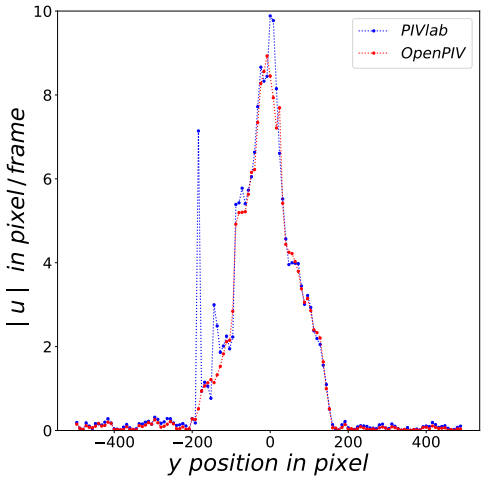
(a) u-displacement profile at x-position around pixel 200.



(b) u-displacement profile at x-position around pixel 400.

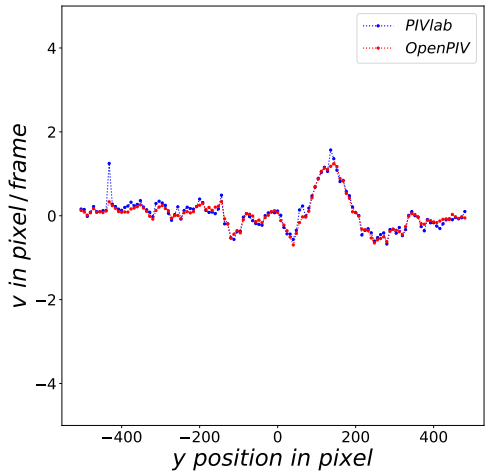


(c) u-displacement profile at x-position around pixel 600.

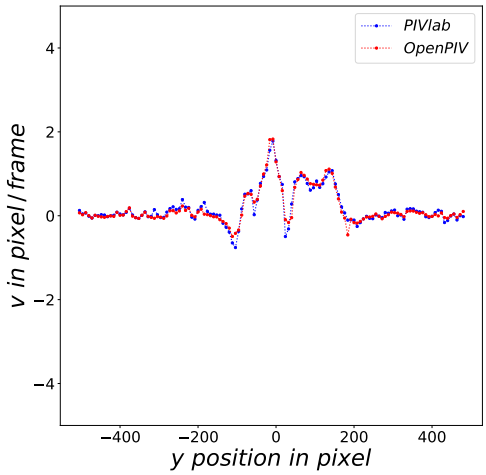


(d) u-displacement profile at x-position around pixel 800.

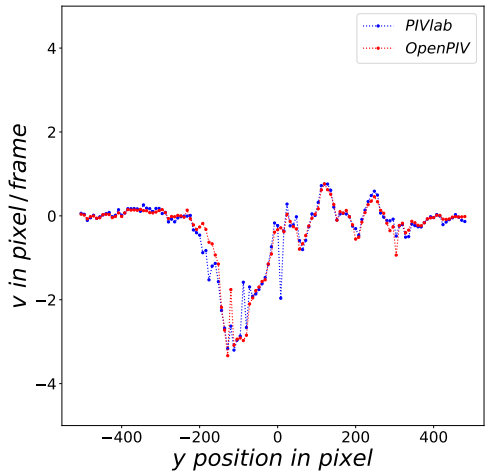
Figure 4.3: Comparison of instantaneous absolute u-displacement profiles at different x-positions. The profiles are centered around the y-coordinate of the maximum displacement.



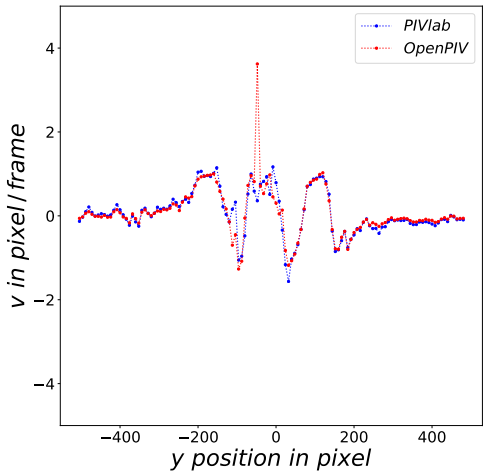
(a) v-displacement profile at x-position around pixel 400.



(b) v-displacement profile at x-position around pixel 400.



(c) v-displacement profile at x-position around pixel 600.



(d) v-displacement profile at x-position around pixel 800.

Figure 4.4: Comparison of instantaneous v-displacement profiles at different x-positions. The profiles are centered around the y-coordinate of the maximum displacement.

Chapter 5

Conclusion

In this work, different methods of cross-correlation, namely linear and circular cross-correlation are explained and compared using one-dimensional examples. Methods for their fast calculation in the Discrete Frequency Domain using FFT algorithms and their limits for application for displacement estimation in PIV are discussed. In general, both linear and circular cross-correlation can be used to determine the particle displacement. Circular cross-correlation is calculated significantly faster, while the linear cross-correlation with background normalization tends to be more robust against outliers. Furthermore, advanced PIV interrogation techniques were implemented into Python code and added to the OpenPIV Python package. It is contributed under open source license and available for anyone to use.
Also, instructions on how to set up and run a PIV interrogation using the example_windef_run.py script have been provided. In the end, a validation based on the Test Case A of the Second International PIV challenge was done. For that, the results of the new OpenPIV package were compared with the results from PIVlab. It was shown that the implemented method performs satisfactorily and compares well with PIVlab.

References

[1] R. J. Adrian. Twenty years of particle image velocimetry. *Experiments in Fluids*, 39(2):159–169, Aug 2005.

[2] T. Astarita and G. Cardone. Analysis of interpolation schemes for image deformation methods in piv. *Experiments in Fluids*, 38(2):233–243, Feb 2005.

[3] Christian Cierpka, Rainer Hain, and Nicolas A Buchmann. Flow visualization by mobile phone cameras. *Experiments in Fluids*, 57(6):108, 2016.

[4] C Fukushima, L Aanen, and Jerry Westerweel. Investigation of the mixing process in an axisymmetric turbulent jet using PIV and LIF. In *Laser techniques for fluid mechanics*, pages 339–356. Springer, 2002.

[5] Damien Garcia. Robust smoothing of gridded data in one and higher dimensions with missing values. *Computational statistics & data analysis*, 54(4):1167–1178, 2010.

[6] Monson H Hayes. *Schaum's outline of digital signal processing*. McGraw-Hill, Inc., 1998.

[7] Richard D Keane and Ronald J Adrian. Optimization of particle image velocimeters. i. double pulsed systems. *Measurement science and technology*, 1(11):1202, 1990.

[8] Alex Liberzon, Davide Lasagna, Mathias Aubert, Pete Bachant, jakirkham, ranleu, tomerast, Theo Kaeufer, Joe Borg, Cameron Dallas, and Boyko Vodenicharski. OpenPIV/openpiv-python: added synthetic image generator, October 2019.

[9] Markus Raffel, Christian E Willert, Fulvio Scarano, Christian J Kähler, Steve T Wereley, and Jürgen Kompenhans. *Particle image velocimetry: a practical guide*. Springer, 2018.

[10] Fulvio Scarano and Michel L Riethmuller. Iterative multigrid approach in PIV image processing with discrete window offset. *Experiments in Fluids*, 26(6):513–523, 1999.

[11] Fulvio Scarano and Michel L Riethmuller. Advances in iterative multigrid PIV image processing. *Experiments in Fluids*, 29(1):S051–S060, 2000.

[12] Michel Stanislas, Koji Okamoto, Christian J Kähler, and Jerry Westerweel. Main results of the second international piv challenge. *Experiments in fluids*, 39(2):170–191, 2005.

[13] Zachary J Taylor, Roi Gurka, Gregory A Kopp, and Alex Liberzon. Long-duration time-resolved PIV to study unsteady aerodynamics. *IEEE Transactions on Instrumentation and Measurement*, 59(12):3262–3269, 2010.

[14] William Thielicke. *The flapping flight of birds: Analysis and application*. University of Groningen, 2014.

[15] William Thielicke and Eize Stamhuis. PIVlab–towards user-friendly, affordable and accurate digital particle image velocimetry in MATLAB. *Journal of Open Research Software*, 2(1), 2014.

[16] Eric W. Weisstein. *Convolution Theorem*. MathWorld–A Wolfram Web Resource, Date: 21.8.2019 http://mathworld.wolfram.com/ConvolutionTheorem.html.