INTRODUCTION TO MPIV
– PIV toolbox in MATLAB –
version 0.97

Nobuhito Mori [*] and Kuang-An Chang [†]

July 2, 2009

1 Brief Introduction

'mpiv' is a PIV (Particle Image Velocimetry) toolbox written in MATLAB. The main purpose of this toolbox is for educational and research. It is intended for both the undergraduate and graduate students working on PIV methods, or need to use PIV for their study. It is also a convenient and useful tool for engineers and scientists who use PIV in their research. Writing the program in MATLAB is, in our opinion, believed to be easier to understand and use, although it may be slower in computation. The project started in Fall 2002, and since then several algorithms have been modified and added. Moreover several options have be included for users of different levels. mpiv has also been tested and proved to be sufficient accuracy and robust.

There are many PIV programs existing today. Some of the programs are written in MATLAB and already available to users (*e.g.*, URAPIV and MatPIV). These programs have excellent capability for research propose and have many options for users of different level and optimization for fast computation. In comparison, mpiv is more general and portable, and easy to use and modify. It is the developers' intention to keep the codes short and simple. In brief, mpiv consists of two main programs (one for image processing and the other for post-processing) and several external functions (.m files) while most of them are very short and are based on simple algorithms. Thus, not only are the algorithms and codes of mpiv easy to understand for undergraduate and graduate students, they are also quite general and easy to modify for use in specific research area for engineers and scientists.

The present version of mpiv has the following features:

- Cross-correlation and MQD (Minimum Quadratic Difference)
- Super-resolution (recursive/hierarchical) algorithm
- Local mean, median and global filtering
- Vector interpolation using linear, spline and Kriging methods
- Simple GUI menu is available

[*] Department of Fluid Mechancis, Central Research Insititute of Electric Power Industry (CRIEPI), 1646 Abiko, Abiko city, Chiba 270-1194, Japan. email:mori@criepi.denken.or.jp

[†] Department of Civil Engineering, Texas A&M University, College Station, Texas 77843-3136, USA. email:kchang@tamu.edu

- Free of charge

To run the program, you need the followings:

- MATLAB version 6 (Release 12) or later
- MATLAB Image Processing Toolbox
- DACE (Design and Analysis of Computer Experiments) Toolbox

Note that the earlier versions of MATLAB may work fine as well though we did not test the program on them, and MATLAB Image Processing and DACE Toolbox are necessary to run the program. Image Processing Toolbox is the one of Toolbox developed by Mathworks, while DACE Toolbox (see next section for more details) is a BSD based toolbox and can be downloaded from http://www.imm.dtu.dk/~hbn/dace/. In addition, you can use MATLAB Compiler to reduce computational time, but it is not a must. The original images in this document are taken from VSJ-PIV (the PIV Standard Project) distributed by the Visualization Society of Japan (http://www.vsj.or.jp/).

mpiv has been tested (MATLAB 6.5.0.180913a, R13) on Linux platform running on Vine Linux 2.6 (http://www.vinelinux.org/), equivalent to Red Hat Linux 7.2, and Microsoft Windows 2000.

2 Getting Start with mpiv

2.1 Installation

To run mpiv, MATLAB with Image Processing Toolbox is required. MATLAB Compiler is optional, it accelerates computational speed more than 30-50%. The mpiv toolbox can be unzipped using tar or unzip command. You may just type in:

```
>> tar zxvf mpiv_toolbox.tgz
or
>> unzip mpiv_toolbox.zip
```

The mpiv package includes the following files in a single directory:

```
mpiv_gui.fig
func_findpeak2.m
func_histfilter.m
func_pivwindowsize.m
func_smooth.m
mpiv.m
mpiv_filter.m
mpiv_gui.m
mpiv_smooth.m
piv_cor.m
piv_crr.m
piv_crs.m
piv_mqd.m
piv_mqr.m
piv_mrs.m
```

```
vector_check.m
vector_exterp_linear.m
vector_filter_global.m
vector_filter_median.m
vector_filter_vecstd.m
vector_interp.m
vector_interp_kriging.m
vector_interp_kriging_local.m
vector_interp_linear.m
vector_interp_spline.m
vector_interp_NaN.m
image1.bmp
image2.bmp
```

In the package, mpiv.m is the main program of mpiv for image processing and velocity computation, mpiv_filter.m and other mpiv_XXX.m are the programs for post-processing, piv_XXX.m files are the main functions to compute velocity vectors using different algorithms, and vector_XXX.m are the functions to interpolate, extrapolate and validate velocity vectors. The rest of programs are external functions called by mpiv.m and other programs. All .m files have to be placed in the same directory or on the path of MATLAB. Two image files (image1.bmp and image2.bmp) taken from VSJ-PIV Standard Project are for testing mpiv.

mpiv requires DACE (Design and Analysis of Computer experiments) Toolbox. DACE Toolbox was developed by Hans Bruun Nielsen, Soren Nymand Lophaven and Jacob Sondergaard at Technical University of Denmark. Download the toolbox from http://www.imm.dtu.dk/ hbn/dace/, unzip it, and put all the files on the path of MATLAB (click on the 'Set Path ...' bottom under 'File', or use 'addpath' command). After these procedure, you are ready to run mpiv.

2.2 How to Use mpiv

The mpiv toolbox does not include image-reading and signal preprocessing in the program so the two sequential images (called image pair) have to be read in and background noise have to be subtracted from manually. The reason that an image-reading routine was not included and the PIV and post-processing programs are separated is for easy access and modification for pre-processing and post-processing. It is recommended that users write a simple program to read in a series of images if one needs to process more than a few images.

The first step is to read in an image pair(s). Perhaps the easiest way to read in images is using the *imread* command (see example below). After that, compute velocity vectors using mpiv.m. Finally, mpiv_filter.m is called to get rid of the stray vectors (replaced with *NaN*) and to fill the eliminated *NaN* vectors. The following is an example to demonstrate mpiv using the sample images. You need to type in the following commands:

```
>> im1 = imread('image1.bmp');
>> im2 = imread('image2.bmp');
>> [xi, yi, iu, iv] = mpiv(im1, im2, 64, 64, 0.5, 0.5, 0, 0, 1, 'cor', 2, 1 );
>> [iu_ft, iv_ft, iu_ip, iv_ip] = mpiv_filter(iu, iv, 2, 2.0, 3, 1 );
```

Figure 1 shows the calculated velocity vectors using the cross-correlation algorithm with a 64×64 pixel window size, one time recursive (reducing the window size by one half) and 50% overlap between adjacent windows. The blue arrows in the figure indicate the estimated

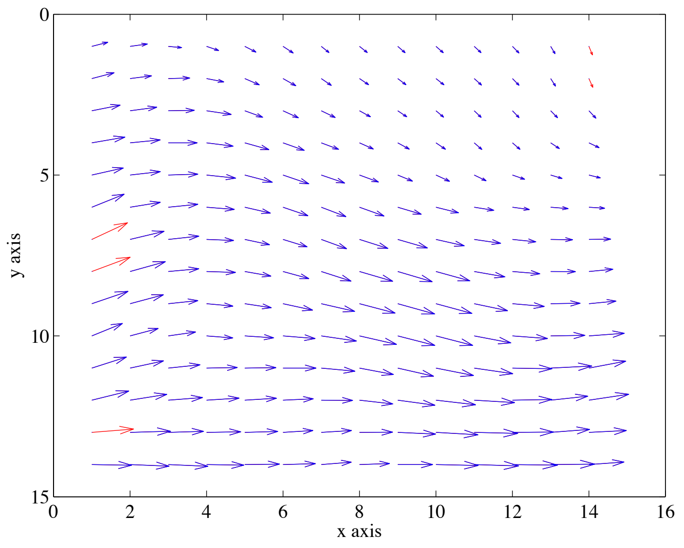


Figure 1: Result from mpiv_filter.m. Error vectors correction and interpolation: blue, original output; red, interpolated after removing error vectors.

velocity vectors while the red arrows indicate the interpolated velocity vectors that replace the eliminated spurious vectors by mpiv_filter.m.

If your MATLAB version is higher than 6.5 (R13) running on MS Windows system, GUI menu is available for demonstration.

```
>> mpiv_gui
```

GUI menu will be popped up. Please select adequate parameters for velocity estimation.

A typical flow chart for running mpiv can be summarized as follows.

1. read in sequential images (using, *e.g.*, '*imread*').
2. Estimate velocity vectors using mpiv.m.
3. Remove error vectors and replace them by interpolation using mpiv_filter.m.

That is all (and hopefully simple)!

3 Basic Principles

The present version of mpiv includes some algorithms and techniques listed below. For more details of PIV methods users should refer to Adrian (1991), Willert and Gharib (1991), and Raffel, Willert and Kompenhans (1998).

3.1 Correlation Algorithm

The cross-correlation algorithm for PIV is the most conventional method to estimate velocity vectors. The formula for two-dimensional cross-correlation of images f_1 and f_2 is as follow:

$$C(\Delta X, \Delta Y) = \frac{\sum_{i=1}^{N}\sum_{j=1}^{N}\left[f_1(X_i, Y_j) - \overline{f_1}\right]\left[f_2(X_i + \Delta X, Y_j + \Delta Y) - \overline{f_2}\right]}{\sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N}\left[f_1(X_i, Y_j) - \overline{f_1}\right]^2}\sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N}\left[f_2(X_i + \Delta X, Y_j + \Delta Y) - \overline{f_2}\right]^2}} \tag{1}$$

where f_1 and f_2 are the small windows from each image in the image pair, N is the window size and the overbar denotes mean quantity. The location of the maximum value (peak) in C is used as the mean particle displacement of this small area. In the program the calculated displacement is retained as ‘valid’ only if the ratio of the values of the highest peak to the second highest peak exceeds a preset threshold value, and the ratio of the values of the highest peak to the r.m.s.(root-mean-square) noise also exceeds a preset (determine by trial and error) threshold value.

3.2 MQD Algorithm

The Minimum Quadric Differences (MQD) algorithm is to simply calculate the pixel value differences between the search windows.

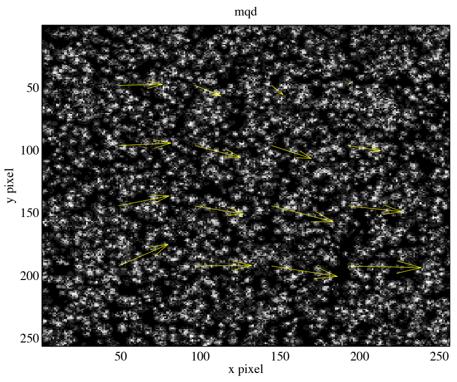
$$C(\Delta X, \Delta Y) = \sum_{i=1}^{N}\sum_{j=1}^{N}\left|f_1(X_i, Y_j) - f_2(X_i + \Delta X, Y_j + \Delta Y)\right| \tag{2}$$

The location of the minimum value in C is used as the particle displacement. Note that MQD is sometimes referred as ‘gray level difference accumulation’. The criteria to retain the calculated vectors is similar to that in the correlation algorithm, *i.e.*, by checking the ratio of the two highest peaks and the signal to noise ratio of the highest peak.

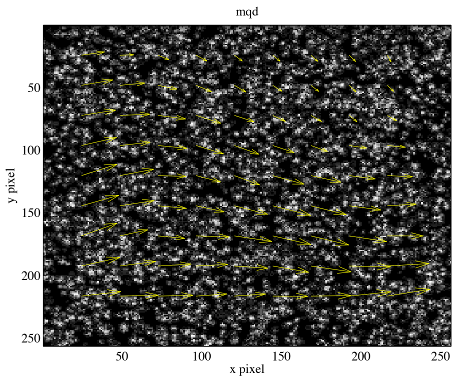
The accuracy of the cross-correlation algorithm and the MQD algorithm are almost the same. However, the MQD algorithm may be more robust than the correlation algorithm in certain situations. For example, the correlation algorithm does not work well for images containing no particles such as speckle images. However, there is a price to pay - the MQD algorithm is in general slower than the correlation algorithm.

3.3 Recursive Super-Resolution PIV

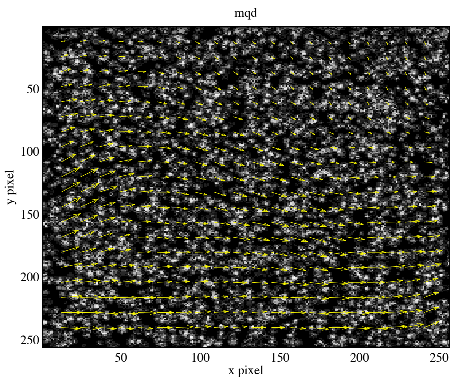
Accuracy and resolutions are the two main objectives in PIV. The so-called recursive or super-resolution PIV is used to achieved the objectives. It is based on beginning with a relatively larger window (*e.g.*, 128×128 or 64×64 pixels) and ending with a smaller window (*e.g.*, 32×32 or 16×16 pixels). Not only does the iterative process increase the spatial resolution by decreasing the window size, it also shifts the corresponding window in the second image [f_2 in equations (1) and (2)] based on the displacement calculated from previous iteration to obtain better correlation. The process therefore increases the spatial resolutions with the increased number of vectors and decreased size of window. A better-correlated result also means better accuracy in the correlation process. During the iterative process, spurious vectors are removed using median filter (discussed later). The Kriging method is used to



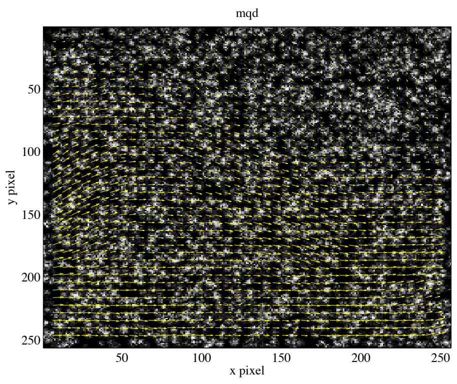
(a) 64×64 pixel



(b) 32×32 pixel



(c) 16×16 pixel



(d) 8×8 pixel

Figure 2: mpiv result from the recursive method with the 'cor' option.

interpolate the removed error vectors from the previously obtained velocity field with a larger window.

Figure 2 shows the estimated velocity vectors using the correlation algorithm (with the ‘cor’ option) with the recursive process. The MATLAB commands are listed below:

```
>> im1 = imread('image1.bmp');
>> im2 = imread('image2.bmp');
>> [xi, yi, iu, iv] = mpiv(im1, im2, 64, 64, 0.5, 0.5, 20, 20, 1,'cor', 4, 1);
>> [iu_ft, iv_ft, iu_ip, iv_ip] = mpiv_filter(iu, iv, 2, 2.0, 3, 1);
```

The window size started from 64×64 pixels and ended at 8×8 pixels. Similar results can also be obtained if the MQD algorithm is used (using the ‘mqd’ option in mpiv).

3.4 Sub-pixel Analysis

A three-point curve fitting technique is used to calculate the ‘real’ peak of the correlated results in both the correlation and MQD algorithms with sub-pixel accuracy. With this process, the sub-pixel accuracy is achieved in comparison to the accuracy of 0.5 pixel without the fit. The Gaussian function is used for the fitting in both the correlation and MQD algorithms. The formula is as following:

$$P_{sub} = i - \frac{1}{2}\frac{\ln C_{i+1} - \ln C_{i-1}}{\ln C_{i+1} - 2\ln C_i + \ln C_{i-1}} \tag{3}$$

4 Accuracy

To test the accuracy of mpiv, the PIV standard images were used. The images were taken from VSJ-PIV (case 1 in PIV Standard Project) distributed by the Visualization Society of Japan (http://www.vsj.or.jp/).

For comparison, figure 3 shows the dependence of the r.m.s. error on different algorithms chosen in mpiv, and different window sizes used in the process. Basically, there is no significant difference between choosing ‘cor’ or ‘mqd’ in terms of accuracy. The correlation algorithm is, however, faster than the MQD method, while the MQD algorithm seems to be more robust than the correlation method.

5 Geometrical Correction for Taken Image

The geometrical correction for taken images are important for distorted images, although mpiv doesn't include the codes. However, the Image Processing Toolbox in MATLAB has nice commands to transform from the distorted images to orthogonal coordinate.

Step 1: Select control points For example, here is a orthogonal coordinate images as 'gird.bmp' and taken calibration images 'calibration.bmp'. Using cpselect, we can relationship between control points of pair images.

```
cpselect('grid.bmp','calibration.bmp')
```

The control points are landmarks that you can find in both images.

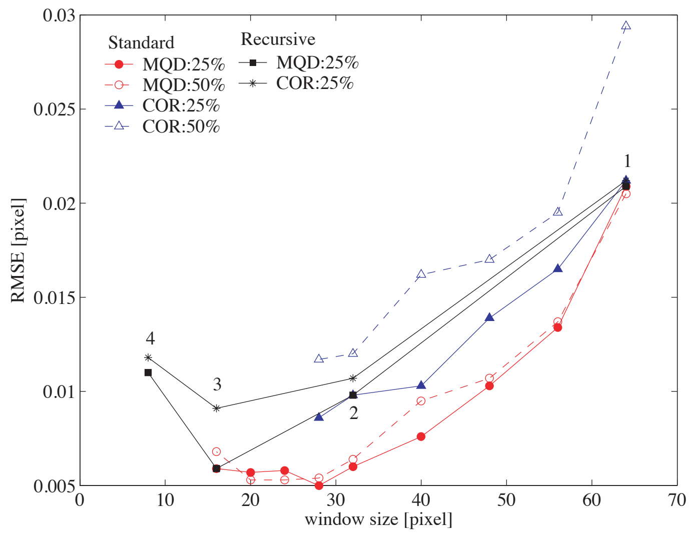


Figure 3: Dependence of r.m.s. error on different algorithms and window sizes

The number of control points depends on which transformation will you used. The 'projective' transformation requires 4 pairs, the linear, second and third order polynomial transformations require 2, 6, 10 pairs. After selection of pairs, please save the data.

Save control points by choosing the '*File menu*', then the '*Save Points*' to Workspace option. Save the points, overwriting variables *input_points* and *base_points*.

Step 2: Making geometric transformation matrix The second step is making geometric matrix as

```
t_concord = cp2tform(input_points,base_points,'projective');
```

cp2tform will find the parameters of the distortion that best fits the stray *input_points* and *base_points* you picked.

Step 3: Transform Image This is final step to get geometrical correction of distorted image. imtransform can do that. Note that the choice of '*XData*' and '*YData*' values ensures the registered image will be aligned with the orthophoto.

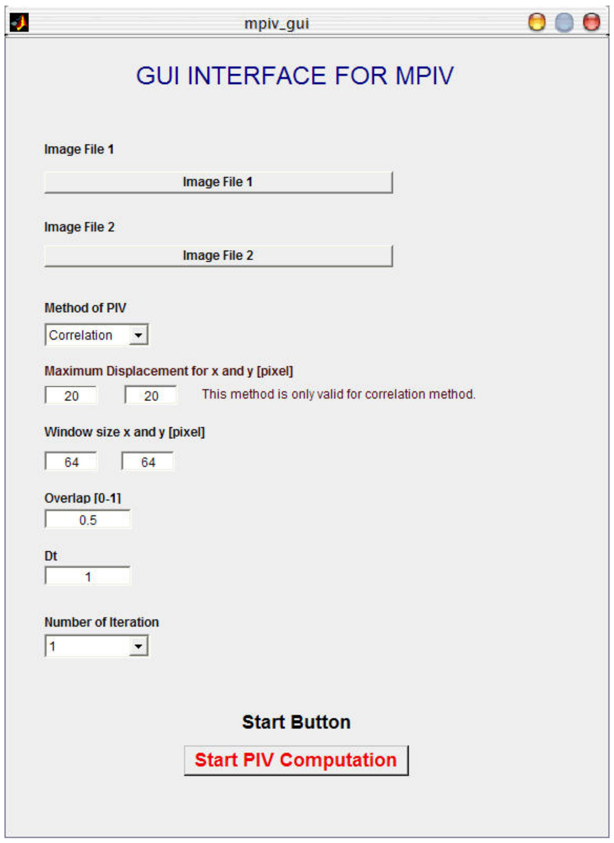
```
im1  = imread('image1.bmp');
info = imfinfo('image1.bmp');
registered = imtransform(im1,t_concord,...
                         'XData',[1 info.Width], 'YData',[1 info.Height]);
```

You can do Step 1-4 once. After that, *t_concord* can be used repeatably.

The original source of this procedure is written by help of Image Processing Toolbox in MATLAB. Please look and check in detail as you like.

6 Graphical User Interface

For beginners, a simple Graphical User Interface (GUI) menu has been added since version 0.96. The only one time velocity estimation procedure is available at the present version.



Method of PIV: Correlation

Maximum Displacement for x and y [pixel]: 20 20 This method is only valid for correlation method.

Window size x and y [pixel]: 64 64

Overlap [0-1]: 0.5

Dt: 1

Number of Iteration: 1

Figure 4: GUI Menu

GUI menu will be improved soon or later.

7 Enhancement of Computational Speed

As being well known, the computational speed for a code running under under MATLAB is slower than that under C or Fortran, generally. The easiest remedy here is to use 'MCC MATLAB to C/C++ Compiler'. Since most of CPU time is consumed when running the correlation or MQD functions (as well as the median filter and Kriging interpolation) in mpiv, the MATLAB-to-C compiler is suggested to be used only for these functions. A typical command is

```
>> mcc -x piv_crs
```

The C-complied .mex?? format files have higher preference than the original .m format files when running MATLAB, even if they are both placed in the same directory.

The following is a list of CPU time-consuming functions in mpiv.

```
func_findpeak2.m
piv_mqd.m
piv_mqr.m
piv_mrs.m
```

```
piv_cor.m
piv_crr.m
piv_crs.m
vector_filter_median.m
xcorr2.m
```

The following is an example for running the C-complied mpiv using the sample images:

```
>> mcc -x piv_mqd
>> im1 = imread('image1.bmp');
>> im2 = imread('image2.bmp');
>> tic; [xi, yi, iu, iv] = mpiv(im1, im2, 32, 32, 0.5, 0.5, 20, 20, 1,'mqd', 2, 1); toc
>> [iu_ft,iv_ft,iu_ip,iv_ip]=mpiv_filter(iu, iv, 2, 2.0, 3, 1);
```

In this example, the CPU time was reduced from 29.7 second originally to 19.1 second with the mex files piv_mqd.m (on an AMD Athlon 2000+). Using the MATLAB-to-C complier, mpiv saves 36% of time in comparison to the same MATLAB program without using the compiler.

We would like to point out that the two-dimensional correlation algorithm in the program uses the MATLAB function 'xcorr2' to compute the cross-correlation function. However, xcorr2.m in MATLAB does not use fast Fourier transform (FFT) therefore it is relatively slow. We replaced xcorr2.m to another faster routine for correlation computation applying FFT, CPU time is found to be greatly reduced (we cannot include this file in the mpiv toolbox due to copyright issue because it was directly modified from xcorr2.m).

8 Reference Manual of mpiv

This section provides the command reference for mpiv.m and mpiv_filter.m with some details.

8.1 mpiv.m

mpiv.m is the main routine of mpiv. mpiv.m is an external function (.m file) in MATLAB with its format as following:

```
function [xi,yi,iu,iv]=mpiv( im1, im2, nx_window, ny_window, ...
                             overlap_x, overlap_y, ...
                             iu_max, iv_max, dt, ...
                             piv_type, i_recur, i_plot).
```

The input augments of mpiv.m are:

```
im1 and im2    : input images (double precision)
{nx,ny}_window : processing window sizes in x and y,
                 -> 16, 32, or 64 are recommended (unit: pixel)
                 64 is good to start with in recursive PIV.
overlap_{x,y}  : fraction of overlapping windows in x and y
                 -> range [0.0 to 0.9] (no overlap to 90% overlap)
                 0.0 or 0.5 is typically used.
{iu,iv}_max    : maximum allowed velocities in x and y (in pixel)
                 If entering 0, they become 1/3 of the window size
                 -> these two parameter also provide the search area
                    if ‘mqd’ is chosen
                    (A large value will increased CPU time dramatically).
dt             : time separation between im1 and im2 (unit: second)
                 use ‘1’ to give output in pixel
piv_type       : = ‘cor’: cross-correlation algorithm
                 = ‘mqd’: MQD algorithm
i_recur        : number of recurrence to enhance vector resolution/accuracy
                 = 0 -> no recurrence
                 = 1 -> one recurrence with the same window size
                 = 2 -> reduce window size by one half
                 each increment (after 1) reduces the window size by one half
i_plot         : = 1 -> plot the calculated results
                 otherwise -> no plot
```

The output from mpiv.m is as follows:

```
xi, yi         : location of vectors in x and y (in pixel)
iu, iv         : output velocity vectors in x and y (in pixel if dt = 1.0)
```

Please mind the arrangement/orientation of the output arrays and matrices in the programs. It normally requires matrix transpose for plotting.

8.2 mpiv_filter.m

mpiv_filter.m is to post-process the output from mpiv. It includes two parts: spurious vectors elimination and missing vectors interpolation. mpiv_filter.m is an external function (.m file) in MATLAB with a format as following:

```
function [iu_f, iv_f, iu_i, iv_i] = mpiv_filter( iu, iv, ...
                                                 i_filter, vec_std,...
                                                 i_interp, i_plot )
```

The input parameters to mpiv_filter.m are:

```
iu, iv         : input vectors from the output of mpiv.m
i_filter       : = 1, standard filter
                 = 2, median filter
                 = 3, global filter
                 i_filter = 2 (or 1) is recommended
vec_std        : threshold value to eliminate error vectors.
                 typical value of vec_std is between 1.5 and 2.0
i_interp       : = 1, linear interpolation
                 = 2, cubic-spline interpolation
                 = 3, Kriging interpolation
                 = 0, no interpolation
                 i_interp = 3 is recommended
i_plot         : = 1;  plot result during piv process with pause for check
                 otherwise, no plot
```

The filter uses a small area (typically 3×3 to 9×9 vectors) to calculate the mean, median, and standard deviation, and the number of valid vectors (with values other than *NaN*) in the area. The size of the area is determined by a preset threshold value for the number of valid vectors. The size of the area increases when the number of valid vectors is below this value. It outputs *NaN* when reaching the preset maximum size while containing less than the threshold vector number. In calculating the statistical values (*e.g.*, mean, median, and standard deviation), the vectors with a value outside the range of twice the r.m.s. value are not included in the calculation to avoid the influence of these potential bad vectors with large deviations.

The standard and median filters use the calculated mean, median, and standard deviation to determine whether the vector of interest (called target vector hereafter) is spurious or not. If the target vector is within the range of vec_std × standard deviation from the mean or median value (for standard and median filter, respectively), it is then considered as a good vector. It is otherwise removed from the 'valid vectors' and replaced with *NaN*.

The interpolation is to assign a value for each *NaN* vectors. Similar to the filter, an expandable small area is used to interpolate each vector if Kriging method is chosen. Note that with more than one continuous *NaN* vectors neighboring to each other, only Kriging method gives interpolated results (the other two interpolation methods keep *NaN* in the output). Therefore the Kriging method is recommended in the filtering process for most cases. However, Kriging is computational intensive. It may take a significant portion of time in the PIV processing (if the recursive process is chosen) and post-processing.

The output of mpiv_filter.m contains two sets of velocity vectors:

```
iu_f, iv_f   : filtered velocity matrices in x and y without interpolation
iu_i, iv_i   : filtered velocity matrices in x and y with interpolation
```

Note that all the error vectors in iu_f and iv_f as well as in other inputs and outputs throughout mpiv are denoted as *NaN*.

8.2.1 vector_interp_linear.m

Interpolate missing vector using linear interpolation. It works only when two immediate adjacent velocity vectors to the target vector exist. No interpolation will be performed if otherwise.

8.2.2 vector_interp_spline.m

Interpolate missing vector using cubic-spline interpolation. Same as the linear interpolation function, it works only when the immediate adjacent velocity vectors to the target vector exist.

8.2.3 vector_interp_kriging_local.m

vector_interp_kriging.m is a Kriging interpolation function. It requires the DACE (Design and Analysis of Computer Experiments) toolbox. The toolbox was developed by Hans Bruun Nielsen, Soren Nymand Lophaven and Jacob Sondergaard at Technical University of Denmark. Download the toolbox from http://www.imm.dtu.dk/~hbn/dace/, expand it, and put it on the path of MATLAB.

8.3 mpiv_smooth.m

mpiv_smooth.m is a post-processing function of mpiv and is an external function (.m file) in MATLAB with the format as following:

```
function [ uo ]=mpiv_smooth( ui )
```

The input and output variables are:

```
ui             : Input vectors estimated from mpiv.m
uo             : Output smoothed vectors
```

This function is optional.

8.4 Other Functions

There are near by 20 functions written in MATLAB and used in mpiv. Users should refer to the documentation in each of the programs if one wants to know more details about mpiv.

9 Request from the Developers

If you find any bugs in the codes, please inform Nobuhito Mori (mori@criepi.denken.or.jp) or Kuang-An Chang (kchang@tamu.edu). Also we greatly appreciate any constructive comments and suggestions for the improvement of the codes, and additional contributions to the codes (proper credits will be granted).

Finally we would greatly appreciate a copy (in PDF or postscript format) of any works you published in which mpiv is used. Please cite the title of this document/user manual in your publications

10 License Agreement and Disclaimer

mpiv is a free software package. Using this package implies you have agreed the agreement stated here in.

You are allowed to redistribute this package and/or modify it under the terms of Berkeley Software Distribution License (BSD). However, mpiv is asserted with the following addendum concerning its usage: This software and any of its derivatives shall only be used for educational purposes or scientific research without the intention of any financial gains or commercial behaviors. The use of this software or its derivatives for any purpose that results in financial gains or commercial behaviors by a person(s) or organization(s) without written consents from the developers is a breach of this agreement.

This software is distributed for a goal that it will be a useful tool for engineers and scientists. However, it comes free but **WITHOUT ANY GUARANTY**. It does not guarantee any **MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE**. In addition the developers are not liable in any consequences, damages, and losses arising from the use of the software for any purposes. The GNU General Public License forms the main part of the license agreement for this software package.

11 Contributors to mpiv

1. Dr. D. F. Liang, who found typos in the document and provided comments to the correlation function (Oct. 4, 2002).
2. Dr. K.-A. Chang, who joined as a major developer since June 2003.

12 References

Adrian, R. J. (1991) ‘Particle imaging techniques for experimental fluid mechanics’, Annual Review of Fluid Mechanics, vol.23, pp. 261-304.

Raffel, M., C. Willert and J. Kompenhans (1998) ‘Particle Image Velocimetry’, Springer.

Willert, C. E. and Gharib, M. (1991) ‘Digital particle image velocimetry’, Experiments in Fluids, vol. 10, pp. 181-193.

13 Update History

0.965 2004/12/21 Geometrical correction has been inserted.

0.961 2004/12/03 new sub function nanmean2.m has been inserted.

0.96 2003/10/08 GUI menu, piv_gui.m, has been added. (0.01)

0.95 2003/ 7/02 piv_mqd.m(1.00) and piv_mqr.m(0.53) are revised. piv_mrs.m has been inserted. New definition of MMR have been added in func_findpeak2.m.

0.93 2003/ 6/26 piv_cor.m(0.73), piv_crs.m(0.49) vector_filter_median.m(0.60), vector_filter_vecstd.m(0.60), have been modified. func_histfilter.m has been inserted.

0.91 2003/ 6/23 piv_cor.m (0.66) and piv_crs.m (0.46) have been modified.

0.90 2003/ 6/20 piv_cor.m and piv_crs.m have been modified.

0.82 2003/06/16 piv_crs.m and vector_interp_kriging.m have been modified.

0.80 2003/06/12 Total package has been tuned and refined.. piv_crs.m has been added. Kriging interpolation has been added.

0.70 2003/06/11 Super-resolution PIV with correlation function, piv_crr.m has been added. Smoothing functions have been inserted by KAC.

0.65 2003/06/10 Filter functions have been modified by KAC.

0.60 2003/06/10 bug fixed in piv_cor.m.

0.52 2003/04/07 Peak search routines in piv_mqr.m and piv_mqd.m have been modified

0.51 2003/04/03 bug fixed in piv_mqr.m.

0.50 2003/04/03 Super-resolution PIV, piv_mqr.m has been modified. Three new functions have been inserted and the some functions have been modified.

0.32 2003/04/02 Super-resolution PIV, piv_mqr.m has been added.

0.31 2003/03/27 piv_mqd_c.m has been added.

0.30 2002/12/04 piv_mqd.m has been simplified and piv_cor.m has been added.

0.26 2002/10/21 some bugs fixed for piv_mqd.m

0.25 2002/10/09 bug fixed for piv_mqd.m

0.21 2002/09/30 mcc (MATLAB Compiler) has been inserted

0.20 2002/09/20 image input routine has been changed

0.15 2002/09/12 vector interpolation routine has been added

0.10 2002/09/12 check error vector routine has been added

0.01 2002/09/11 First version reduced from full program

14 mpiv is NOT

- Max-Planck Institute Gesellshaft (http://www.virtual-institute.de/)
- Mitchell Prehistoric Indian Village (http://www.mitchellindianvillage.org/)
- MP IV: Le Mega Pentenng (http://www.blada.com/chroniques/mpiv/pentenng1.htm)