UNIVERSITY OF KENTUCKY
College of Engineering

Particle Tracking Velocimetry with *OpenPTV*

REPORT BY DOMINIK BAUER

ADVISOR: PH.D. SEAN BAILEY

FEBRUARY 26TH, 2016

Table of Contents

Introduction

Particle tracking velocimetry (PTV) is a method used to track individual particles in fluids. In contrast to Particle Image Velocimetry (PIV), which represents the Eulerian approach to visualize flows, PTV is a Lagrangian approach which yields particle trajectories, velocities and velocity gradients. In general, PTV can either be utilized as a two-dimensional or three-dimensional method. Whereas a three-dimensional approach is affiliated with a high experimental effort, requiring multiple cameras and three dimensional volume illumination, two-dimensional PTV can be applied rather easily. Currently there are several open source software codes available which perform particle tracking velocimetry. The software used for this project is called '*OpenPTV*'. Although this software seems to represent one of the more mature PTV codes available, it requires training and routine to effectively produce trajectories with it. This report is supposed to guide through the entire process of generating trajectories from raw experimental imagery using '*OpenPTV*'. It does not cover the scientific idea behind PTV, since there already is plenty of literature available on this topic. Suggestions for literature and scientific papers can be found in the appendix.

This report outlines how to get two and three-dimensional trajectories, although it focuses on the three-dimensional PTV approach as proceedings are similar.

First of all, a brief overview on pre-processing the raw imagery is given. Then the actual processing of the data using '*OpenPTV*' is described. This includes a detailed description of calibrating the camera orientation, identifying individual particles and how to track particles along a set of images using a tracking algorithm. In the end, post-processing methods established and used through the course of this project are presented.

Preprocessing

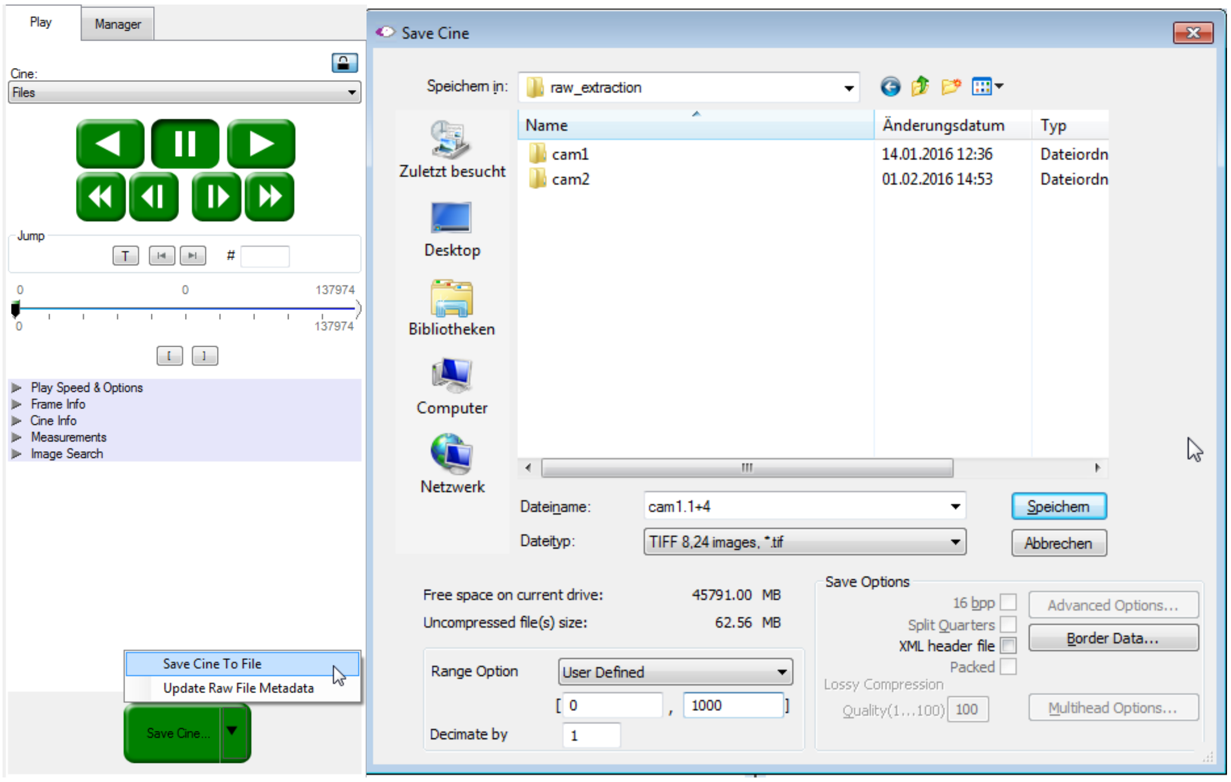
Several preprocessing steps are required before the actual particle tracking code can be applied. These steps include the extraction of individual images from raw high speed camera data as well as filtering disruptive background illumination and preparing the images for processing in '*OpenPTV*'.

Image extraction

In order to extract single images from the high speed imagery, the 'Phantom Cine Viewer' by vision research needs to be downloaded and installed. This software provides a multi-camera interface, filter options such as low or high pass filters and converts the .cine files into the desired file format. In order to further process the files using '*OpenPTV*' the images need to be converted into an 8-bit TIFF data format.

Following steps are recommended:

1. Download the 'Phantom CineViewer' at http://www.highspeedcameras.com/Service-Support/Downloads/Details/ID/539 and install it.
2. Open the *.cine files
3. Under the 'play' tab click 'Save cine to file' as shown in figure 1



Speichern in: raw_extraction

Name	Änderungsdatum	Typ
cam1	14.01.2016 12:36	Dateiordn
cam2	01.02.2016 14:53	Dateiordn

Dateiname: cam1.1+4
Dateityp: TIFF 8,24 images, *.tif

Free space on current drive: 45791.00 MB
Uncompressed file(s) size: 62.56 MB

Range Option: User Defined
[0 , 1000]
Decimate by 1

Quality(1...100) 100

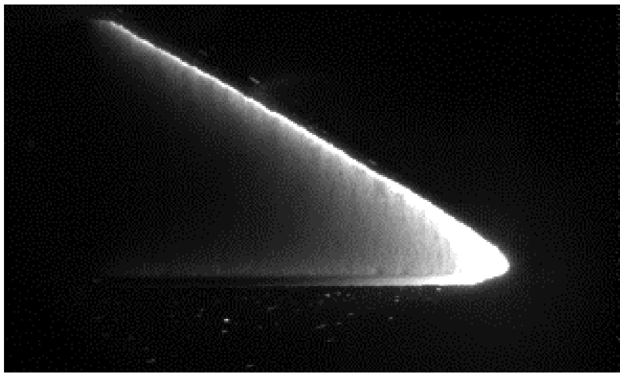
Figure 1 - User Menu in CineViewer and Save Cine prompt

4. To batch process a number of images a naming convention is required. Type the desired file name and add the '+' sign followed by a number of digits (e.g. 'cam1.1+4' → cam.1.10001, cam.1.10002, cam1.1000x.....)
5. Select 'TIFF 8,24images.tif' as output format

6. Under ‘Range Option’ specify the images to save (this needs to be in accordance with the digit specified in Step 4).

Filter Images

The experiments investigated in the course of this project as well as all further experiments related to ablative thermal protection systems are characterized by extreme flow conditions. These conditions lead to an intense glowing of stationary parts which hinders the particle tracking algorithm to track particles correctly. To improve the tracking results, all glowing parts need to be subtracted from the images. The MATLAB function ‘Background_Subtraction.m’ in the ‘/Pre_Processing’ folder is dedicated to subtract all disruptive elements from the raw imagery. This is done by averaging all images and subsequently subtracting the base mask from each image, leaving moving particles only.



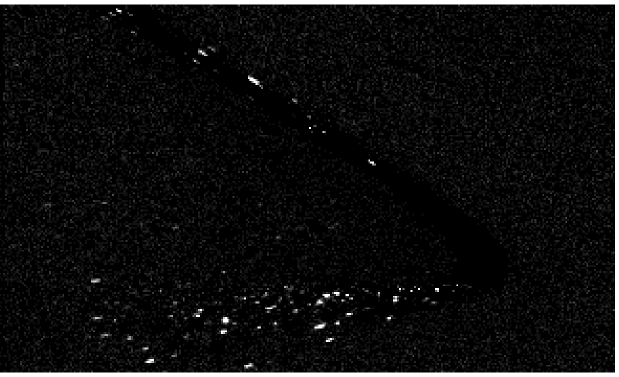


Figure 2 - Image after pre-processing

Figure 3 - Raw unprocessed image

Flip Images

This pre-processing step is especially relevant for three-dimensional PTV experiments. Using multiple cameras, some experimental setups require the use of mirrors to successfully capture images from different orientations. In case the data set includes images which were taken with the help of a mirror, these images need to be flipped accordingly prior to further processing.

It is recommended to use the matlab function ‘flip_img.m’ to flip larger numbers of images. The use of image editors should be avoided since some editors automatically add RGB channels when saving the flipped image. This will cause problems later as ‘*OpenPTV*’ is not be able to read images with RGB channels.

The ‘flip_img.m’ function uses built-in MATLAB functions and needs an input folder, an image base name (e.g. ‘cam2.’), an output folder and the first and last image numbers of the sequence. First, each pixel grayscale value is read into an array of the same dimensions as the image resolution. Then the array is flipped along the second dimension (vertical axis). The resulting array is stored again as TIFF file into the specified output folder. The for-loop runs through this procedure for each image specified in the input interval.

Figure 4 - Flipped pre-processed Image

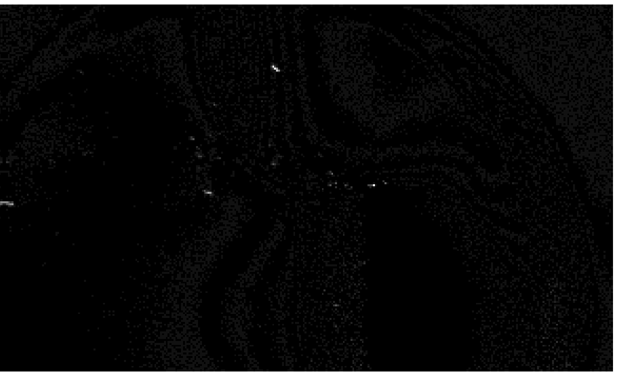
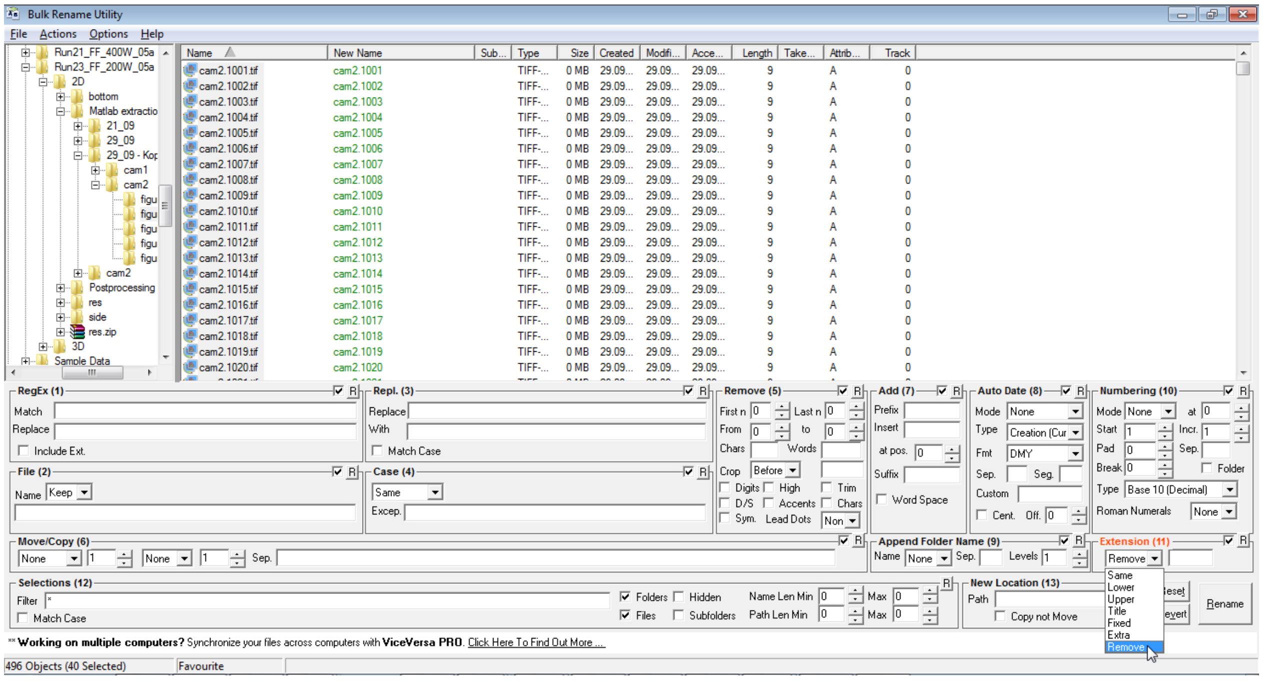


Figure 5 - Pre-processed Image

Remove file name extension

Due to unknown reasons '*OpenPTV*' cannot process images which have a file name extension such as '.tif'. Unfortunately MATLAB needs a file name extension in order to properly safe images in the right format. Therefore it is necessary to remove the extension '.tif' from each image to be processed with *OpenPTV* before continuing (calibration pictures are exempt!). It is not recommended to do this manually. Through the course of this project an open source tool named 'Bulk Rename Utility' (http://www.bulkrenameutility.co.uk/Download.php) is used. This tool renames even very large datasets quickly.



Name	New Name	Sub...	Type	Size	Created	Modifi...	Acce...	Length	Take...	Attrib...	Track
cam2.1001.tif	cam2.1001		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1002.tif	cam2.1002		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1003.tif	cam2.1003		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1004.tif	cam2.1004		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1005.tif	cam2.1005		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1006.tif	cam2.1006		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1007.tif	cam2.1007		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1008.tif	cam2.1008		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1009.tif	cam2.1009		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1010.tif	cam2.1010		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1011.tif	cam2.1011		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1012.tif	cam2.1012		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1013.tif	cam2.1013		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1014.tif	cam2.1014		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1015.tif	cam2.1015		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1016.tif	cam2.1016		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1017.tif	cam2.1017		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1018.tif	cam2.1018		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1019.tif	cam2.1019		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0
cam2.1020.tif	cam2.1020		TIFF-...	0 MB	29.09...	29.09...	29.09...	9		A	0

RegEx (1): Match, Replace, Include Ext. ☐
Repl. (3): Replace, With, Match Case ☐
Remove (5): First n 0, Last n 0, From 0 to 0, Chars, Words, Crop Before, Digits ☐, High ☐, Trim ☐, D/S ☐, Accents ☐, Chars ☐, Sym. ☐, Lead Dots Non
Add (7): Prefix, Insert, at pos. 0, Suffix, Word Space ☐
Auto Date (8): Mode None, Type Creation (Cur, Fmt DMY, Sep., Seg., Custom, Cent. ☐, Off. 0
Numbering (10): Mode None, at 0, Start 1, Incr. 1, Pad 0, Sep., Break 0, Folder ☐, Type Base 10 (Decimal), Roman Numerals None
File (2): Name Keep
Case (4): Same, Excep.
Move/Copy (6): None, 1, None, 1, Sep.
Append Folder Name (9): Name None, Sep., Levels 1
Extension (11): Remove (Same, Lower, Upper, Title, Fixed, Extra, [✓ Remove])
Selections (12): Filter *, Match Case ☐, Folders ☑, Hidden ☐, Files ☑, Subfolders ☐, Name Len Min 0, Max 0, Path Len Min 0, Max 0
New Location (13): Path, Copy not Move ☐

** **Working on multiple computers?** Synchronize your files across computers with **ViceVersa PRO**. Click Here To Find Out More ...

496 Objects (40 Selected) Favourite

Figure 6 - GUI of Bulk Rename Utility which was used to remove the '.tif' file extensions

OpenPTV

Software Installation

Although it is possible to install '*OpenPTV*' on Windows, it is highly recommended to use either a Linux or Mac operating system to run the software. This is due to the fact that Windows does not contain a default C build environment as well as it does not implement the UNIX standards base. In order to install '*OpenPTV*' follow the installation instructions found on the official website (http://*OpenPTV*-python.readthedocs.org/en/latest/installation_instruction.html).

Personal remark by the author: Don't get the impression that installing '*OpenPTV*' will be easy, only because the manual is short and everything looks straight forward. If it doesn't work the first time, don't give up! Eventually- after some days or weeks- it will.

Running the software

In order to run the software open a terminal and navigate to the ~/*OpenPTV*/liboptv/*OpenPTV*-python/pyptv_gui directory using the 'cd' command. Entering 'python pyptv_gui.py' should start the software. If this produces the following error:

```
> Wrong experimental directory /home/bluecat/ptv_test_folder
> src_path= /home/bluecat/OpenPTV-python/src_c
> Traceback (most recent call last):
> File "pyptv_gui.py", line 60, in <module>
> import ptv1 as ptv
> ImportError: liboptv.so: cannot open shared object file: No such file or
> directory
```

it is recommended to try either

```
PATH=/usr/local/lib:$PATH python pyptv_gui.py
```

or

```
LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH python pyptv_gui.py
```

For the course of this project usually the second command worked fine.

Note:
It is possible to fix this problem by updating the local ~/.bashrc file. This can be done by opening a new terminal and running the command '$ vi .bashrc. After pressing 'i', the following line should be entered at the very end of the file:

```
export  LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH python
pyptv_gui.py
```

To save the file, press 'Esc', type :wq and press enter.

In Ubuntu this file can be found in the user directory using the explorer. Since the bashrc file is a hidden file, it is necessary to press ‘Ctrl+h’ to make it visible in Ubuntu. Open the file and change the ‘export’ path to whichever path worked.

Creating a new Project

Upon start ‘*OpenPTV*’ will request an experimental directory. This experimental directory represents one individual project and must contain following file structure:

- img - contains preprocessed images
- parameter – several text files which will be edited within the software
- calibration - contains calibration images, camera orientations etc.
- res – storage folder for output data

Note: If the experimental directory does not contain the above-mentioned folder structure, ‘*OpenPTV*’ will not start.

A default experimental project is provided in addition to this report. Parameters and calibration files are specific for each individual project and therefore need to be changed accordingly each time. Nevertheless, for similar experiments the default folder can serve as a good first guess.

Calibration

Calibration Introduction

Calibrating the camera system is a crucial step in the process of producing trajectories. Before the actual image sequence is ready for processing it is necessary to determine intrinsic and extrinsic camera parameters. These parameter define how the three dimensional position in global coordinates $(x_g, y_g, z_g, 1)$ is linked to the two dimensional position in pixel coordinates [u,v,1] according to following relation:

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\,[R\ \ T] \begin{bmatrix} x_g \\ y_g \\ z_g \\ 1 \end{bmatrix} \qquad K = \begin{bmatrix} \alpha_x & \gamma & x_p \\ 0 & \alpha_y & y_p \\ 0 & 0 & 1 \end{bmatrix}.$$

Matrix K contains five intrinsic parameters α_x , α_y , γ , x_p and y_p . The parameters $\alpha_x = f * m_x$ and $\alpha_y = f * m_y$ represent the focal length in terms of pixels, with f as the focal length and $m_{x,y}$ as scale factors relating pixels to distance. Gamma is the so called skew factor, which usually is zero. The remaining two parameters x_p and y_p represent the principal point, which ideally coincide with the image center.

Extrinsic parameters R, T specify the coordinate system transformations from 3D global coordinates to 2D camera coordinates. The 3x3 rotational matrix R and the translation vector T define the cameras position coordinate system. Figure 7 depicts the different coordinate systems.

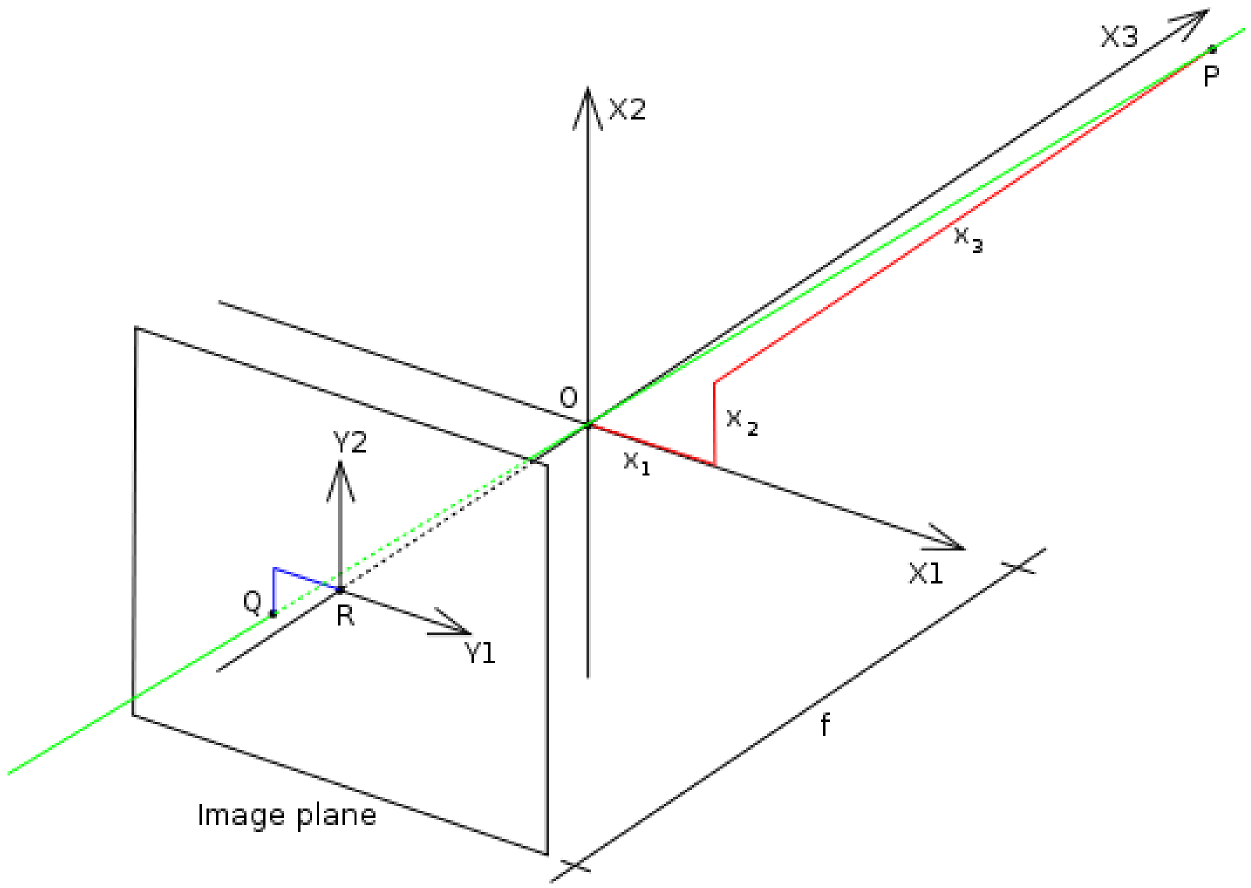


Figure 7 - Schematic depiction of a pinhole camera model

Since physical elements in camera lenses are not perfectly aligned, images usually show some kind of distortion effects. In order to comply with the physical reality Brown's distortion model is introduced in '*OpenPTV*'. As additional input to correct for radial and tangential distortion parameters k1, k2, k3, p1, p2, scale in x-direction and shear in rad are required.

This is a very short and sparse explanation, therefore it is recommended to review literature about camera calibration to better understand the meaning of the different parameters.

'*OpenPTV*' needs most of the above mentioned parameters as input, to create a calibration. Therefore a specific file structure and convention exists which is explained in the following section.

For each camera the 'calibration' folder contains a calibration image 'camX.tif', an orientation file 'camX.tif.ori', a lens distortion parameter file 'camX.tif.addpar' and a 'calblock.txt' file that contains the x,y,z coordinates (in mm) of the calibration target. While the *.ori files will be edited in '*OpenPTV*' itself, *.addpar and 'calblock.txt' files have to be updated manually.

Figure 'cameraorientation' depicts the format in which the camera orientation is stored. The first row contains the distance of the camera to the projective center, respectively the position of the global coordinate systems origin relative to the camera-centered coordinate system. The second row contains the angular displacement (in radian) of the camera relative to the global coordinate system. This coordinate system is defined as a right hand system, with x in flow direction and z coinciding with the imaging axis of the camera (in other words: z-axis is the axis connecting the camera sensor with the target).

The following three rows represent the 3x3 rotation matrix R, which is calculated with respect to the first to rows and thus does not need to be updated.

Sometimes sensors are not exactly parallel, meaning that the position of the pinhole in respect to the image center is shifted. Then the parameters x_p and y_p in row six need to be adjusted (in mm). If the camera imaging axis is orthogonal to the sensor x_p and y_p are zero.

Row seven stores the focal length or focal distance.

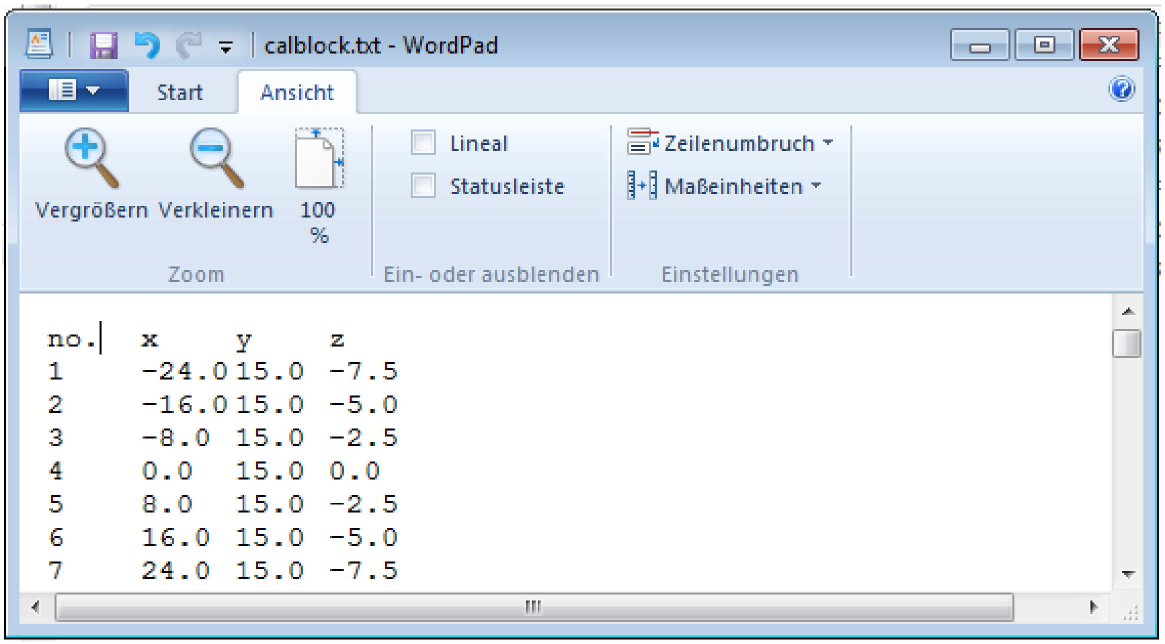
The parameters in the last row are relevant if the imaging target is located behind a glass surface or in a water tank for example. If this is the case, the last three parameters represent the position of the glass in respect to the origin of the target coordinate system.

Additional parameters for Browns distortion model are stored in the 'camX.tif.addpar' file.

Calibration Procedure

Following procedure is recommended for calibration:

1. Use the sample calibration folder provided and copy its content into the experimental folder.
2. Open the 'calblock.txt' file. This file stores the coordinates of the points on the calibration target. It contains four columns; the calibration point number and X, Y, Z coordinates in mm. Note: the origin of the coordinate system used to reference the calibration points should be in the center of the calibration target. Later, this will be the origin of the coordinate system in which the trajectories are stored. **Important!:** Sequence the calibration points from the **upper left corner to the bottom-right corner** of the calibration target as depicted in Figure 9 - Calibration target depicting the numbering convention for the target dots.



```
no.  x     y    z
1    -24.0 15.0 -7.5
2    -16.0 15.0 -5.0
3    -8.0  15.0 -2.5
4    0.0   15.0 0.0
5    8.0   15.0 -2.5
6    16.0  15.0 -5.0
7    24.0  15.0 -7.5
```

Figure 8 - Example of calblock.txt file opened in word pad

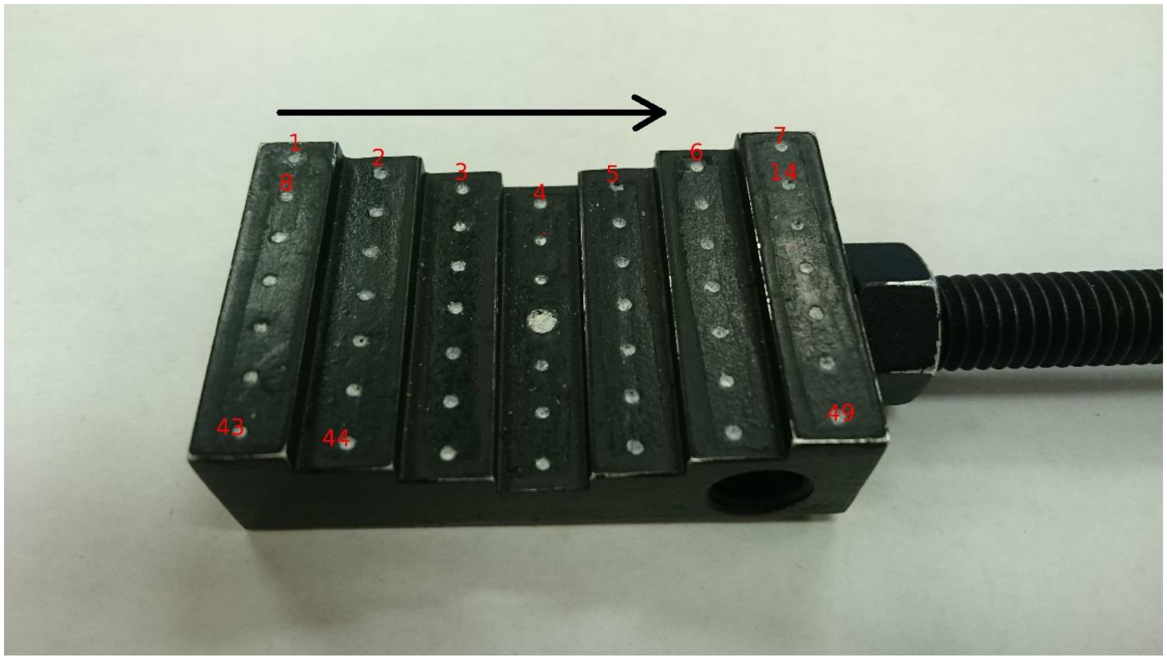
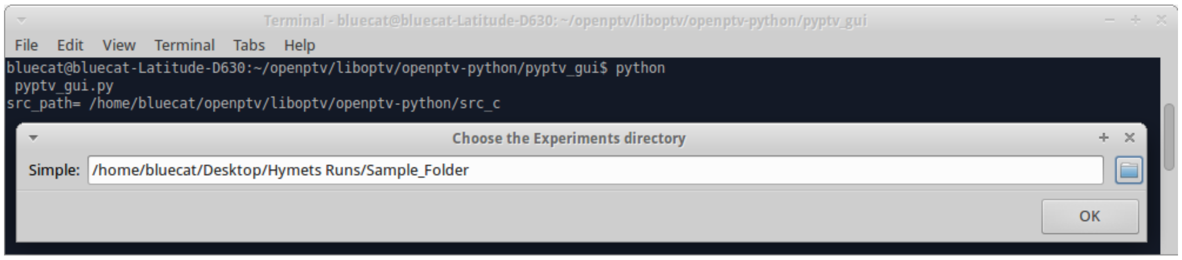


Figure 9 - Calibration target depicting the numbering convention for the target dots

3. Run '*OpenPTV*' and select experimental directory


Terminal - bluecat@bluecat-Latitude-D630: ~/openptv/liboptv/openptv-python/pyptv_gui
File Edit View Terminal Tabs Help
Choose the Experiments directory
OK

```
bluecat@bluecat-Latitude-D630:~/openptv/liboptv/openptv-python/pyptv_gui$ python
 pyptv_gui.py
src_path= /home/bluecat/openptv/liboptv/openptv-python/src_c
```

Simple: /home/bluecat/Desktop/Hymets Runs/Sample_Folder

4. Click 'Start' →'Init/Restart'
5. Right-click on 'Run1' in the sidebar menu to open the context menu

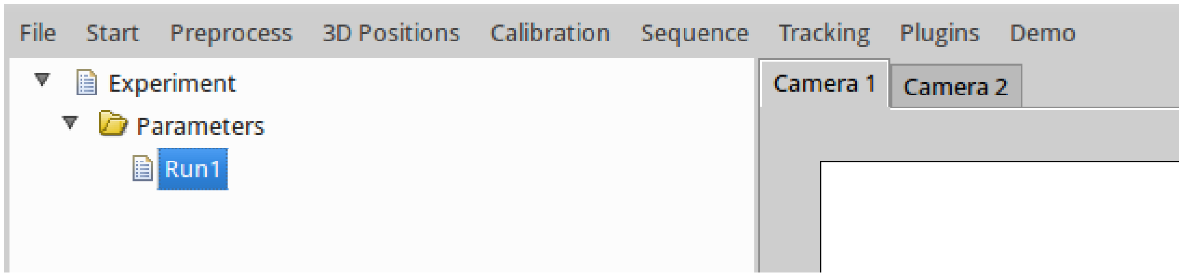


Figure 10 - Menu Bar and Experimental Directory in OpenPTV

6. Select 'Main parameters' in the context menu

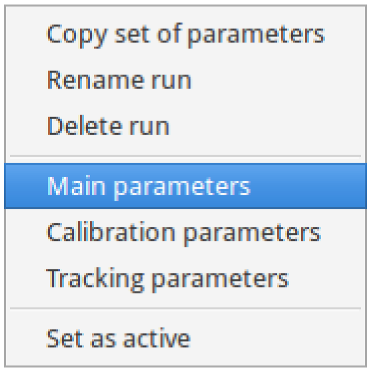
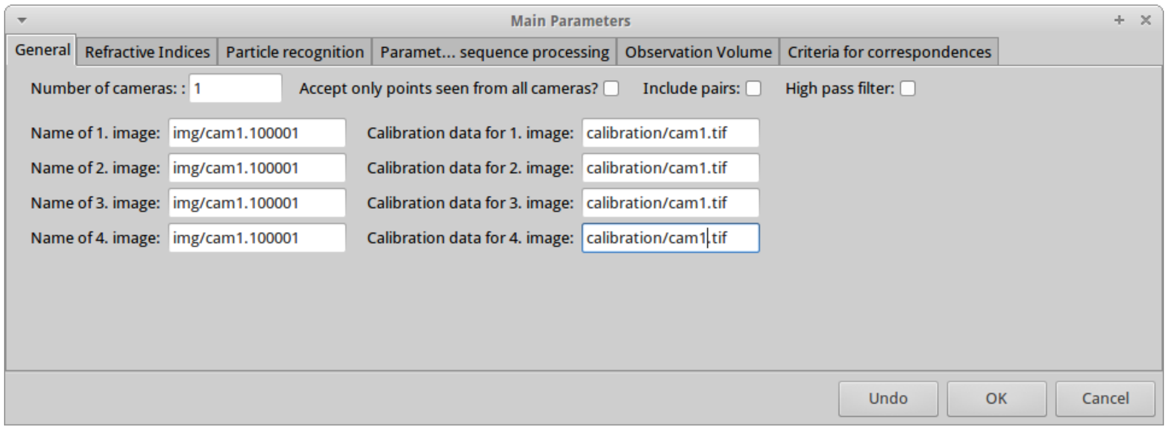


Figure 11 - Context Menu which opens after right-clicking on an experimental run

7. Under the 'General' tab check and if necessary change the 'Number of cameras' used in the experiment. Note: For 2D cases use the camera directory and copy it into each row as depicted in figure'main_parameters2D'.



Number of cameras: : 1
Accept only points seen from all cameras? ☐
Include pairs: ☐
High pass filter: ☐

Name of 1. image:	img/cam1.100001	Calibration data for 1. image:	calibration/cam1.tif
Name of 2. image:	img/cam1.100001	Calibration data for 2. image:	calibration/cam1.tif
Name of 3. image:	img/cam1.100001	Calibration data for 3. image:	calibration/cam1.tif
Name of 4. image:	img/cam1.100001	Calibration data for 4. image:	calibration/cam1.tif

Figure 12 - Main Parameters, General Parameters

8. Click 'OK', then close '*OpenPTV*' and restart the entire program again. This is necessary for the changes to be effective.
9. After restart click 'Init/Restart', click on 'Create calibration' from the dropdown menu 'Calibration'.
10. Click 'Load/Show Images. If folders and image paths are correct it will show the calibration images, otherwise the windows will stay blank. Check whether the right images are displayed!

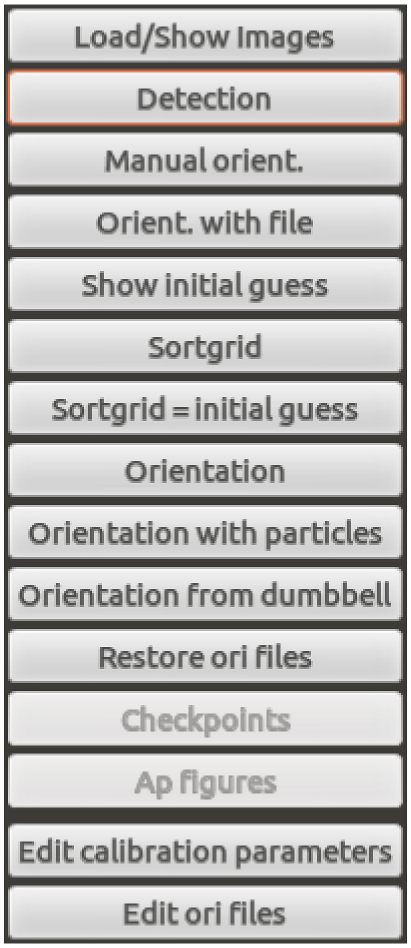


Figure 13 - Calibration Menu

11. Click ‘Detection’. Based on the calibration parameters set, the program will track the calibration target dots and mark them with blue crosses. Most likely not every point will be recognized at first try. Therefore it is necessary to click ‘Edit calibration parameters’ and edit the parameters ‘Threshold’, particle detection and image and pixel size. Threshold specifies the minimum grey value for pixels to be recognized as particles. Particle detection parameters define the size of the calibration target dots in x-, y-direction and in terms of total amount of pixels. The image size simply refers to the image resolution. Pixel size can be calculated looking at the calibration target in the calibration image. For example, if the dots are 5 mm apart from each other in reality and 100 pixels apart in the image, the pixel size would be 0.05mm. After adjusting the parameters, click ‘Load/Show Images’ and ‘Detection’ again and check if all dots are recognized. Sometimes several adjustments are required.

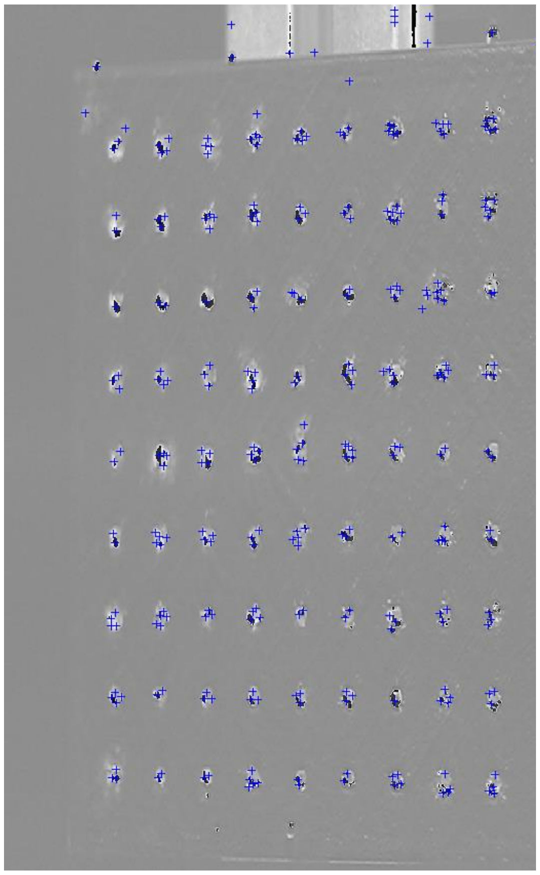


Figure 14- Example of successful detection of the calibration target points on the calibration target

12. Now, four target dots (usually the corner dots of the calibration target) need to be manually selected by 'left-clicking' their centroids. It is very important to start in the upper left corner, continue with upper right, lower left and finish in the lower right corner. In addition to manually selecting the corner dots, it is also necessary to specify the respective number of each of the four dots from the 'calblock.txt' file in the calibration parameters. In detail: If the calibration target has a 7x7 pattern the corner points will be dots number 1, 7, 42 and 49. In order to precisely click the centroid it is possible to zoom into the image using upward scroll. Pressing 'esc' will reset the view. After the four points are selected click 'Manual Orient.'. This will save the coordinates into a 'man_ori.dat' file.
13. Clicking on 'Orient. with file' will show the previously specified target points. If it is clicked right after 'Manual Orient.' nothing will happen because the targets were just stored right before. This option is for verification purposes only.
14. 'Show initial guess' will display the projection of all target points from the 'calblock.txt' file as yellow dots, depending on the camera orientation specified in the 'camX.ori' file.
    Ideally the yellow crosses are in accordance with the blue crosses. Since this virtually never occurs the initial attempt, one has to update the 'camX.ori' files. This is done clicking on 'Edit ori files'. Do not change the 3x3 rotation matrix. It is recommended to primarily change the first two lines. After modifying the orientation files, click on 'Show initial guess' again and see if the alignment improved. Repeat these steps until the initial guess is somehow aligned with the blue crosses. If nothing works, review the chapter on calibration basics or have a look into the literature recommendations. Unfortunately this probably is the most challenging task working with '*OpenPTV*'.
15. Clicking on 'Sortgrid' will assign the point labels from the 'calblock.txt' file to the projected target points.
16. 'Orientation' concludes the calibration process. This will perform the iterative calibration of the camera system. If this step was successful 'Orientation finished' will be displayed at the bottom of the calibration window. In the terminal you can read the iteratively calculated camera orientations. If several cameras are used, it is necessary to scroll up to read orientations of all cameras.
17. Copy the calculated orientations from the orientation procedure into the 'camX.ori' files and repeat the calibration procedure. Check if the initial guess is improved.

18. Try to use distortion parameters in order to further improve the calibration (This was not yet done in the course of this project).

Besides the single plane calibration described above, also a multi-plane calibration is possible with '*OpenPTV*'. This has not been tested yet.

Parameter Set up

Particle Recognition

Setting up parameters for particle recognition represents the first step in the actual processing of the experimental imagery. The goal is to recognize as many 'real' particles as possible without accidentally tracking noise. If the preprocessing was successful, the noise should already be reduced to a minimum. In total, '*OpenPTV*' provides eight different parameters for particle recognition which can be adjusted under 'Main Parameters' → 'Particle Recognition'. These parameters generally define what is considered a particle in terms of size, dimension and grey value. After the parameters have been adjusted the result can be checked running 'Preprocess' →'High pass filter' → 'Image coord'. It is recommended to find the best set of parameters by trial-and-error.

Particle Correspondences

As described earlier, PTV represents the Lagrangian approach to investigate flows. Therefore particles need to be matched among all views. Since particles usually are similar in terms of size, shape and illumination, the only way to find correspondent particles in different images, is to use the camera orientation parameters from the calibration. The method '*OpenPTV*' uses to find correspondences is based on epipolar geometry.

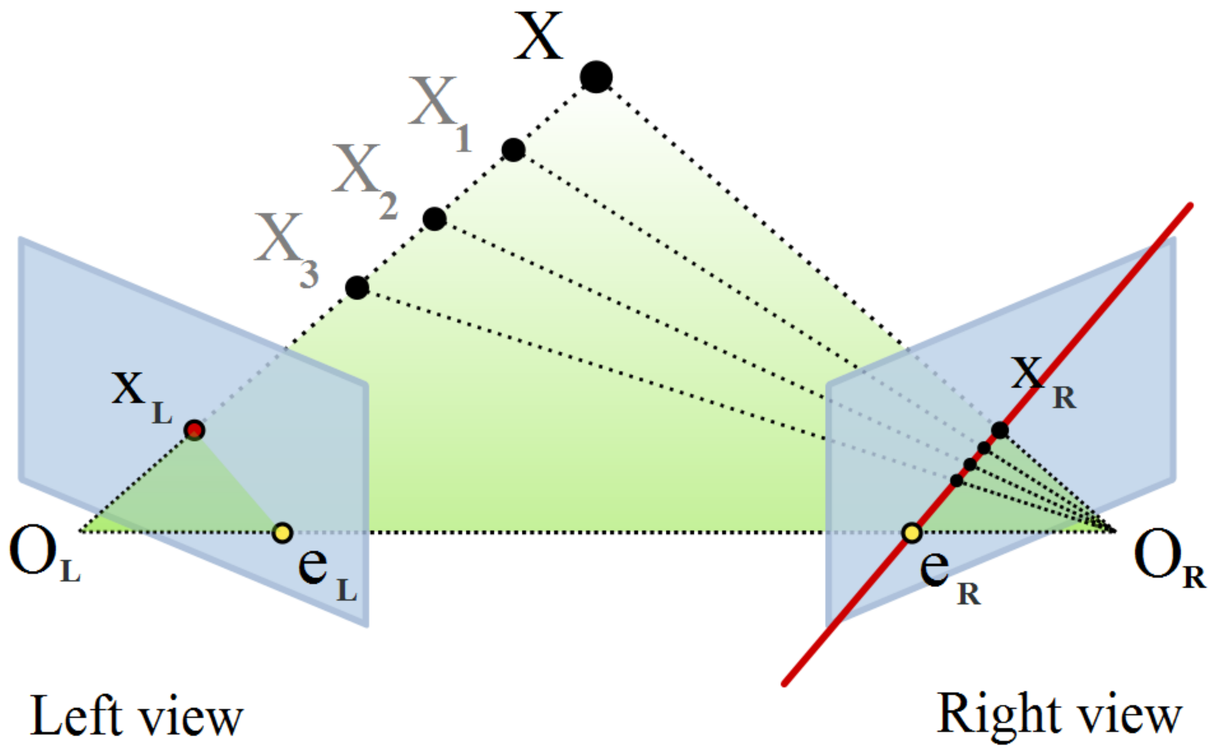


Figure 15 - Schematic depiction of a stereoscopic camera setup

Figure 15 depicts an experimental setup with two cameras. Points 'O_L' and 'O_R' represent the centers of each camera lens, while 'X' represents an arbitrary particle. 'x_L' and 'x_R' are the projections of 'X' in the respective image plane. Each camera center is also visible in each other's image plane. These so called epipolar points are depicted in **Error! Reference source not found.** by points 'e_L' and 'e_R'. The line 'O_L' – 'X' is seen as a point in the left camera, whereas the same line is seen as an epipolar line 'e_R' – 'x_R' in the right cameras image plane. The same applies to line 'O_R' – 'X' which is seen as a point in the right camera and as an epipolar line in the left cameras image plane. Thus, an epipolar line describes nothing else than a function of point 'X' in 3D- space. Since the line 'O_L' – 'X' passes through the optical center of lens 'O_L', the respective epipolar line in the right image must pass through the epipolar point regardless of where point 'X' is located.

As in '*OpenPTV*' the relative position of the camera system is known from the calibration, two important conclusions can be made based on epipolar geometry. No matter what point is observed in one image, the same point has to be observed on a known epipolar line in the other image. Furthermore, if points 'x_L' and 'x_R' are known, their projection lines 'O_L' – 'X' and 'O_R' – 'X' are also known. Consequently, point 'X' is located exactly at the intersection of the two projection lines and thus can be calculated from the image coordinates of the two images. This very well illustrates why a good calibration is crucial for three-dimensional particle tracking.

In '*OpenPTV*' six different parameters can be modified to tune the epipolar matching algorithm. An important parameter is the 'tolerance to epipolar line' (mm). The exact impact of these parameters is unknown though, due to insufficient documentation. Therefore in the course of this project particle correspondences and the respective parameters were found simply by trial-and-error.

Particle Tracking

As the name Particle Tracking Velocimetry indicates, PTV is all about tracking. This can be especially difficult with a large amount of particles present. According to (Maas 1993) the particle spacing displacement ratio

$$p = \frac{\Delta_0}{\mathrm{u}'\Delta t}$$

is a fundamental indicator on how difficult it is to track particles with PTV. The ratio is defined as the average particle spacing Δ_0 to the mean distance u' moved by particles within one imaging time step Δt. For p >>1 tracking is relatively easy, while for p <<1 tracking becomes almost impossible. In general terms this means that the farther particles are apart from each other and the slower they move, the better it is to track them.

'*OpenPTV*' uses a spatio-temporal tracking algorithm which is described in (Maas 1993) and (Willneff 2003). As input, this algorithm needs eight parameters. These include the minimum and maximum displacement in each coordinate direction, the angle a particle in direction along its trajectory and a magnitude acceleration. Since these parameters constitute the general boundary conditions of the particle velocities, it is important to have an approximate idea what velocities to expect.

It is not always easy to specify the minimum and maximum displacement. Especially for certain flow geometries this can be though. Figure 16 depicts a case in which the particle displacement in y- direction is dependent of the particle location. If a particle is traveling on the top of the wedge, a positive displacement in y-direction can be observed. However, using a large positive displacement as maximum displacement in y-direction will cause problems, because the particles at the bottom of the wedge are relatively close together. Thus, the algorithm will falsely track particles located at the bottom of the wedge moving in an upward/positive-y-direction. If this is the case, it is necessary to delete such misinterpreted trajectories during post-processing (see below).

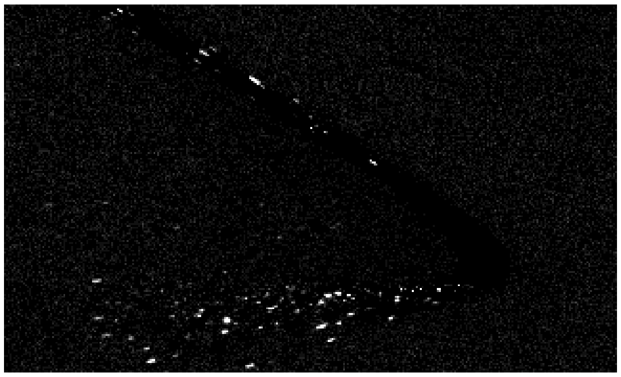


Figure 16- preprocessed image of hypersonic flow field around a wedge shaped sample

Processing a Run

Based on the previous chapters, the general parameters and parameters for particle recognition should be set up under the 'Main Parameters' or 'Tracking Parameters' context menu.

Assuming all parameters are set previously, this procedure describes how to process an entire run with '*OpenPTV*'.
Please note, that it is basically impossible to set up all parameters initially and produce satisfying trajectories. It is rather necessary to iteratively run through the described procedure, changing one parameter at a time and checking its effects on the results.

1. Click 'Start'.
2. Then open the experimental images and parameters by clicking 'Init/Restart'
3. Under the tab 'Preprocess' click
   - (optional) 'High pass filter' to apply a high pass filter (if images are preprocessed this might be useless or even disadvantageous)
   - In order to detect the particles with the specified parameters click 'Image Coord'. Detected particles will be marked with blue crosses.
   - Clicking on 'Correspondences' will run the epipolar matching algorithm. Depending on the amount of cameras used, the matched particles will be marked with differently colored crosses. In case four cameras are used, particles matched in all views are marked red,green if matched in three views. For a setup with two cameras, matched particles are marked green.

     This step will save the centroid coordinates of the matched particles into text files located in the 'img' folder.
4. To check the error between the particle locations and the corresponding epipolar lines, click '3D Positions'. This will display the RMS in the terminal.

These first four steps are especially useful to iteratively optimize parameters against a single time step. After the parameters produce satisfying results in terms of particle detection and correspondences, the algorithms can be applied to the entire dataset. Therefore adjust the 'Parameters sequence processing' tab in the Main Parameters menu. Make sure that the images file name extension such as '.tif' is deleted. Adhere to the convention that the first image has to start with cam*.**1**.....**1.** Thus for 1000 images the sequence has to start with 10001 and end with 11000.

5. By clicking ‘ Sequence’ →’Sequence without display’, ‘*OpenPTV*’ will detect particles and correspondences in the specified image sequence and store the respective *_target files into the ‘img’ folder. Additionally this step will also store rt_is.* files in the ‘res’ folder for epipolar matching.
6. In order to visualize the detected particles from the previous step, it is possible to display them in the first frame by clicking ‘Tracking’→ ‘Detected Particles’
7. Finally, run the tracking algorithm by clicking ‘Tracking’ → ‘Tracking without display’. It is recommended to choose this option especially for large datasets because it requires significantly less computational time than ‘Tracking with display’. To validate the used tracking parameters the ‘Tracking with display’ option proved to be helpful though. In this case make sure to use a small sequence (20-50 images) because you cannot cancel the tracking other than closing the entire program pressing 'ctrl+c’ in the terminal. The output ‘ptv_is.*’ files are being stored in the ‘res’ folder. Be careful! ‘*OpenPTV*’ will always store results in this folder, which means that results from previous parameter sets are overwritten. Always copy the files to another folder before you try out a new set of parameters!
8. For validation purposes it is possible to display the trajectories in the first image by clicking ‘Tracking’ →’Show Trajectories’. This option is a good way to quickly judge the quality of the results.

Post-processing

In order to analyze and visualize the trajectories obtained with ‘*OpenPTV*, several post processing programs and routines are available. Although post processing methods for python exist, for the course of this project MATLAB proved being more straight forward and easy to operate.

While there is preexisting MATLAB code available on the *OpenPTV* website, additional code is provided with this report. The entire post processing MATLAB code used and created in the course of this project, can be found in the Postprocessing folder. Before being able to use the functions, the folder needs to be added to the local MATLAB workspace. This can either be done by copying the folder to the workspace or by adding the folder to the MATLAB path using ‘Set Path’.

The first postprocessing step is to create a MATLAB struct which contains the x, y, z coordinates (in mm), the u, v, w velocities (in mm/s) and the ax, ay, az accelerations for each trajectory found. This can be done using the ‘ptv_is_to_traj.m’ function, which reads the rt_is.* and ptv_is.* files from ‘*OpenPTV*’ and saves each trajectory into a struct. To start the function following command should be typed into the MATLAB console:

```
traj = ptv_is_to_traj(directory, first, last, minlength, dt).
```

With ‘directory’ being a string referencing the folder containing the rt_is.* and ptv_is.* files. The variables first and last specify the first and last output file to be read in. The ‘*OpenPTV*’ tracking algorithm usually loses particles after a certain amount of images. Reasons for this can either be that the particles traveled out of sight or that the particle density is too high for the algorithm to track individual particles. Therefore some trajectories found with ‘*OpenPTV*’ can only be 2 or 3 units long. The variable ‘minlength’ specifies the minimum length of a trajectory still to be taken into consideration. In order to calculate the velocities and accelerations of the particles, the time step between each frame ‘dt’ is also required as input (e.g. If the images were captured at 10 000 frames per second ‘dt’ = 1/10 000 s).

After the trajectories have been stored into a MATLAB struct following options to analyze and visualize the trajectories are recommended:

1. Convert the u, v, w velocities from mm/s to m/s using the 'vel_to_ms.m' function
2. Clean data set from misinterpreted trajectories. Depending on the tracking parameters used in '*OpenPTV*' it can occur that the tracking algorithm links particles together which actually are not the same. While this usually can be seen with the naked eye, it is not too easy to erase the bad trajectories from the data set. The function '**clean_trajectories.m'** deletes trajectories with an acceleration component in y direction above a certain threshold. It should be noted that the investigated direction as well as the threshold set, depend on the geometry and the flow properties at hand. Thus, the 'if condition' in the function needs to be changed accordingly before use.
3. Investigate the amount of particles ejected over time and per image using the functions **'particles_per_time.m**' and '**read_particle_amount.m'**.
4. Calculate the angle at which the particles are ejected from the sample with the function '**ejection_angles.m'**.

In order to visualize the data processed in MATLAB, it is found to be helpful to extract the data from the MATLAB structure to ASCII text files. Those text files can then be read into Tecplot. Two functions which arrange the data for tecplot are provided in the post processing folder. Function '**traj_to_ascii.m**' writes each trajectory into a separate text file, while function '**time_to_ascii.m**' will store all particles, including their locations and velocities for each time step. Some example plots are provided below.

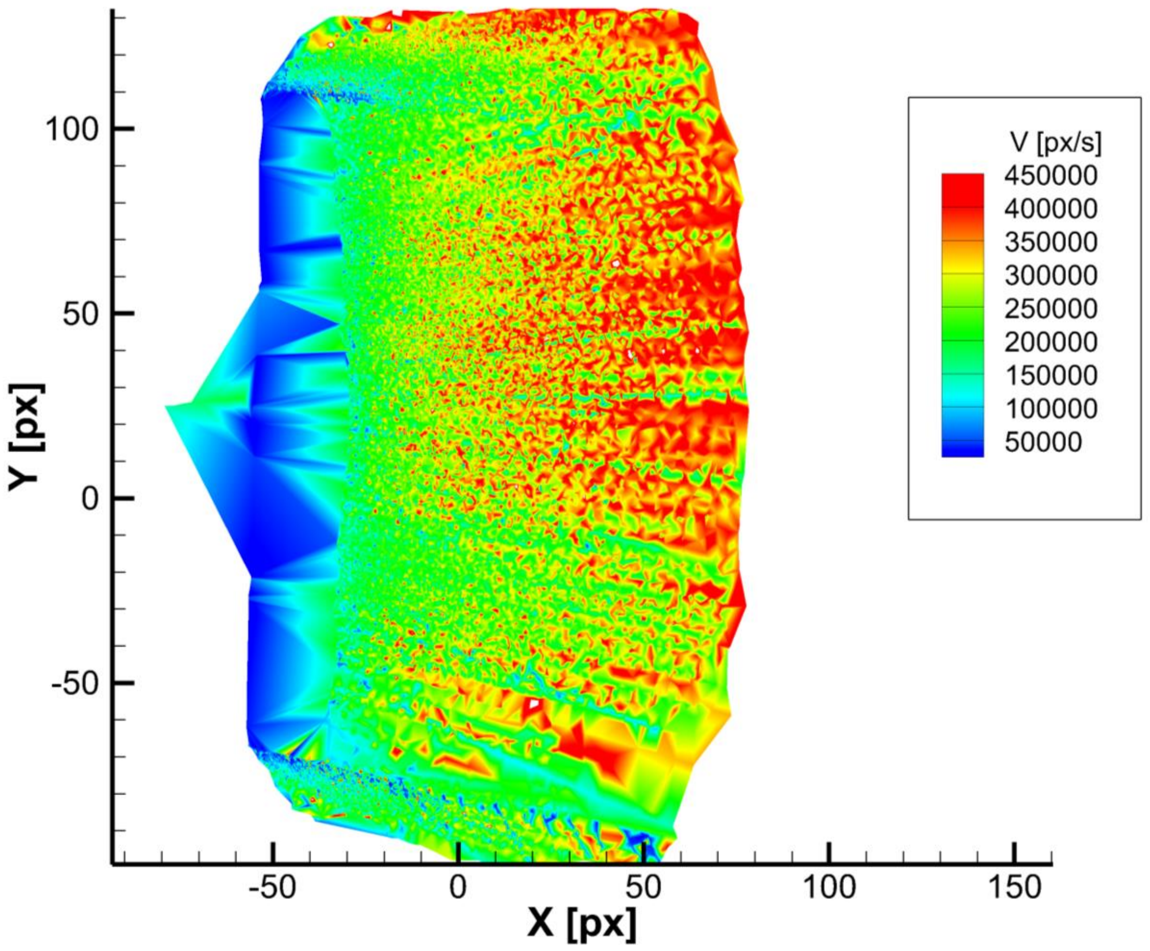
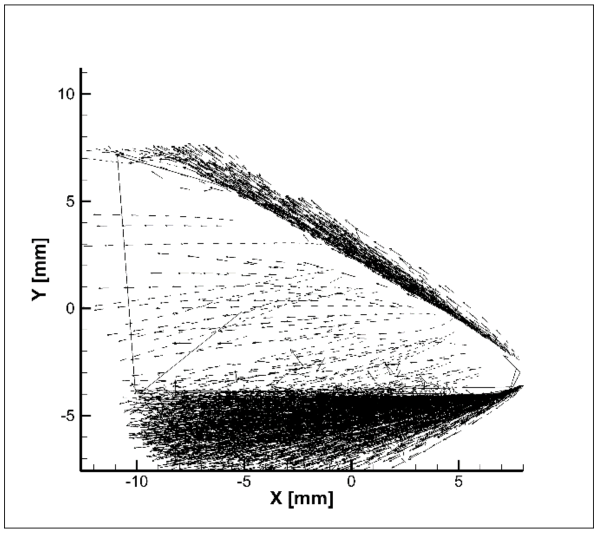


Figure 17 - 2D velocity average plot



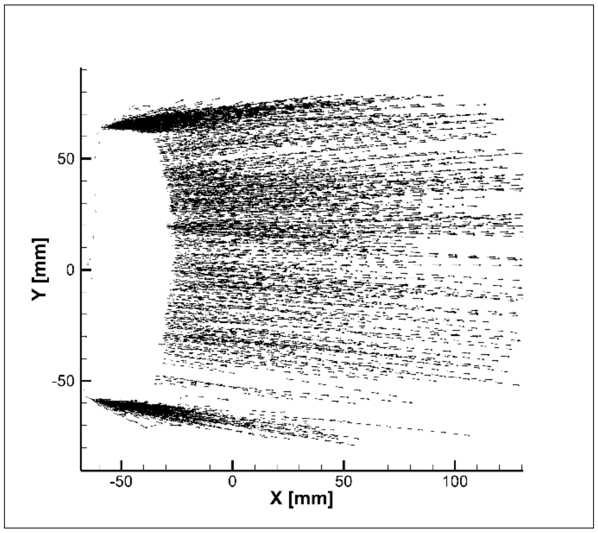


Figure 18 - 2D Velocity vectors (Run-–23side)

Figure 19 - 2D Velocity vectors from (Run23-bottom)

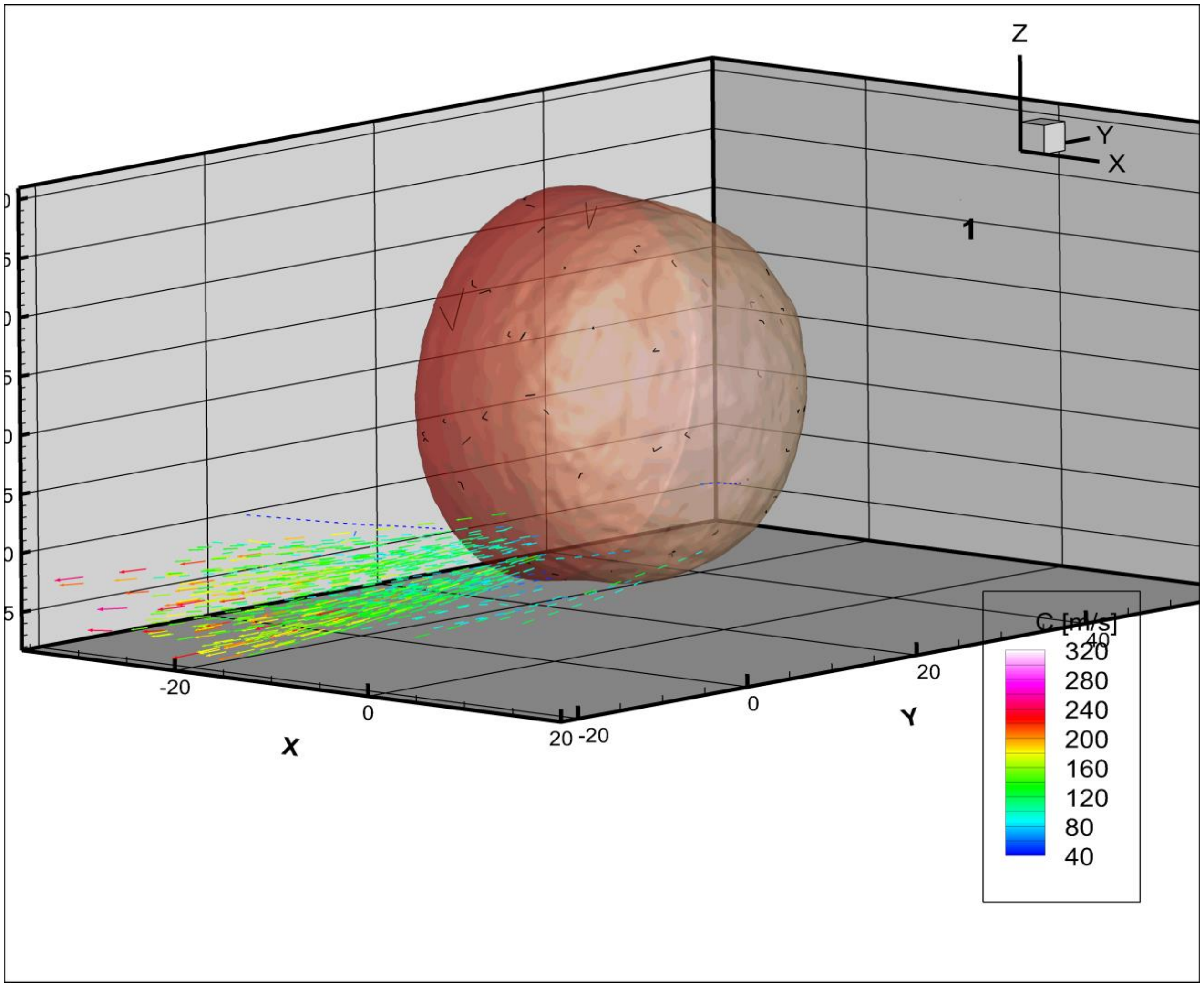


Figure 20 - 3D Velocity vector including sperical test sample scan of Run 11

Identified problems and suggestions to solve them

Several problems and challenges emerged during this project, due to the fact that all experimental data processed has been recorded earlier. In retrospective certain measures suggested in this section would facilitate the process of producing trajectories significantly.

The characteristics of a hypersonic flow implies the use of high-speed cameras in order to be able to capture particle motions. Since the imagery investigated is recorded at 44 000 frames per second the resolution of the images is very low. With 320x192 pixels the respective pixel size is only about 0.25mm. Considering the size of the spalled particles this is critical, as a lot of particles are of equal or even less size which makes them difficult to track consistently. The easiest solution to this problem is obviously to increase the overall resolution of the images. Yet, at 44 000 FPS this will hardly be possible as a higher resolution comes with larger image sizes and thus the images cannot be stored in time. As digital technologies and storage capabilities advance quickly, increasing the image resolution might be a feasible option in a few years.

A more appropriate and also practical solution is to introduce a third or even a fourth camera to the experimental setup. This way it is possible to reduce the number of ambiguous particles and increase the spatial resolution of the experiment. According to (Maas no.1 - 1993) a three or four-camera system can improve the accuracy of particle coordinates by a factor of 1.3 or 1.7 compared to a two camera system.

The major difficulty is the calibration of the camera system. Figuring out a proper initial guess costs a lot of time. Especially if the user is unexperienced and new to optics and calibration. It is easier if a calibration tool which determines the extrinsic and intrinsic camera parameters is used. The initial guess found can then be entered directly into '*OpenPTV*'. Unfortunately these tools usually need a set of checkered board images to iteratively find the camera parameters. Since only one calibration image of the calibration target was taken during the experiment, it was impossible to utilize calibration tools. For further experiments it is highly recommended to get familiar with calibration tools as the 'Camera Calibration Toolbox' for MATLAB. Furthermore, calibrating a camera system can be practiced in the lab rather simple, because it does not even require high-speed cameras. If time at the testing facility is short one can practice calibrating a camera system beforehand and test whether it is possible to determine extrinsic and intrinsic parameters with the calibration toolbox.

References

Maas, H. G., A. Gruen, and D. Papantoniou. 1993. "Particle tracking velocimetry in three-dimensional flows - no.2." *Experiments in Fluids* 133-146.

Maas, H. G., A. Gruen, and D. Papantoniou. no.1 - 1993. "Particle tracking velocimetry in three-dimensional flows." *Experiments in Fluids* 133-146.

Tsai, Roger Y. 1987. "A Versatile Camera Calibration Technique for High Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses." *IEEE Journal of Robotics and Automation, Vol. RA-3, NO. 4 .*

Willneff, Jochen. 2003. "A spatio-temporal matching algorithm for 3 D particle tracking velocimetry." *Mitteilungen- Institut fur Geodasie und Photogrammetrie an der Eidgenossischen Technischen Hochschule Zurich .*