MyPTV user manual

20th March 2022

Contents

1 3D-PTV principles

The 3D-PTV method is used to measure trajectories of particles in 3D space. It utilises the pronciples of stereoscopic vision in order to reconstruc 3D positions of particles from images taken from several angles. A scheme of a typical 3D-PTV experiment using a four camera system is shown in Fig. 1a. The "work horse" behind the 3D-PTV method is the colinearity condition, the 3D model. In principle, if we know what is the position and what is the orientation of the camera in 3D space (O' and θ in Fig. 1b), we can use the pin-hole camera model to relate the image space coordinates of a particle (η, ζ in Fig. 1b) to the ray of light connecting the imaging center and the particle. Then, if we have more than one camera, the particle will be located at the intersection of the two rays. Detailed information is given in [1, 2].

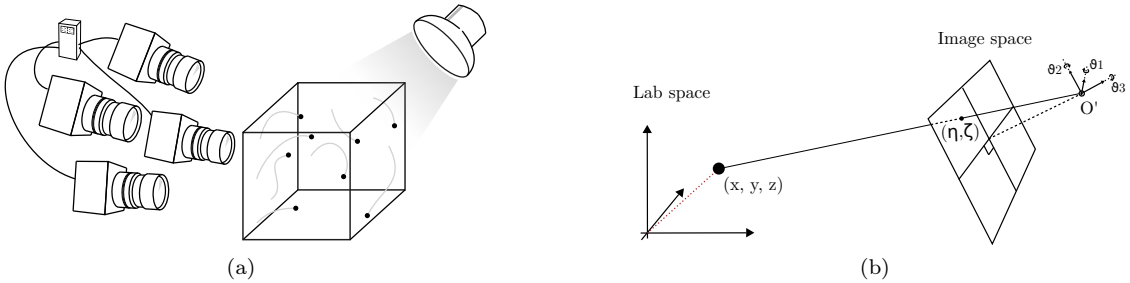

Figure 1: (a) A schematics of a 3D-PTV experiment. (b) A schematic description of the 3d model, the pin-hole camera model.

Once the experiment, namely data aquisition, is done, there are six intrinsic steps to follow in order to complete the analysis. The six steps are outlined in Fig. 2. In Camera calibration, we use images of known calibration targets to estimate the position, orientation and internal parameters of the cameras. In particle segmentation we use image analysis to obtain the paritlces' image space coordinates (η, ζ). In the Particle matching step we use the ray crossing principle to decide which particle image in each of the cameras correspond to the same physical particle, and triagulate their positions through stereo mathcing. In particle tracking we connect the positions of particles in 3D space to form trajectories. In data conditioning we might use smoothing and re-tracking algorithms to enhance the quality of our data according to some physical heuristics. Lastly, we can analyze the data to obtain information on the physics of the particles we are studying. The MyPTV package is meant to handle the first five of these steps.

The sections that follow outline the code used to handle the 3D-PTV method in MyPTV.

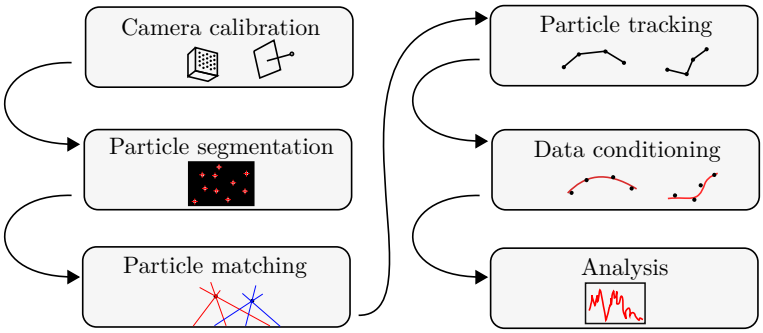

Figure 2: Basic steps in the analysis of PTV raw data into particl trajectories and scientific output. The first five steps are handled by MyPYV.

Table 1: Description of mathematical notation.

Symbol	Description
$\vec{r}$	Particle position in the lab space coorindates
$\vec{O}$	Position of a camera's imaging center
η, ζ	image space coordinates (pixels) of a particle
x_h, y_h	Correction to the camera's imaging center (in pixels)
f	The camera's principle distance divided by the pixel size
$\vec{e}(\eta, \zeta)$	A nonlinear correction term to compensate for image distortion and multimedia problems.
$[R]$	The roation matrix which corresponds to the camera orientation vector.

2 Imaging module - `imaging_mod.py`

The imaging module is used to handle the translation from 2D image space coordinates to lab spcae coordinates and vice-versa. For that, we use the following mathematical model:

$$\vec{r} - \vec{O} = \left(\begin{bmatrix} \eta + x_h \\ \zeta + y_h \\ f \end{bmatrix} + \vec{e}(\eta, \zeta) \right) \cdot \Big[R \Big] \tag{1}$$

where the description of the notations is given in Table 1. The matrix $[R] = [R_1] \cdot [R_2] \cdot [R_3]$ is the rotation matrix calculated with the components of the orientation vector, $\vec{\theta} = [\theta 1, \theta 2, \theta 3]$. In addition, the correction temr $\vec{e}$ is assumed to be a quadratic polynomial of the image space coordinates:

$$\vec{e}(\eta, \zeta) = [E] \cdot P(\eta, \zeta) = \begin{bmatrix} E_{11} & E_{12} & E_{13} & E_{14} & E_{15} \\ E_{21} & E_{22} & E_{23} & E_{24} & E_{25} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \eta \\ \zeta \\ \eta^2 \\ \zeta^2 \\ \eta\,\zeta \end{bmatrix} \tag{2}$$

where $[E]$ is a 3×5 matrix that holds the correction coefficients; the last row is filled with zeros because we do not attempt to correct f.

2.1 The `camera` object

An object that stores the camera external and internal parameters and handles the projections to and from image space and lab space. Inputs are:

1. `name` - string, name for the camera. This is the name used when saveing and loading the camera parameters.
2. `resolution` - tuple (2), two integers for the camera number of pixels
3. `cal_points_fname` - string (optional), path to a file with calibration coordinates for the camera.

The important functionalities are:

1. `get_r(eta, zeta)` - Will solve eq. 1 for the orientation vector $\vec{b} = \vec{r} - \vec{O}$, given an input of pixel coordinates (η, ζ).
2. `projection(x)` - Will reverse solve equation (1) to find the image space coordinates (η, ζ), of an input 3D point, (`x`=$\vec{r}$).
3. `save(dir_path)` - Will save the camera parameters in a file called after the camera name in the given directory path, see Fig. 3.

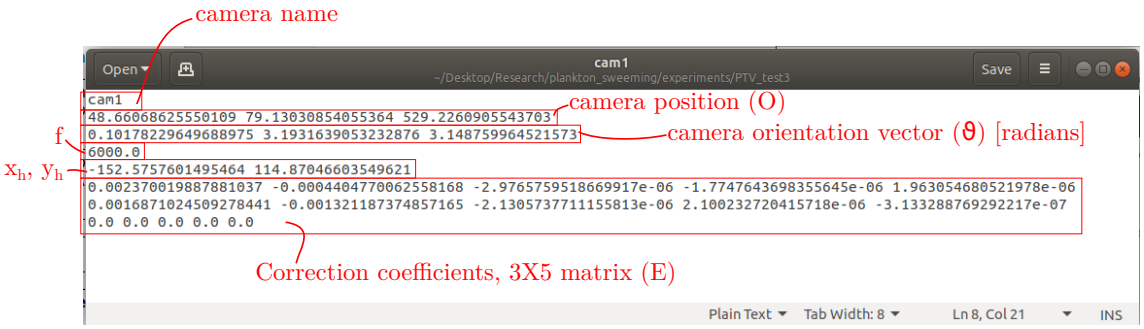

Figure 3: The structure of a camera file. The files are simple text files where each row corresponds to a specific paramter and the values in each row are separated by a whitespace.

4. `load(dir_path)` - Will load the camera parameters in a file called after the camera name in the given directory path, see Fig. 3.

After calibration we can save the camera parameters on the hard disc. The camera files have the structure shown in Fig. 3.

2.2 The `imsys` object

An object that holds several camera instances and can be used to perform stereo-matching. The important functionalities are:

1. `stereo_match(coords, d_max)` - Takes as an input a dicionary with coordinates in image space from the several cameras and calculates the triagulation position.
   The coordinate dicitonary has keys that are the camera number and the values which are the coordinates in each camera. d_max is maximum allowable distance for the triangulation.

3 Camera calibration - `calibrate_mod.py`

Holds the `calibrate` object that is used to find the camera calibration parameters.

3.1 The `calibrate` object

Used to solve for the camera parameters given an input list of image space and lab space coordinates. The inputs are:

1. `camera` - An instance of a `camera` object which we would like to calibrate.
2. `lab_coords` - a list of lab space coordinates of some known calibration target.
3. `img_coords` - a list of image space coordinates that is ordered in accordance with the lab space coordinates.

The important functionalities are:

1. `searchCalibration(maxiter=5000, fix_f=True)` - When this is run, we use a nonlinear least squares search to find the camera parameters that minimize the cost function (item 3 below). This function is used to find the $\vec{O}$, $\vec{\theta}$, f, and x_h, y_h parameters (in case `fix_f=False`, it will not solve for f. `maxiter` is the maximum number of iterations allowed for the least squares search.
2. `fineCalibration(maxiter=500)` - This function will solve for the coefficients of the quadratic polynomial used for the nonlinear correction term ($[E]$).

3. `mean_squared_err` - This is our cost function, being the sum of distances between the image space coordinates and the projection of the given lab space coordinates.

To find an optomal calibration solution, we might need to run each function several times, and run the coarse and fine calibrations one after the other until a satisfactory solution is obtained. Once it is obtained, we should keep in mind to save the results using the `save` functionality of the `camera` object.

References

[1] M. Virant and T. Dracos. 3D PTV and its application on lagrangian motion. *Measurement*, 8:1552–1593, 1997.

[2] H. G. Mass, D. Gruen, and D. Papantoniou. Particle tracking velocimetry in three-dimensional flows part i: Photogrammetric determination of particle coordinates. *Experiments in Fluid*, 15:133–146, 1993.