Documentation of 3D Particle Tracking Velocimetry

Technical aspects, Installation, Handling, Software Architecture

by Jochen Willneff, October 2003

1. Installation and Handling of the PTV-Software

The ptvmanual.pdf contains information to use the software and gives a description of the input data file which have to be provided.

The test data set contains the following:

```
Cam1                Cam3.addpar         man_ori.dat
Cam1.addpar         Cam3.ori            parameters/
Cam1.ori            Cam4                ptvmanual.pdf
Cam2                Cam4.addpar         res/
Cam2.addpar         Cam4.ori            start.bat
Cam2.ori            calFieldApril.txt
Cam3                img/
```

Images for calibration, camera orientation data, files for additional parameters as well as the coordinate file of the points on the reference. In `man_ori.dat` the manually measured image coordinates for the pre-orientation (for calibration purpose) are stored.

- subdirectory	`/img`	contains the image sequences
- subdirectory	`/res`	for storage of results
- subdirectory	`/parameters`	contains the parameter files

The data mentioned above are the data for the experiment itself. To avoid confusion this data should be kept separated to the software data!

The software for PTV is stored under `/tk84ptv`. To start the software >> click double on `start.bat`, which establishes the link to the software. Project and software data should not be confused. To start the software it is sufficient that a `start.bat` file, thus the software (and code) can be stored independently.

2. Tcl/Tk-Installation

The install executable for Tcl/Tk is `ActiceTcl8.4.2-win32-ix86.exe` in this directory. Or can be downloaded from a webpage. Download under:

`http://downloads.activestate.com/ActiveTcl/Windows/8.4.2/`

After installing Tcl/Tk 8.4.2 all files with extension `#.tcl` should appear with the Tcl/Tk-symbol (feather). Otherwise repeat installation. Make sure that the

flag for the extensions (`*.tcl`) is included in the path. Otherwise not all needed dll-files can be found from arbitrary directories on the PC.

3. Compilation of TIFF library

The handling of TIFF images requires a according library `libtiff.lib` and the files `tiff.h`, `tiffio.h` and `tiffvers.h`. If these files are not available, they can be generated as described in the following. Download file `tiff-v3.5.7.zip` from the webpage `www.libtiff.org`. It's also possible to use the newer version `tiff-v3.6.0.zip`. Both versions were tested for Windows2000 and WindowsXP. After unzip change to `/libtiff`, where you will find the file named `makefile.vc`.
In the header of this file the command (`nmake /f makefile.vc all`) to compile the library with `nmake` is given. It is also possible compile with Microsoft Visual C++. To open the makefile in the right way, change the filename from `makefile.vc` to `makefile.mak`. Only the files `libtiff.lib`, `tiff.h`, `tiffio.h` and `tiffvers.h` are necessary for PTV (and may be copied in the directory `/tk84ptv/src_c`, if done so no paths have to be adjusted for the compilation of PTV).

4. Compilation of the source code of PTV

In the `/tk84ptv` directory You will find the following data:

index	script to generate tclIndex (/ might be missing in the generated tclIndex!)
ptv.tcl	main script to start graphical user interface (Windows)
ptvunix.tcl	dito for Unix
start	start file for Unix
tclIndex	Index with relative paths to Tcl functions
/src_c	source code directory
/src_tcl	tcl script directory

The contents of the /src_c:

change_parameter.c	jw_main.c	segmentation.c
checkpoints.c	jw_ptv.c	sortgrid.c
correspondences.c	libtiff.lib	unixmakefile
demo.c	lsqadj.c	tiff.h
draw.c	mousefunction.c	tiffio.h
epi.c	multimed.c	tiffvers.h
globals.h	orientation.c	tools.c
homemakefile	peakfitting.c	track.c
homemakefile.mak	pointpos.c	trafo.c
image_processing.c	ptv.c	ttools.c
imgcoord.c	ptv.h	typedefs.h
intersect.c	ray_tracing.c	vrml.c
jw_ImgFmtTIF.c	rotation.c	

The contents of the `/src_tcl`:

button.tcl	display.tcl	mainpar.tcl
calpar.tcl	draw.tcl	trackpar.tcl

The source code is written in C in combination with Tcl/Tk. The directory `/src_c` contains a makefile (`homemakefile.mak`) which can be open with Microsoft Visual C++. During opening this file, a new workspace will be generated.
NOTICE! The following paths to the libs in the makefile have to be adjusted:

```
INC_DIR1 = C:\Tcl\include\

TCL_LIB = C:\Tcl\lib\tcl84.lib
TK_LIB = C:\Tcl\lib\tk84.lib
TIFF_LIB = H:\tk84ptv\src_c\libtiff.lib
```

An alternative is the compilation with nmake in the DOS prompt. For compilation in the DOS prompt first perform vcvars32.bat for initalization, after that:

```
nmake -f homemakefile.mak
```

IMPORTANT! Before running the software some paths have to be set. `start.bat` may contain:

```
G:/tk84ptv/src_c/jw_prog G:/tk84ptv/ptv.tcl
```

This has to be modified to the actual position on the PC. First the path to the `jw_prog.exe` followed by the path to the `ptv.tcl` script file.

In addition, change path in first line of `ptv.tcl` (use # for comments):

```
set auto_path "G:/tk84ptv . $auto_path"
```

Change to according path on PC! The `start.bat` should be copied to the project data directory. Start with double click!

5. 3D PTV source code

Authors: Hans-Gerd Maas / Jochen Willneff
Address: Institute of Geodesy and Photogrammetry
ETH – Hoenggerberg
CH - 8093 Zurich

Functions in ***jw_ptv.c***:

init_proc_c

initialization, allocation of memory for image data, reading of parameter files

- parameters/ptv.par
- parameters/criteria.par
- parameters/sequence.par

call of ***parameter_panel_init***, to fill Tcl/Tk sheets

start_proc_c

reading of parameters/ptv.par

create file names for low and high pass image and ori/addpar data

Reading orientation data from file, reading image
Allocation of tracking structure

pre_processing_c

reading of parameters/unsharp_mask.par, default value 12, if not existing, call of ***highpass***, with optional display

detection_proc_c

reading of parameters/pft_version, switch pft for peak fitting

three cases:

- 3, call of ***peak_fit_new***
- 0, call of ***simple_connectivity***
- 1, call of ***targ_rec***

mostly ***peak_fit_new*** was used recently
call of ***quicksort_target_y***, to sort detected particles

correspondences_proc_c

Transformation from pixel to metric coordinates with call of
- ***pixel_to_metric***
- ***correct_brown_affin***

sort coordinates for binary search with ***quicksort_coord2d_x***
reading of parameters/criteria.par

calculation of look up table for multimedia radial displacement
- ***init_mmLUT***, performed only once!

search for correspondences with most important function!
- ***correspondences_4***

create #_targets data for each image, writing data to *_targets files

determination_proc_c

create res/dt_lsq file, call of function for 3D coordinate determination
- ***det_lsq***, writing data to res/dt_lsq file,

sort coordinates for binary search in epi line segment drawing
- ***quicksort_coord2d_x***

sequence_proc_c

reading of parameter file parameters/sequence.par, first and last time step of sequence

create file names, res/rt_is.#
processing of each time step with
- ***read_image***
- ***pre_processing_c***
- ***detection_proc_c***
- ***correspondences_proc_c***
- ***determination_proc_c***

calibration_proc_c

reading of parameters/unsharp_mask.par, default value 12, if not existing

8 cases, switch set by sel:

- 1, read calibration parameter file parameters/cal_ori.par
  create file names, call of ***read_image***

- 2, detection procedure, call of ***pre_processing_c***, target recognition by call of ***targ_rec***, reading parameters/detect_plate.par, writing #pix files for each image

- 3, manual orientation, read parameters/man_ori.par, interactive measurement of four reference points in each image, writing pixel coordinates to file man_ori.dat, can be used for further orientation calculation, see case 4

- 4, read points numbers from parameters/man_ori.par, read pixel coordinates of older pre-orientation from man_ori.dat, display

- 5, sort grid, read coordinate from reference body,
  read orientation and addpar files, calculation of raw orientation
  with 4 points by call of ***raw_orient***, write orientation data,
  call of ***write_ori***, sorting of detected points by
  back-projection by call of ***sortgrid_man***, with display,
  if examine is set to 4, create files for dump dataset,
  allowing layerwise calibration

- 6, orientation, if examine set to 4, reading files for layerwise calibration, else calculation of resection by call of ***orient***, if examine set to 4, read dumped data sets first, then call orient, write results in ori and addpar files

- 7, ***checkpoint_proc***, to display residuals of checkpoints

- 8, draw additional parameter figures, display regular grid and
  (exaggerate) distorted grid

quit_proc_c

free memory, delete unneeded file and quit

6. PTV argument examine

The start.bat file without using the examine option is similar to:

```
H:/prog/tk84ptv/src_c/jw_prog H:/prog/tk84ptv/ptv.tcl
```

Different examine options can be set by the start of
PTV (e.g. `.../jw_prog .../ptv.tcl 4`)

1 (or any number) more details on output during orientation
(calibration), double zoomfactor, creates #_pix files

3 to save low pass image,
doesn't work on Windows system, sorry!

4 creates ASCII output with 3D object point list and
referring image coordinates for calibration with dumped
data sets, detailed descriptions see below.

5 more detailed output for statistical examinations
of Qvv, Qxx at determination of particle positions
and during orientation (calibration)

10 gives information about average and expected number of
touch events and the average number of pixel per target,
at detection of particles

Option 4 is used for the calibration with different z-positions of the reference body. The examine option appears in the following source code files:

```
draw.c: if ( examine  && zoom_f[nr] > 2 )
jw_ptv.c: int    examine = 0;                    /* for
more detailed output */
jw_ptv.c:  valp = Tcl_GetVar(interp, "examine",
TCL_GLOBAL_ONLY);
jw_ptv.c:  examine = atoi (valp);
jw_ptv.c:  if (examine == 4)
jw_ptv.c:      if (examine == 4)
jw_ptv.c:      if (examine)     for (i=0; i<n_img; i++)
jw_ptv.c:      if (examine == 4)
jw_ptv.c:         if (examine == 4)
jw_ptv.c:         if (examine != 4)
jw_ptv.c:         if (examine == 4)
mousefunction.c:  if (examine)  zf *= 4;
orientation.c:  if (examine)    for (i=0; i<16; i++)
peakfitting.c:  if (examine==10)
pointpos.c:  if (examine == 5)
segmentation.c:  if (examine == 3)
segmentation.c:  if (examine == 3)
```

How to calibrate with different z-positions of the reference body?

Set examine = 4!

After the **Detection** (case 2, under calibration) `#_pix` files are generated, which lists the detections of each view (no correspondences, only image coordinates), was used for template matching outside of PTV, not used for further processing steps (could be removed or commented in the code).
--->> continue as usual!!!

During **Sortgrid** (case 5, under calibration) a file `dump_for_rdb` is created. This is a list of the 3D points with the according 2D image coordinates of each view. Is not used for further steps.
--->> continue as usual!!!

Important!!! In **Orientation** (case 6, under calibration). For each z-position an according 3D coordinate file has to be provided (`File of Coordinates on Plate`, at **Calibration Parameters** has to be adjusted for that!!!)

During **Orientation** with examine set to 4, please consider the DOS prompt. You will be asked:

```
Resection with dumped datasets? (y/n)
```

Answer with n in the DOS prompt to write data to disk, which will later be used for the common calibration. For each camera you have to answer individually (up to four times depending on the number of cameras)!

The files, which are created have the following name structure:

`resect_#.crd`, containing the corrected metric image coordinates (e.g. `resect_Cam1.crd`).

`resect_#.fix`, containing the 3D coordinates of the active reference body file (e.g. `resect_Cam1.fix`).

After the generation of these files, the different file names have to be adjusted to a sequential structure:

`resect.fix0`	3D reference body file of z-position 0
`resect.fix1`	3D reference body file of z-position 1
`resect.fix2`	3D reference body file of z-position 2

... and so on.

For each camera and each z-position a set of metric image coordinates has to be provided with the following names:

`resect_0.crd0`	metric image coordinates of camera 0 at z-position 0
`resect_0.crd1`	metric image coordinates of camera 0 at z-position 1
`resect_0.crd2`	metric image coordinates of camera 0 at z-position 2
`resect_1.crd0`	metric image coordinates of camera 1 at z-position 0
`resect_1.crd1`	metric image coordinates of camera 1 at z-position 1
`resect_1.crd2`	metric image coordinates of camera 1 at z-position 2
`resect_2.crd0`	metric image coordinates of camera 2 at z-position 0
`resect_2.crd1`	metric image coordinates of camera 2 at z-position 1
`resect_2.crd2`	metric image coordinates of camera 2 at z-position 2
`resect_3.crd0`	metric image coordinates of camera 3 at z-position 0
`resect_3.crd1`	metric image coordinates of camera 3 at z-position 1
`resect_3.crd2`	metric image coordinates of camera 3 at z-position 2

... and so on.

After the generation of this file structure, it's recommended to save these files first before continuing with the calibration. Restart PTV, press **Show Calib. Image** and continue directly with **Orientation**, consider the DOS prompt. Now answer to

```
Resection with dumped datasets? (y/n)
```

with y for each individual camera (again up to four times depending on the number of cameras), `#.ori` and `#.addpar` file are generated according to the given cameraname (e.g. `Cam1.ori, Cam1.addpar`). The orientation and additional parameters for each individual cameras are calculated in a common adjustment of all z-positions.

7. Using C code in connection with Tcl/Tk

The source code of PTV, which is written in C is connected to a graphical user interface realized in Tcl/Tk. This requires the declaration and initilization of commands.
In `jw_main.c` the C function ***main*** is called. In this function only the initialization for the Tcl/Tk application is performed. The ***main*** just calls the function ***Tk_Main(argc, argv, Tcl_AppInit)***, also included in `jw_main.c`, which starts the function ***Tcl_AppInit(interp)***, (in `jw_main.c`). ***Tcl_AppInit(interp)*** calls the function ***jw_Init*** (again in `jw_main.c`).
With the call of ***jw_Init*** the additional Tcl/Tk commands are defined. For example:

```
Tcl_CreateCommand(interp, "start_proc_cmd", start_proc_c,
    (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL);
```

The name of the defined Tcl/Tk command is ***start_proc_cmd***. This command can be used in the Tcl/Tk script. The related C function ***start_proc_c*** is evaluated, when ***start_proc_cmd*** is called.
The C function is defined as Tcl/Tk command in `ptv.h` like this:

```
extern Tcl_CmdProc start_proc_c;
```

In addition the function itself has to be defined in `global.h`:

```
int start_proc_c();
```

For the actual implementation the function needs some Tcl/Tk related arguments:

```
int start_proc_c(ClientData clientData, Tcl_Interp*
interp, int argc, const char** argv)
```

To call the function from the Tcl/Tk script (in this case by pressing the button "**Start**", see in `ptv.tcl`):

```
button .mbar.start -text "Start" -command
"start_proc_cmd; bindingsstart "
```

The option `-command` also allows the combination with other commands or function calls. In this example with **bindingsstart**, which defines the current mouse functions.

If these C functions are called from some other C function and not from the Tcl/Tk script do like this:

```
pre_processing_c (clientData, interp, argc, argv);
```

Exchange of variables between C and Tcl/Tk

In the C source code and the Tcl/Tk script variables have to be exchanged. This is the case for reading and writing the input data from the input files and the Tcl/Tk sheets.
The according variable is defined in the C code as well as in Tcl/Tk (e.g. `n_img` in C refers to `mp(ncam)` in Tcl/Tk). The reading of the input files is realized in C. The exchange to Tcl/Tk is done by the commands:

`Tcl_SetVar2` and `Tcl_GetVar2`

Remarks:
- Tcl/Tk treats the variables as string (therefore `char` type in C)
- `Tcl_SetVar2`, `Tcl_GetVar2` (not `Tcl_SetVar`, `Tcl_GetVar`) are used to be able to define variables in vector structure

Example in `src_tcl/mainpar.tcl`:

```
global mp
```
`mp` is the structure for all main parameters. `mp(ncam)` represents a specific parameter

The exchange works the following way, from C to Tcl/Tk (see in function ***parameter_panel_init*** in `src_c/change_parameter.c`):

```
int parameter_panel_init(Tcl_Interp* interp)
{
 char val[256];

 /* read 20 parameters from ptv.par */
 fp1 = fopen_r ("parameters/ptv.par");

 fscanf (fp1, "%s", val);
 Tcl_SetVar2(interp, "mp", "ncam", val, TCL_GLOBAL_ONLY);

 fclose (fp1);
 return TCL_OK;
}
```

The parameter is read from the input file to C variable `val`, its value is transferred (`Tcl_SetVar2`) to Tcl/Tk variable `mp(ncam)` and is available in the Tcl/Tk scripts. From Tcl/Tk to C (see in function ***done_proc_c*** in `src_c/change_parameter.c`):

```
int done_proc_c(ClientData clientData, Tcl_Interp*
interp, int argc, const char** argv)
{
 const char *valp;

 /* rewrite all parameter files */
 fp1 = fopen ("parameters/ptv.par", "w");

 valp=Tcl_GetVar2(interp, "mp", "ncam", TCL_GLOBAL_ONLY);
```

```
 fprintf (fp1, "%s\n", valp);
 fclose (fp1);
 return TCL_OK;
}
```

Parameter `mp(ncam)` from Tcl/Tk is transferred (`Tcl_GetVar2`) to C variable `valp` and written to file.

ATTENTION!!! `valp` is a `char` type. For later use in C (for calculations) this variable may has to be converted, for e.g. to integer:

```
 valp = Tcl_GetVar(interp, "examine",  TCL_GLOBAL_ONLY);
 examine = atoi (valp);
```

8. 3D PTV data acquisition and setup parameters

The processing of the image sequences from PTV experiments requires some information about the hardware configuration. From the used cameras at least a rough estimation of the focal length, size of the pixel in x- and y-direction and the image size (in pixel) itself has to be specified.
To be able to model the multimedia geometry the refractive index of the fluid and glass plate as well as the glass plate thickness has to be known.
For calibration purposes the coordinates of the used reference body have to be specified. The boundary of the glass plate and fluid defines the x-y-plane (z = 0). The coordinate file of the reference body has to be edited according to its actual position, which requires the knowledge of the distance of the reference body to the inner glass plate side.

The image sequences have to be provided according to the following convention. For the sequence of each camera the filename is a combination of a base name and a number of the current time step. No extensions after the current number of the time steps are possible.

Example, basename: cam1., time steps from 1 to 1001, the files have to be provided with the following names:

cam1.001 … cam1.009, cam1.010 … cam1.099, cam1.100 … cam1.999, cam1.1000, cam1.1001