OpenSRD

Installation and user guide

Lukas Engelen

December 28, 2016

Contents

Chapter 1

Introduction

OpenSRD, abbreviation of Open Source Shape from Refractive Distortion, is a software tool developed for the spatio-temporal reconstruction of a water surface. It started as a master thesis project of Lukas Engelen at the Hydraulics Laboratory of the University of Ghent and was further developed in his following PhD-research. The entire program is written in C++ of which the source code is free available online[1]. Please feel free to contact us in case you have comments to improve the program or would like to contribute to the project.

We would like to acknowledge that the program makes use of three open source libraries, namely OpenCV (Bradski [3]), ALGLIB (ALGLIB Project [1]) and MINPACK (Burkardt [4]).

1.1 Need for accurate and image-based surface reconstruction techniques

Accurate measurement of the deformation of a liquid surface is important for numerous hydraulic and oceanographic research fields. Although numerical models offer the advantage of being able to simulate much more variations in hydraulic and geometric boundary conditions, experimental research remains required to validate these numerical codes. Additionally, experimental research remains indispensable to obtain fundamental knowledge and discover new aspects of the studied flow phenomena.

Conventional measurement techniques, such as capacitance, resistance or pressure gauges and ultrasonic distance sensors, only provide point measurements of the local water height. Additionally, the former are intrusive and therefore not recommended for a detailed study of the surface characteristics. In contrast, optical techniques allow to capture the entire surface area, without interference with the studied phenomenon. We pursued an image-based technique, making no use of laser-scanners etc., due to the ready available, low-cost equipment without the safety issues related to lasers. As such, these techniques are easy implementable in existing laboratory setups with only limited investment.

1.2 Short summary of the reconstruction methodology

1.3 Reconstruction process

Setting up of the technique/initialization

1. Install a feature pattern on the bottom of the reconstructed volume
   In order to locate the feature points in the images, a feature or corner detection algorithm is required. We currently implemented a corner detection algorithm based on CHESS (Bennett and Lasenby [2]), which is specifically designed to locate checkerboard-patterns (2×2 white and black squares). Therefore,

[1] Available at: `https://github.com/lengelen/OpenSRD`

it is recommended to choose a feature pattern in which the feature points correspond with the internal checkerboard corners. In case you would like to use a different feature pattern, note that you will have to implement your own feature detection algorithm and make the necessary adjustments to the source files.

2. Install an illumination system
   One possibility is to light the pattern from the side/top to make it visible in the view of the cameras that are located above the reconstructed water volume. It must be noted that this had the disadvantage of reflections on the specular water surface which complicates feature detection. Our recommendation is therefore to light the pattern from below, using a uniform light panel located in or below the bottom of the tank. In case the white and black part of the checkerboard pattern are respectively transparent and nontranslucent for the light source, the feature pattern should be clearly distinguishable without the risk of reflections on the surface. Another possibility is to use UV light and use a UV-sensitive pattern, which reflects the energy of the UV light in the visible light spectrum. For more information, we refer to Engelen [5].

3. Calibration of the (multi-)camera system
   The calibration of the camera consists of determining the intrinsic camera parameters and distortion coefficients of each camera. This requires a sequence of images depicting a calibration grid and running the executable *Calibration* (for more information, see section 3.1). Additionally, the extrinsic camera parameters need to be determined based on an image of a known reference grid. Make sure that the position of the cameras is not altered between this calibration step and the measurement campaign. The entire surface reconstruction methodology is highly dependent on the camera position in the chosen word coordinate system, which needs to be known as accurately as possible.

Measuring the water surface

DATA ACQUISITION

1. a) Illuminate the pattern
      This can be done using a continuous or pulsed light source. Pulsed light has the advantage that in case high-lumen LED's or COB's are used, the heat production is much smaller which makes the requirements for the cooling system less stringent. Additionally, the LED's can be used in overdrive mode, resulting in more than 100 % brightness which improves the contrast in the images. This allows shorter shutter times and therefore reduces the risk on image blur.

   b) Take images of water surface
      The covered area and possible frame rate of the images depend on your image acquisition system, consisting of cameras, data link and storage device. The frame rate of the cameras has a direct impact on the temporal resolution of the surface reconstruction. Please keep in mind that also the shutter time needs to be short enough to avoid motion blur, which requires strong lighting of the scene (previous step).

   c) Store locally or transmit through data link to storage device
      Store the data (images) in the local memory of your camera or send the images through a data cable (with e.g. a switch in between) to your local storage device (e.g. laptop).

   The experiments performed in the Hydraulics laboratory of the University of Ghent were performed with Basler Ace Gige cameras (acA1920-25gc) in combination with a Gigabit switch (brand TP-Link) which allowed to obtain a frame rate of X Hz. Using a combination of a (pulsed) high-lumen COB array (Cree CXA1304 LED) and a diffusive plate, the shutter time of the camera could be reduced to X Hz.

2. DATA PROCESSING

   a) Finding features using a corner detection algorithm
   Currently, a corner detection algorithm is implemented which is merely based on the CHESS-algorithm of Bennett and Lasenby [2].

   b) Feature matching between consecutive time frames
   In order to reconstruct a time sequence of a dynamic surface, feature points need to be tracked over the image sequence. The location of a certain feature point is for every frame (image) predicted using a linear prediction model, based on its previous location and assuming a constant velocity within the image plane. In case a local maximum of the corner strength measure is sufficiently close to this predicted position, the corresponding maximum is accepted as the updated image point **q'**.

   c) Perform the multivariate optimization
   Run the optimization of the surface coefficients, using the error metric and optimization parameters that are chosen.

Output

- Camera pose estimation
- Detected feature points per image
- Surface coefficients
- Related average total error (collinearity or disparity difference metric)

1.4 Advantages of the technique

- Low-cost
- Non-intrusive
- Scalable
- Accurate and computational efficient
- Compatible with PTV or PIV velocity measurements
- Applicable to modal analysis of free surface oscillations

1.5 Bibliography

[1] ALGLIB Project. ALGLIB: a cross-platform open source numerical analysis and data processing library, 1999. Currently available at: `http://www.alglib.net/`, Last revised: 2016.

[2] S. Bennett and J. Lasenby. Chess – quick and robust detection of chess-board features. *Computer Vision and Image Understanding*, 118:197 – 210, 2014. ISSN 1077-3142. doi: http://dx.doi.org/10.1016/j.cviu.2013.10.008. URL `http://www.sciencedirect.com/science/article/pii/S1077314213001999`.

[3] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. Currently available at: `http://opencv.org/` , Last revised: 2016.

[4] J. Burkardt. MINPACK Least Squares Minimization , 2010. Currently available at:`http://people.sc.fsu.edu/~jburkardt/cpp_src/minpack/minpack.html`, Last revised: 2010.

[5] L. Engelen. Spatio-temporal Image-based Water Surface Reconstruction: Application to Sloshing In Navigation Lock Filling. Master's thesis, University of Ghent, Belgium, 2016. Master of Science in Civil Engineering.

[6] Horace Lamb. *Hydrodynamics*. Cambridge University Press, 1932.

[7] OpenMVG authors. cameras-Pinhole camera model. `http://openmvg.readthedocs.io/en/latest/openMVG/cameras/cameras/`. [Online; Accessed: 2016-02-24; Last Update: 2015].

[8] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Trans. Pattern Analysis and Machine Intelligence (TPAMI)*, 22(11):1330–1334, 2000.

Chapter 2

Installation of source code

2.1 Introduction

The reconstruction algorithm presented in the previous chapter is implemented in OpenSRD as a modular C++ program. The different parts of the reconstruction are split over several modules that are coded in different C++ source files and header files. Although the code which implements the theoretical framework (for more information we refer to Engelen [5]) is written by the OpenSRD developpers, OpenSRD employs functions and classes from other open source libraries. In case you want to contribute in the further development of the OpenSRD library, it is recommended to install and get familiar with these libraries before making changes to the OpenSRD program.

2.2 Required external open-source libraries

OpenCV

OpenCV (Open Source Computer Vision Library) is an open-source BSD-licensed library developed by Bradski (2000) that includes several hundreds of computer vision and machine learning algorithms (Bradski and Kaehler (2008)). The library is cross-platform and mainly written in C++ (although an older C interface exists) but contains bindings for other languages such as Python or JAVA. Some of the application areas include 2D and 3D feature toolkits, camera calibration, facial recognition systems, object identification and motion tracking, ...

Different functions from the OpenCV library will be used for several aspects of the water reconstruction procedure:

- Camera calibration (estimation of the distortion coefficients)
- Camera pose estimation (estimation of the camera position and orientation w.r.t. a predefined reference system).
- Input and output (images, settings, ...)
- Object classes

In order to compile the reconstruction program from source, it is required that OpenCV (version 3.1) is installed. Additionally, the OpenCV's libraries and location of the header files should be added as a flag during compilation. More information on how to install OpenCV and compile while including OpenCV can be found on `http://opencv.org/opencv-3-1.html`.

ALGLIB

ALGLIB is a cross-platform numerical analysis and data processing library and supports multiple programming languages, including C++. In general, the ALGLIB functions are used for several programming challenges such as:

- Data analysis
- Optimization and nonlinear solvers
- Interpolation and linear/nonlinear least-squares fitting
- Linear algebra (direct algorithms, EVD/SVD), direct and iterative linear solvers, Fast Fourier Transform and many other algorithms (numerical integration, ODEs, statistics, special functions)

OpenSRD employs an ALGLIB-implented version of the Levenberg-Marquardt method. This method is used for the multivariate optimization of the surface coefficients of the chosen surface model. The algorithm uses a combination of the steepest descent method with a Newton-Raphson method in order to obtain maximum operational stability and rapid convergence towards the optimal solution. It also offers the possibility to set an upper and lower limit to the search interval of these coefficients. Finally, the algorithm uses a initial guess as starting position which has a minor influence on the final result but has a significant influence on the convergence rate. In case the frame rate of your camera is sufficient, a good initial guess, based on the previously processed time step, allows a more rapid convergence to the next solution due to the temporal smoothness of the water surface.

The ALGLIB's C++ source and header files are also included in the src directory found on Github[1]. These source files need to be compiled together with the reconstruction source files, which for most used compilers (GCC, MSVC, Sun Studio) does not require any additional settings. In other cases, preprocessor definiton might be required for which we refer to the ALGLIB reference manual.

MINPACK

Additionally, we make use of a C++ version of MINPACK by J. Burkardt, originally a FORTRAN library by J. More, B. Garbow and K. Hillstrom, to solve a set of non-linear equations. The required source and header file (minpack.cpp and minpack.h), which contain an OpenSRD modified version of a non-linear solver, are also included in the src directory and need to be compiled together with the rest of the project.

[1] `https://github.com/lengelen/OpenSRD`

Chapter 3

Using the pre-compiled executable

For the people that do not want to contribute and are just interested in using the reconstruction program as a stand-alone water surface measurement technique, running the executable SurfaceReconstruction is sufficient. Please make sure that the settings-file is updated and the required subdirectories are made at the location of the executable. A good calibration of the camera system is crucial to obtain accurate results and is explained in Section 3.1 and Section 3.2.

3.1 Calibration of the camera's

A first step in the reconstruction of the water surface consists of correcting the images for lens distortion. This form of optical aberration requires a correction of the images to obtain the undistorted pixel coordinates of the detected feature points. The distortion coefficients according to Brown's distortion model, which takes both radial as tangential distortion into account, are determined using an OpenCV function and describe the relationship between the undistorted and distorted pixel coordinates (Zhang [8]):

$$x_d = x_u \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1\, x_u\, y_u + p_2(r^2 + 2x_u^2) + s_1\, r^2 + s_2\, r^4$$
$$y_d = y_u \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + p_1(r^2 + 2y_u^2) + 2p_2\, x_u\, y_u + s_3\, r^2 + s_4\, r^4$$

where:

(x_d, y_d) The distorted image point as projected on image plane using specified lens.
It must be noted that these are not the same as the pixel coordinates obtained from the image. The relationship between the (distorted) pixel coordinates and 2D coordinates in the image plane is given by following equation, as will be explained later:

$$u = f_x \cdot x_d + c_x$$
$$v = f_y \cdot y_d + c_y$$

(x_u, y_u) The undistorted image point as projected by an ideal pin-hole camera:

$$u = f_x \cdot x_u + c_x$$
$$v = f_y \cdot y_u + c_y$$

k_n	The n^{th} radial distortion coefficient.
p_n	The n^{th} tangential distortion coefficient.
s_n	The n^{th} thin prism distortion coefficient.
r	$= \sqrt{x_u^2 + y_u^2}$ in case it is assumed that the distortion center is equal to the principal point.

In order to compute the world coordinates of features seen in the images by the cameras, the camera parameters are also required. The camera is considered as a pinhole camera, schematically represented in Figure 3.1.

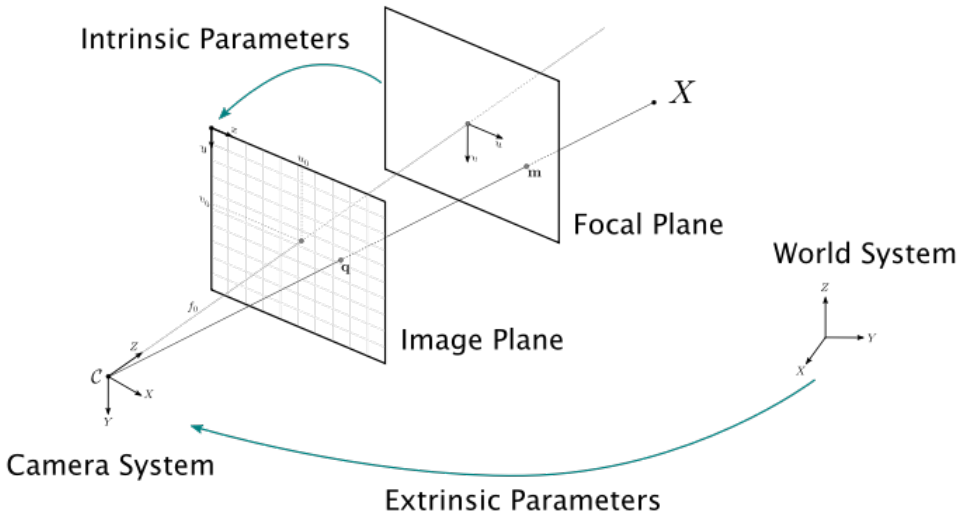

Figure 3.1: Schematic representation of pinhole camera and the transformation from pixel to world coordinates. From OpenMVG authors [7].

The transformation from pixel positions in the image (u,v) into real physical locations in the scene includes both a scaling and a translation of the 2D pixel coordinates based on the intrinsic camera parameters. The intrinsic camera parameters are the origin of the physical coordinate system (principal point (c_x, c_y)) and the focal lengths f_x and f_y. The relationship between the 2D and 3D coordinates in the camera reference frame is given by the equations of perspective projection (expressed in homogeneous coordinates):

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = K \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

in which the matrix $K \in \mathbb{R}^{3\times3}$ is known as the intrinsic matrix or camera calibration matrix.
Using OpenCV's functions, the distortion coefficients are found together with the intrinsic (or internal) camera parameters. This requires for each camera a set of calibration images using the same camera settings (image size, camera focus, ...) as during the measurement campaign. Each image should contain a checkerboard pattern of a predefined size. To obtain the best results, some recommendations are made with respect to this calibration procedure:

1. Calibration checkerboard pattern
   First of all, it is recommended that the width and height of the checkerboard pattern is not the same. Otherwise, robust detection of the pattern (and its orientation) in the image becomes more difficult for the OpenCV functions used during the calibration. Secondly, the size of the pattern should not be too small, too have sufficient amount of internal corners that serve as input, but also not too large. In the latter case, OpenCV sometimes fails in detecting the calibration pattern. As a suggestion, Engelen [5] uses is a pattern of 11 X 17 internal checkerboard corners. Finally, the physical size of the checkerboard-squares should be chosen so that the pattern is large enough depicted in the images to be distinguishable. This is dependent on the distance between camera and calibration pattern, for which we refer to recommendation 4 (Image quality).

2. Amount of images
   A sufficient amount of calibration images is required, in which the checkerboard pattern is viewed under different angles. In case too few images or images in which the camera position is not varied are used, a bad calibration result could be obtained which has a detrimental influence on the final accuracy of the water surface reconstruction. Because points are never detected perfectly, too much images on the other hand just add 'noise' to the calibration and therefore result in lower calibration quality. Based on scientific studies and practical experience, a training set of approximately 20 images is advised.

3. The location of the chessboard in the calibration pictures
   Enough corner points should be located close to the edges and corners of the image to get a better estimate of the distortion coefficients. Because lens distortion is most pronounced in the corners of an image, the correction of the pixel coordinates at the edges is more important than in the center of the image. A correct distortion model can however only be determined in case enough information about the distortion in those regions of the image is provided.

4. Image quality
   The quality of the calibration images should be as good as possible. The calibration images are taken with the camera parameters (focal length, aperture size, ...) that will be used during capturing of the reconstructed image sequence. In order to facilitate feature detection, the camera is focused to capture the feature pattern as sharp as possible. Scenes that are located at a significantly different distance to the camera will as such appear blurry for those specific camera settings. For this reason, the calibration images should be taken with a distance camera-calibration pattern approxametely equal to the distance camera-water surface during the measurement campaign. This allows to obtain sharp calibration images in which the chessboard corners can unambiguously be located.

To perform the calibration of a single camera, run the executable Calibration in the command line by the command:

```
$ ./Calibration
```

A line will be printed which asks the Settings-file. Therefore, adjust the calibration-settings-file (.xml file) that is also located in the Calibration folder to your specific project. The input variables are:

- BoardSize_Width, BoardSize_Height
  Size of the checkerboard pattern (number of rows and columns of internal corners of the pattern).

- Square_Size
  Square size (in mm's).

- nframes
  Number of images used for calibration (<= amount of images in InputDirectory).

- Showcalib
  Flag to indicate if calibration images need to be shown to user: 1 → yes, 0 → no.

- adaptImage
  Flag to indicate if calibration images need to be pre-processed to facilitate corner detection (perform brightness and contrast adjustments[1]): 1 → yes, 0 → no. In case 'yes', the α and β values are asked as input from the user.

- Write_outputFileName
  Name of the output file, e.g. *"Calib_CAM1.yml"*.

[1] Each output pixel's value depends on the corresponding input pixel value and multiplication parameter α and addition parameter β: $g(x) = \alpha f(x) + \beta$ with f and g the original and modified intensity value respectively.

- writePerview
  Flag to indicate if the average reprojection error for each calibration image need to be included in the output file: 1 → yes, 0 → no.
- Show_UndistortedImage
  Flag to indicate if undistorted calibration images need to be shown to the user: 1 → yes, 0 → no.
- InputDirectory
  Input directory containing calibration images: give full path or place the image folder in the same directory as the executable.

The result of the camera calibration is a .yml file, saved under the name that is given in the settings-file. This will contain the distortion coefficients, intrinsic camera parameters in camera matrix K and, if requested, the reprojection error for each image.

3.2 Initialization & set-up

Expressing the position of points (X,Y,Z) in a world coordinate system, independent of the camera reference system, also requires the extrinsic camera parameters. These allow a coordinate transformation between the camera's reference system and the world reference system. In such a way, we can relate the coordinate systems of the different camera's to a single reference frame in which the position of the water surface can be described. Therefore, the current implementation requires a regular pattern (e.g. checkerboard pattern) on the bottom of the tank for which the exact point locations are known and which is visible for every camera. By imposing the world coordinates of these corners (based on the number of rows/columns and square-size), a transformation from the different camera coordinate systems to a single, fixed Cartesian world reference system can be determined. The relationship between the points in the world coordinate system and the camera's coordinate system is given by:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \tag{3.1}$$

or

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + T \tag{3.2}$$

in which $R \in \mathbb{R}^{3\times3}$ is called the rotation matrix and $T \in \mathbb{R}^{3\times1}$ the translation vector. This 'camera pose estimation' is run automatically in the beginning of the surface reconstruction procedure for each camera used, based on images of which the names need to be passed by the settings-file. In case multiple experiments are performed using the same camera system and world reference system, the extrinsic camera parameters can also be read from a text-file that contains the matrix R and vector T of each camera.

The surface reconstruction program requires input parameters, i.e. a directory to read the images or coordinate text-files, and directories to store the results. More specifically, one should make sure that following folders are located in the same directory as the executable SurfaceReconstruction:

1. PRE_PROCESS
   A folder which holds the .yml files that contain the results from the camera calibration of the cameras used (see section 3.1). Additionally, a separate text-file needs to be present which holds the theoretical, fixed location of the feature points F on the bottom of the tank with respect to the chosen world reference frame.

2. DATA
   A folder which holds the images or text files containing the detected features (expressed in undistorted pixel coordinates) from a previous run (see 3. RESULTS; FEATURES). In case multiple cameras are used, the directory should contain sub-directories for the different cameras. The names of these images/files should be chosen in a chronological order because the program reads them from the last till the first alphabetically.

3. RESULTS
   A folder in which the results of the surface reconstruction will be stored. Depending in the settings-file (see next section), this will contain (a subset of) the following files at the end of the reconstruction procedure:

   - Folder *FEATURES*
     Directory with sub-directories CAM1, CAM2, ..., CAMi,... containing the detected features (in undistorted pixel coordinates) in the images of each camera for every time step. These can be used for a following surface reconstruction with for example a different surface model for the same experiment. Reading in those files is much faster than detecting features in the images so highly recommended in case multiple runs are performed based on the same images.
   - File *CameraPoseCam_i.txt*
     File containing the results of the camera pose estimation, i.e. the rotation matrix and translation vector of camera i.
   - File *Coefficients.txt*
     File containing the result of the surface reconstruction. Each line represents the coefficients of the surface model corresponding to the instantaneous surface shape for a single time step. Note that the reconstruction is started at the last image/file and continues till the first.
   - File *Errors.txt*
     File containing the resiudal error at the end of the optimization (normal collinearity metric or disparity difference metric), averaged over all feature points for every time step. In case the error is too large, it is possible to exclude the corresponding optimization result during post-processing.

Additionally, a .xml-file containing the settings of the optimization needs to be present in the directory of the executable. For more information about this settings-file, please read the next section.

3.3 Surface reconstruction

To start the global surface reconstruction, run the executable SurfaceReconstruction in the command line by the command:

```
$ ./SurfaceReconstruction
```

This will initiate the executable which will then read in the settings file settings.xml that needs to be present and adapted to your specific case. An overview of the different variables that need to adapted are given hereafter:

General settings

- ThreadAmount
  Amount of threads used in multicore or multi-CPU systems (must be smaller the amount of threads possible): > 1 accelerates the reconstruction but please keep in mind that at the moment the total amount of time frames must be a multiple of the amount of threads (otherwise some time frames will not be processed).

- showCorners
  Flag to indicate if for every time step the detected features (corners) need to be shown in the corresponding image: 1 → yes, 0 → no. This can be useful in case the reconstruction gives strange results, possibly due to incorrect feature localization.

- saveErrors
  Flag to indicate if error (averaged error metric over all features) corresponding to the optimized surface coefficients needs to be saved in text-file: 1 → yes, 0 → no.

- saveFeatureCoordinates
  Flag to indicate if undistorted feature locations (in pixel coordinates) need to be saved to text files: 1 → yes, 0 → no.

- saveCameraPose
  Flag to indicate if the result of the camera pose estimation (matrix R and vector T) need to be saved: 1 → yes, 0 → no. Only possible in case camera pose estimation is done based on an estimation from a reference image with calibration grid.

Input/output settings

- InputTypeRef
  Flag to indicate how the camera pose estimation will be performed: 1 → based on a reference image with calibration pattern, 0 → using a file containing previously found matrix R and vector T.

- InputType
  Flag to indicate how feature position will be determined: 1 → detection of features in images using corner detection algorithm, 0 → using file containing previously detected feature locations.

- InputInitial
  File name containing the fixed locations of the feature pattern.

- InputDirectory1
  The name of the input directory (images or text files) of camera 1

- InputDirectory2
  The name of the input directory (images or text files) of camera 2.

- InputDirectory3
  The name of the input directory (images or text files) of camera 3.

- InputReference1
  The name of the reference image/file used for extrinsic camera parameters of camera 1.

- InputReference2
  The name of the reference image/file used for extrinsic camera parameters of camera 2.

- InputReference3
  The name of the reference image/file used for extrinsic camera parameters of camera 3.

- CalibrationFile1
  The name of the .yml file containing the intrinsic camera parameters of camera 1 (see section 3.1).

- CalibrationFile2
  The name of the .yml file containing the intrinsic camera parameters of camera 2 (see section 3.1).

- CalibrationFile3
  The name of the .yml file containing the intrinsic camera parameters of camera 3 (see section 3.1).
- OutputCameraPose1
  The name of the output file containing the saved camera pose estimation of camera 1 if flag saveCameraPose is set on true.
- OutputCameraPose2
  The name of the output file containing the saved camera pose estimation of camera 2 if flag saveCameraPose is set on true.
- OutputCameraPose3
  The name of the output file containing the saved camera pose estimation of camera 3 if flag saveCameraPose is set on true.
- OutputDirectory1
  The name of output directory for saving feature coordinates of camera 1 in case flag saveFeatureCoordinates is set on true.
- OutputDirectory2
  The name of output directory for saving feature coordinates of camera 2 in case flag saveFeatureCoordinates is set on true.
- OutputDirectory3
  The name of output directory for saving feature coordinates of camera 3 in case flag saveFeatureCoordinates is set on true.

Position of cameras for camera pose estimation (see Appendix A)

- Position1
  Position of camera 1.
- Position2
  Position of camera 2.
- Position3
  Position of camera 3.

Characteristics of the chosen surface model

- SurfaceModel
  Amount of parameters used in the surface model. Currently models with a 1, 3, 5, 8 or 12 terms are possible (see Section 3.4).
- Square_Size
  The size of the camera-pose checkerboard squares (in mm).
- Lx
  Length scale in x-direction (crosswise direction) (in mm).

- Ly
  Length scale in y-direction (streamwise direction) (in mm).
- Initialguess
  Initial guess to start optimization procedure of coefficients for every thread. The amount of parameters passed in the initial guess does not has to match with the surface model that is chosen for the rest of the optimisation. This allows to pass a basic estimate, e.g. the average water level during the measurement campaign, which is then improved by the algorithm. In case at least 4 surfaces are already reconstructed for a specific thread, a best estimate is chosen from the four previous reconstructions as well as this initial guess. This best estimate is chosen as the set of surface coefficients that give the smallest global error for the updated set of image points. The initial guess can have an influence on the final result but most importantly influences the convergence rate of the optimization RE(with a very bad estimate convergence may never be reached).
- Scaling
  Scaling of the coefficients during optimization (length=SurfaceModel).[1] In case one term has no extremely large coefficients compared to the others, it is recommended not to use a variable scaling.

Detection of feature points settings

- BoardSize_Width, BoardSize_Height
  Number of rows and columns of the checkerboard pattern (internal corners) used for camera pose estimation.
- PatternSize_Width, PatternSize_Height
  Number of rows and columns of the feature pattern used for surface reconstruction.
- ResponseThreshold
  The threshold for response strength corners (see Appendix B).
- MinDistance
  The minimum distance in pixels between feature points.
- responseRadius
  The radius to calculate the corner response: 5 or 10 pixels are currently possible (see Appendix B).
- MatchesThreshold
  The maximum distance between the detected feature point (based on corner strength measure) and predicted feature point (based on prediction from its location in the previous frame) (in pixels).

Optimization settings

- CameraAmount
  Amount of camera's used.
- ErrorMetric
  Error metric used: 1 $\rightarrow$ Normal collinearity metric, 2 $\rightarrow$ Disparity difference metric.
- epsg
  Gradient based stopping condition for optimization.[1]

- epsf
  Function-value based stopping condition for optimization.[1]
- epsx
  Step-size based stopping condition for optimization.[1]
- maxits
  Maximum number of iterations during optimization.[1]
- diffStep
  Numerical differentiation step for gradient-calculation during optimization.[1]

An example of a settings file is given below:

```
<?xml version="1.0"?>
<opencv_storage>
<Settings>

  <!-- Amount of threads used-->
  <ThreadAmount>7</ThreadAmount>

  <!-- Show image with found corners or not-->
  <showCorners>0</showCorners>
  <!-- Save errors in text file-->
  <saveErrors>1</saveErrors>
  <!-- Save feature coordinates in text file-->
  <saveFeatureCoordinates>1</saveFeatureCoordinates>
  <!-- Save camera pose estimation in text file-->
  <saveCameraPose>1</saveCameraPose>

  <!-- Type of reference input: image->true:1, file->false:0-->
  <InputTypeRef>1</InputTypeRef>
  <!-- Type of input: images->true:1, file->false:0-->
  <InputType>1</InputType>
  <!-- Fixed location of points f-->
  <InputInitial>"/home/lengelen/Documents/PRE_PROCESS/FeaturesF.txt"</InputInitial>
  <!-- The name of the input directory of camera 1-->
  <InputDirectory1>"/home/lengelen/Documents/DATA"</InputDirectory1>
  <!-- The name of the input directory of camera 2-->
  <InputDirectory2>"/home/lengelen/Desktop/MS_20160524/m5/86/GRAY"</InputDirectory2>
  <!-- The name of the input directory of camera 3-->
  <InputDirectory3>"/home/lengelen/Desktop/Ms09052016/Processed/C84/M5"</InputDirectory3
      >
  <!-- The name of the reference image/file - camera 1-->
  <InputReference1>"/home/lengelen/Documents/Simulations/ref3.png"</InputReference1>
  <!-- The name of the reference image - camera 2-->
  <InputReference2>"/home/lengelen/Documents/Doctoraat/PhDMS1-adapted/Reference/REF15-
      small.png"</InputReference2>
  <!-- The name of the reference image - camera 3-->
  <InputReference3>"/home/lengelen/Desktop/Ms09052016/Processed/Ref84/Result of Ref84 (
      green)-10000.png"</InputReference3>

  <!-- The name of the calibration file of the camera 1-->
  <CalibrationFile1>"/home/lengelen/Documents/Doctoraat/Calib/Calib_AnSophie2.yml"</
      CalibrationFile1>
  <!-- The name of the calibration file of the camera 2-->
  <CalibrationFile2>"/home/lengelen/Documents/Calibration/Calib_C86_20160509-rotation.
      yml"</CalibrationFile2>
  <!-- The name of the calibration file of the camera 3-->
```

[1] For more information, we refer to the manual of the open source library ALGLIB from which we implemented the Levenberg-Marquardt algorithm.

```
<CalibrationFile3>"/home/lengelen/Documents/Calibration/Calib_C84_20160428.yml"</
    CalibrationFile3>

<!-- The name of ouput file for optimized coefficients according to chosen surface
    model-->
<OutputFileName>"/home/lengelen/Documents/RESULTS/coeffs.txt"</OutputFileName>
<!-- The name of ouput file for average error per frame-->
<OutputFileNameErrors>"/home/lengelen/Documents/RESULTS/errors.txt"</
    OutputFileNameErrors>

<!-- The name of ouput file for saving camera pose estimation camera 1-->
<OutputCameraPose1>"/home/lengelen/Documents/RESULTS/camerapose.txt"</
    OutputCameraPose1>
<!-- The name of ouput file for saving camera pose estimation camera 2-->
<OutputCameraPose2>"/home/lengelen/Desktop/test/CameraPose2.txt"</OutputCameraPose2>
<!-- The name of ouput file for saving camera pose estimation camera 3-->
<OutputCameraPose3>"/home/lengelen/Desktop/test/CameraPose3.txt"</OutputCameraPose3>

<!-- The name of ouput directory for saving feature coordinates camera 1-->
<OutputDirectory1>"/home/lengelen/Documents/RESULTS/FEATURES"</OutputDirectory1>
<!-- The name of ouput directory for saving feature coordinates camera 2-->
<OutputDirectory2>"/home/lengelen/Desktop/test/CameraPose2.txt"</OutputDirectory2>
<!-- The name of ouput directory for saving feature coordinates camera 3-->
<OutputDirectory3>"/home/lengelen/Desktop/test/CameraPose3.txt"</OutputDirectory3>

<!-- Position camera 1-->
<Position1>1</Position1>
<!-- Position camera 2-->
<Position2>4</Position2>
<!-- Position camera 3-->
<Position3>4</Position3>

<!-- Amount of parameters used-->
<SurfaceModel>5</SurfaceModel>
<!-- The size of a square for camera pose estimation in some user defined metric
    system (pixel, millimeter)-->
<Square_Size>15</Square_Size>
<!-- Length scale in x-direction (crosswise direction)(mm)-->
<Lx>40</Lx>
<!-- Length scale in y-direction (streamwise direction)(mm)-->
<Ly>160</Ly>
<!-- Initial guess to start optimization in each thread-->
<Initialguess>"[70.0,0.000,0.000,0.000,0.000]"</Initialguess>
<!-- Scaling of coefficients for optimization (length=SurfaceModel)-->
<Scaling>"[1.0,10.000,10.000,10.000,10.000]"</Scaling>

<!-- Number of inner corners per a item row and column. (square) -->
<BoardSize_Width>9</BoardSize_Width>
<BoardSize_Height>14</BoardSize_Height>
<PatternSize_Width>5</PatternSize_Width>
<PatternSize_Height>17</PatternSize_Height>
<!-- The threshold parameter for response strength corners-->
<ResponseThreshold>350</ResponseThreshold>
<!-- The min distance in pixels between corner points-->
<MinDistance>15</MinDistance>
<!-- The radius to calculate corner response: 5 or 10 currently possible -->
<responseRadius>5</responseRadius>
<!-- The max distance between located and predicted feature point -->
<MatchesThreshold>10</MatchesThreshold>

<!-- Amount of camera's-->
<CameraAmount>1</CameraAmount>
<!-- Error metric used: 1->normal collinearity metric, 2->disparity difference metric
    -->
<ErrorMetric>2</ErrorMetric>
```

```
    <!-- Gradient based stopping condition-->
    <epsg>0.000001</epsg>
    <!-- Function-value based stopping condition-->
    <epsf>0</epsf>
    <!-- Step size-based stopping condition-->
    <epsx>0</epsx>
    <!-- Maximum number of iterations during optimization-->
    <maxits>0</maxits>
    <!-- Numerical differentation step for calculation of gradient-->
    <diffStep>0.01</diffStep>

  </Settings>
  </opencv_storage>
```

3.4 Post-processing

The text file containing the optimized surface coefficients can finally be used to obtain a temporal description of the surface topography. Based on the Fourier theory regarding the decomposition of a periodic signal in a Fourier series, a surface model containing different cosine-terms is currently implemented in the program. Especially for the study of standing gravity waves, Lamb [6] proved that the water surface can be described by following model:

$$\eta(x,y) = \sum_m \sum_n A_{mn}\, cos(\frac{m\pi x}{L_x})\, cos(\frac{n\pi y}{L_y}), \quad m,n = 0,1,2,\ldots$$

Because most often only the first terms have a significant contribution to the surface topography, we implemented different surface models with a limited amount of terms. A linear term in both the x- and y-direction, $B\frac{x}{L_x}$ and $C\frac{y}{L_y}$, are also included to cope with a feature plane F that is not perfectly parallel with a still and flat water surface in the tank. Table 3.1 gives an overview of the possible surface models currently implemented.

Table 3.1: Overview of possible surface models

Model	**# parameters**	$\boldsymbol{\eta(x,y)}$
Constant height	1	A_{00}
Linear	3	$A_{00} + B\frac{x}{L_x} + C\frac{y}{L_y}$
First order	5	$A_{00} + B\frac{x}{L_x} + C\frac{y}{L_y} A_{10}\, cos(\frac{\pi x}{L_x}) + A_{01}\, cos(\frac{\pi y}{L_y})$
Second order	8	$A_{00} + B\frac{x}{L_x} + C\frac{y}{L_y} + A_{10}\, cos(\frac{\pi x}{L_x}) + A_{01}\, cos(\frac{\pi y}{L_y})$ $+A_{11}\, cos(\frac{\pi x}{L_x})\, cos(\frac{\pi y}{L_y}) + A_{20}\, cos(\frac{2\pi x}{L_x}) + A_{02}\, cos(\frac{2\pi y}{L_y})$
Third order	12	$A_{00} + B\frac{x}{L_x} + C\frac{y}{L_y} + A_{10}\, cos(\frac{\pi x}{L_x}) + A_{01}\, cos(\frac{\pi y}{L_y})$ $+A_{11}\, cos(\frac{\pi x}{L_x})\, cos(\frac{\pi y}{L_y}) + A_{20}\, cos(\frac{2\pi x}{L_x}) + A_{02}\, cos(\frac{2\pi y}{L_y})$ $+A_{21}\, cos(\frac{2\pi x}{L_x})\, cos(\frac{\pi y}{L_y}) + A_{12}\, cos(\frac{\pi x}{L_x})\, cos(\frac{2\pi y}{L_y})$ $+A_{30}\, cos(\frac{3\pi x}{L_x}) + A_{03}\, cos(\frac{3\pi y}{L_y})$

In case you want to visualize the surface evolution corresponding to the optimized coefficients, just insert the optimized coefficient in the chosen surface model. Then plot the function $\eta(x,y)$ in the domain (x,y) in which feature points were located, as the optimization result is only valid for those x- and y-values. To obtain spatial plots, you can write your own C++ function using e.g. gnuplot or use GUI possibilities embedded in Qt or GTK. Another solution is to load your data in more user-friendly data-processing software such as Matlab or R. An example in R is given by the file X.

Appendix A

Camera positions

Camera pose estimation requires that the detected pattern points in the reference image are matched with known 3D coordinates. To do this, the detected pattern is ordered such that the pattern point located in the top-left corner corresponds with the origin. The direction of the physical y- and x-axis are assumed to be approximately the same as the u- and v-axis in the image plane respectively. In case the cameras view the pattern from other corners with respect to the reference pattern, the top left pattern points in the different camera views do not correspond with the same physical point in the scene. Figure A.1 shows that the observed pattern is reversed for position 2 & 3 compared to position 1 & 4. This can automatically taken into account in the camera pose estimation by passing the camera positions, such that each camera is calibrated in the same reference system.

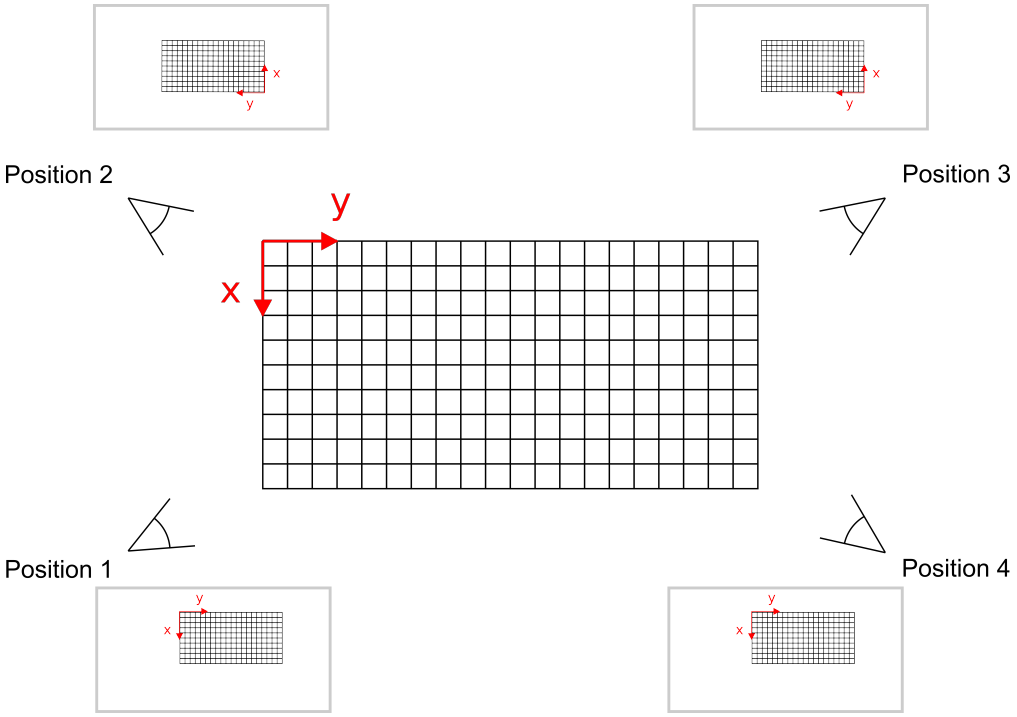

Figure A.1: Illustration of cameras around pattern with corresponding image used for camera pose estimation. The top-left pattern point in the images of camera 2 & 3 will not correspond with the origin of the physical reference system.

Appendix B

Corner detection algorithm

Corner detection is based on the algorithm developed by Bennett and Lasenby [2]. The algorithm is based on a corner strength which is calculated for every pixel in the image plane. Checkerboard corners are characterized by their 2×2 reciprocal black and white pattern. By comparison of the pixel values on a discretized sampling circle around the pixel of interest, a corner strength measure can be defined which gives a maximum value for the pixel at the center of such a 2×2 pattern. Bennett and Lasenby [2] proves that a robust but still computational efficient corner strength measure can be obtained by comparison of the pixel intensities (for a gray-scale image between 0 and 255) on a discretized sampling circle with given radius R. The corner strength measure is composed of two parts, namely the *sum response* and the *diff response*.

1. Sum response
On the vertex of a checkerboard pattern, pairs of points located on opposite sides of the sampling circle around the vertex should be of similar intensities. Points that are 90 out of phase on the circle should be of very different intensity. We denote I_n the image intensity of the n^{th} sampling point by proceeding on the sampling circle, starting at an arbitrary point with I_0. Around a vertex of the checkerboard pattern, the magnitude $(I_n + I_{n+8}) - (I_{n+4} + I_{n+12})$ should be very large (positive or negative). The *sum response* (SR) is then computed as the summation of these image pairs over all points (notice that we only have to loop from point 0 to 3 to compare all pixel pairs):

$$SR = \sum_{n=0}^{3} |I_n + I_{n+8}) - (I_{n+4} + I_{n+12})| \tag{B.1}$$

This quantity is large at a checkerboard vertex point, but additional calculations are necessary to avoid false positives which might occur at edges or due to image noise. Edges, especially in combination with image blur and image distortion, can have a significant *sum response* as well and are therefore difficult to distinguish from vertices's of the checkerboard pattern.

2. Diff response
Points located on a simple edge are characterized by very different intensities on opposite sides of the sampling circle. Therefore, the *diff response* (DR) should be large for edge points:

$$DR = \sum_{n=0}^{7} |I_n - I_{n+8})| \tag{B.2}$$

By subtracting the *diff response* from the *sum response*, the signal-to-noise ratio of the corner strength measure is significantly improved This results in the *total response* (TR):

$$TR = SR - DR \tag{B.3}$$

The implemented methodology is rotationally invariant and proved to locate checkerboard corners accurately and robust in case of optical blur and image noise. The radius of the sampling circle should however be fitted

to the size of the pattern within the image plane for which a compromise between two conflicting criteria needs to be made. The radius of the sampling should as small as possible in order to avoid aliasing on the other grid squares and allow the use of more dense patterns. On the other hand, the radius should be large enough to escape the central region in case of blurriness in the image. At the moment, two different 'response radius's' are possible, namely with 5 or 10 pixels. Choose the one best-fitted for your experimental set-up. In case both are insufficient, you can adjust the current response functions (located in ParticleTracking.cpp) or contact the OpenSRD development team.