Eidgenössische Technische Hochschule
Swiss Federal Institute of Technology
Zürich

Quick Guide to Multi-Camera Self-Calibration

Tomáš Svoboda
svoboda@vision.ee.ethz.ch
Computer Vision Lab, Swiss Federal Institute of Technology, Zürich

BiWi-TR-263, Version 0.2, August 20, 2003

We present a convenient calibration method for multiple cameras. Three cameras are the minimum but there is no upper limit. The method is fully automatic and a laser pointer is the only hardware used. A set of virtual 3D points is made by waving the laser pointer through the working volume. Its projections are found with sub-pixel precision and verified by a robust RANSAC analysis. The cameras do not have to see all the same set of points, only reasonable overlap between camera subgroups is assumed. Projection structures are computed via rank-4 factorization and the Euclidean stratification is done by imposing geometric constraints. This linear estimate initializes a post-processing computation of radial distortion which is also fully automated. We show that it is possible to calibrate an immersive virtual environment with 16 cameras within 30 minutes reaching about 1/4 pixel accuracy. The method has been tested on numerous multi-camera environments scaling quantity such as quality of the cameras. A short user's guide is part of the report, too.

Contents

1 Introduction

With decreasing prices of powerful computers and cameras, smart multi-camera systems start to emerge. A complete multi-camera calibration is the inevitable step towards the efficient use of such a system even though many things can be accomplished with uncalibrated cameras. To our best knowledge, no fully automatic calibration method for multi-camera environments exists.

Very recent multi-camera environments [13] or [5] which are primarily designed for a real-time 3D acquisition use advanced calibration methods based on a moving plane [20] and [1]. These calibration methods do not require a full 3D object with a known 3D coordinates. However, they share the main drawback with the old classical methods. The moving plane is not visible in all cameras and the partial calibrated structures have to be chained together which procedure is very prone to errors. Kitahara et al. [10] calibrate their large scale multi-camera environment by using a classical Tsai-like method [18]. The necessary 3D points are collected by a combined use of calibration board and 3D laser-surveying instrument. Lee et al. [11] establish a common coordinate frame for a sparse set of cameras that all observe a common dominant plane. They tracked objects moving in this plane and from their trajectories they estimated the external parameters of the cameras in one coordinate system. Baker and Aloimonos propose a calibration method for a multi-camera network which, however, still requires a planar pattern with a precise grid [3].

We propose a fully automatic calibration method which yields complete camera projection models and needs only a laser pointer[1]. At least approximately synchronized acquisition is assumed. The user is required to wave the laser pointer throughout the working volume. This is the only user action required. The laser projections are detected independently in each camera. We fit 2D Gaussian as a point spread function reaching sub-pixel precision. The points are validated through a pairwise epipolar constraints. Projective motion and shape are computed via rank–4 factorization. Geometric constraints are applied and projective structures are stratified to Euclidean ones. The parameters of the non-linear distortion are computed through iterative refinement. All these steps are described in this paper. The complete coupled calibration software yields about 1/3 pixel reprojection error even for cameras with significant radial distortion.

The paper reads as follows. Section 2 explains very shortly the necessary mathematical theory behind the algorithm which practical implementation is described in Section 3. Experiments on two different multi-camera environments are presented in Section 4. The paper closes with the concluding Section 5. Impatient users may jump directly to the Appendix and try to perform own self-calibration step by step. A description of the output parameters is also there.

[1]The small modification suggested later is extremely simple and costs essentially nothing.

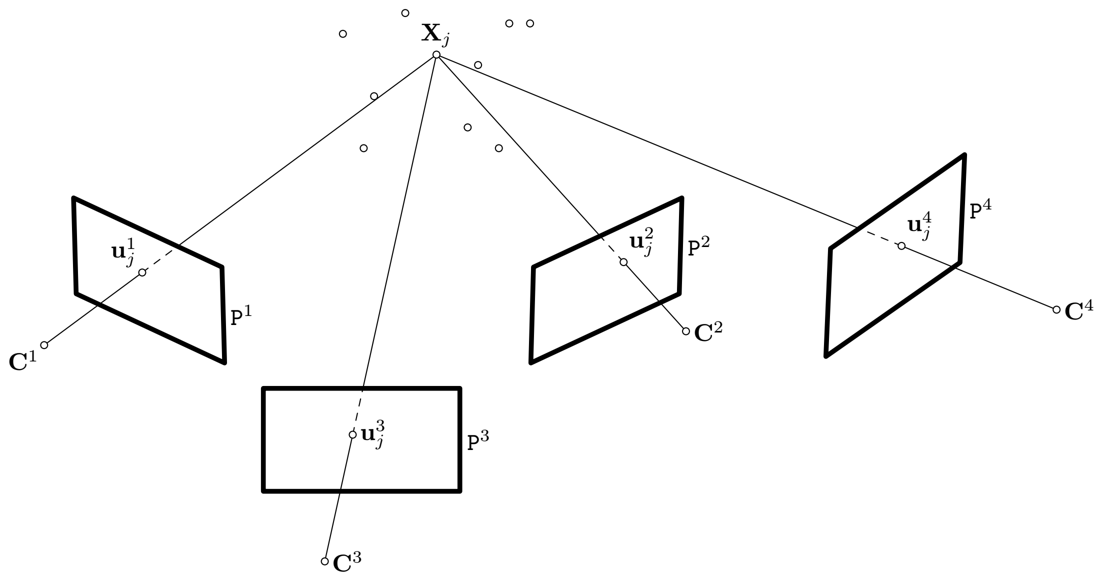

Figure 1: Multi-camera setup with 4 cameras.

2 Algorithm — Theory

Let us consider m cameras and n object points $\mathbf{X}_j = [X_j, Y_j, Z_j, 1]^\top, j = 1, \ldots, n$. We assume the pinhole camera model, see [9] for details. The 3D points $\mathbf{X}_j$ are projected to 2D image points $\mathbf{u}_j^i$ as

$$\lambda_j^i \begin{bmatrix} u_j^i \\ v_j^i \\ 1 \end{bmatrix} = \lambda_j^i \mathbf{u}_j^i = \mathtt{P}^i \mathbf{X}_j \,, \qquad \lambda_j^i \in \mathcal{R}^+ \tag{1}$$

where each $\mathtt{P}^i$ is a 3×4 matrix that contains 11 camera parameters, and u, v are pixel coordinates. There are six parameters that describe camera position and orientation, sometimes called external parameters, and five internal[2] parameters which describe the properties of the camera. The $\mathbf{u}_j^i$ are observed pixel coordinates. The goal of the calibration is to estimate scales λ_j^i and the camera projection matrices $\mathtt{P}^i$. We can put all points and camera projections (1) into one matrix $\mathtt{W}_s$:

$$\mathtt{W}_s = \begin{bmatrix} \lambda_1^1 \begin{bmatrix} u_1^1 \\ v_1^1 \\ 1 \end{bmatrix} & \cdots & \lambda_n^1 \begin{bmatrix} u_n^1 \\ v_n^1 \\ 1 \end{bmatrix} \\ \vdots & \vdots & \vdots \\ \lambda_1^m \begin{bmatrix} u_1^m \\ v_1^m \\ 1 \end{bmatrix} & \cdots & \lambda_n^m \begin{bmatrix} u_n^m \\ v_n^m \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \mathtt{P}^1 \\ \vdots \\ \mathtt{P}^m \end{bmatrix}_{3m \times 4} [\mathbf{X}_1 \cdots \mathbf{X}_n]_{4 \times n} \tag{2}$$

[2]Term *intrinsic* is also widely used.

$$\mathtt{W}_s = \mathtt{PX}\,, \tag{3}$$

where $\mathtt{W}_s$ is called the *scaled measurement matrix*, $\mathtt{P} = [\mathtt{P}^1 \cdots \mathtt{P}^m]^\top$ and $\mathtt{X} = [\mathbf{X}_1 \cdots \mathbf{X}_n]$. $\mathtt{P}$ and $\mathtt{X}$ are referred as the *projective motion* and the *projective shape* respectively. If we collect enough noiseless points (u^i_j, v^i_j) and the λ^i_j are known, then $\mathtt{W}_s$ has rank 4 and can be factorized into $\mathtt{P}$ and $\mathtt{X}$.

It should be noted that the only input we have is the collection of $\mathbf{u}^i_j$. Let assume that the point coordinates are organized in a matrix $\mathtt{W}$. This matrix is very similar to $\mathtt{W}_s$ in (2). However, the scales λ^i_j are not known in advance moreover, some of the points $\mathbf{u}^i_j$ may be missing because of occlusions or simply mis-detection. To compute the scales λ^i_j we applied the approach proposed in [14]. The missing points are filled in by applying rank-4 constraint, an approach suggested in [12].

The factorization of (3) recovers the motion and the shape up to a 4×4 projective transformation $\mathtt{H}$:

$$\mathtt{W}_s = \mathtt{PX} = \mathtt{PHH}^{-1}\mathtt{X} = \hat{\mathtt{P}}\hat{\mathtt{X}}\,, \tag{4}$$

where $\hat{\mathtt{P}} = \mathtt{PH}$ and $\hat{\mathtt{X}} = \mathtt{H}^{-1}\mathtt{X}$. Any non-singular 4×4 matrix may be inserted between $\mathtt{P}$ and $\mathtt{X}$ to get another compatible motion and shape pair $\hat{\mathtt{P}}, \hat{\mathtt{X}}$. The self-calibration process computes such a matrix $\mathtt{H}$ that $\hat{\mathtt{P}}$ and $\hat{\mathtt{X}}$ become Euclidean, sometimes is this process called *Euclidean stratification* [9]. The task of finding the appropriate $\mathtt{H}$ can be achieved by imposing certain geometrical constraints. The most general constraint is the assumption that rows and columns of camera chips are orthogonal. Alternatively, we can assume that some internal parameters of the cameras are the same, which is more useful for a monocular camera sequence. The minimal number of cameras for a successful self-calibration depends on the number of known camera parameters or the number of parameters that are unknown but same for more cameras. For instance, 8 cameras are needed when the orthogonality of rows and columns is the only constraint and three cameras are sufficient if all principal points are known or if the internal camera parameters are completely unknown but the same for all cameras [9]. The process described in more theoretical detail in [16].

We often require to remove nonlinear distortion if high precise 3D computation is requested. Our multi-camera setups offer more convenient (robust) configurations for 3D computations than classical stereo-based systems. On the other hand, we use lenses with short focal lengths which suffer from relatively significant distortion.

The principle is the following. First, reconstruct the calibration points by using the linear parameters and then feed theses 3D-2D correspondences into a standard method for estimation of the nonlinear distortion and repeat the self-calibration with the undistorted points. We decided to apply a part of the CalTech camera calibration toolbox [4]. Its Matlab codes are freely available and the estimated parameters are compatible the OpenCV library [1] which is useful for an eventual on-line distortion removal. The self-calibration is then repeated with the corrected point coordinates. This estimate and refine cycle is repeated until it reaches the required precision. This coupled iterative approach yields typically an average reprojection error less than 1/4 pixel.

3 Algorithm — Practical Implementation

In the previous section, we have argued that the data matrix `W` containing the image points is the only input we need for the calibration. This matrix may contain some missing points however, the more is this matrix full the more accurate and stable calibration results may be expected. Finding points $\mathbf{u}_j^i$ and establishing correspondences across many images is a difficult task. We solved the problem by waving a slightly modified laser pointer through the working volume, see Figure 2. The very bright projections of the laser can be detected in each image with sub-pixel precision by fitting an appropriate point spread function. These particular positions are then merged together over time, creating thus projections of a virtual 3D object. Our proposed self-calibration scheme can be outlined as follows:

1. Find the projections of the laser pointer in the images.
2. Discard wrongly detected points by pairwise RANSAC analysis [7].
3. Estimate projective depths λ_j^i and fill the missing points by the method described [12].
4. Optimize the projective structure by using the Bundle Adjustment [17], if applicable.
5. Perform the rank 4 factorization of the matrix W_s to get projective shape and motion [9].
6. Upgrade the projective structures to Euclidean by the method described in [16].
7. Detect remaining outliers by evaluating the 2D reprojection error. Remove them and repeat steps 3–6.
8. Estimate the parameters of the non-linear distortion and repeat the steps 2–7. Stop if the reprojection error is below the required threshold or if the maximum allowed number of iteration exceeds.
9. Optionally, if some true 3D information is known, align the computed Euclidean structures with a world system.

3.1 Finding corresponding points

We need a rather robust method for finding points since it is not always possible to make the working volume completely dark. The camera room may have windows and glossy surfaces making thus mis-detection probable. Any user interaction is not an option because of large number of images. However, it is assumed that the imaging conditions provide enough contrast between the laser pointer and the background. Our automatic finding procedure contains the following steps:

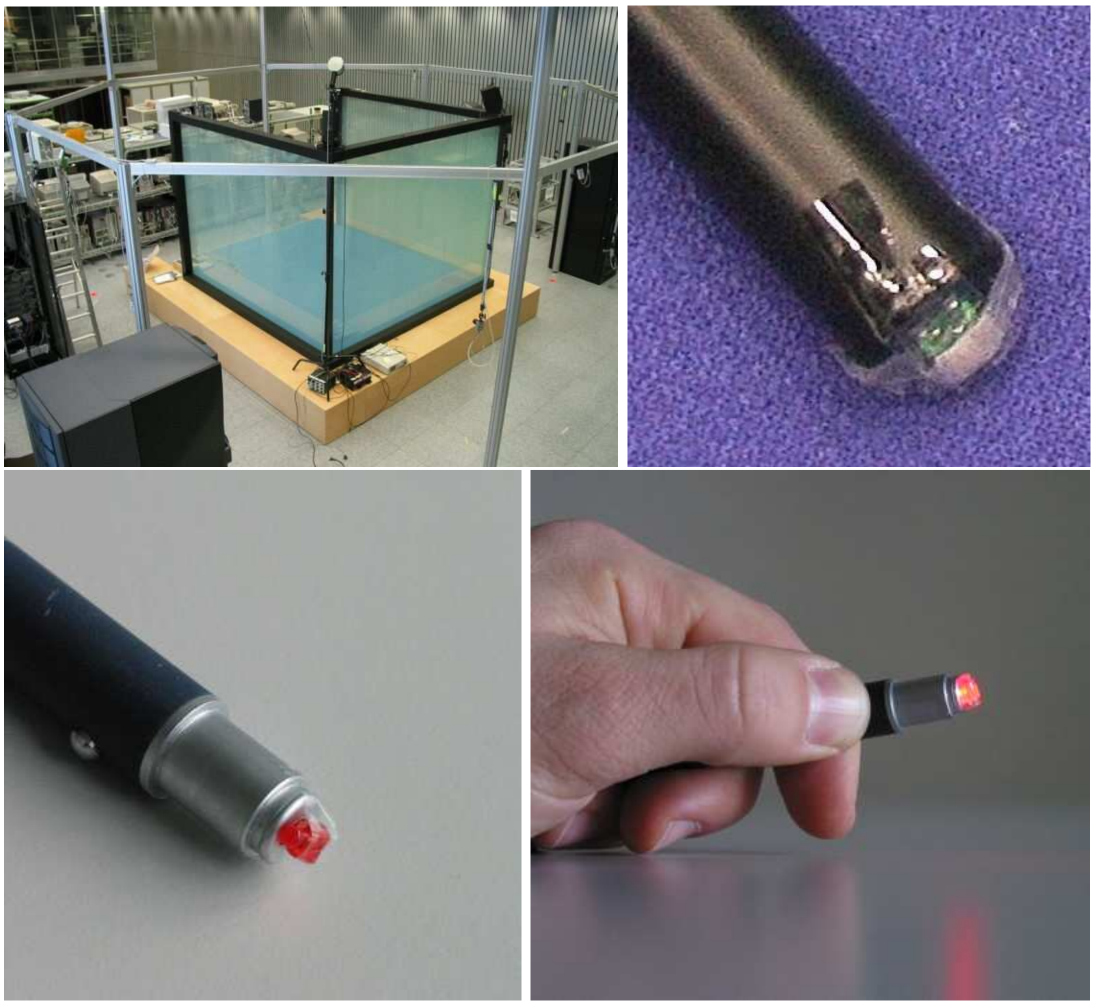

Figure 2: Immersive virtual environment BlueC [8] and the modification of a laser pointer. Small piece of transparent green or red plastic is attached to the laser pointer. The modification has been invented in order to get better visibility from different viewpoints. However primitive solution it is, it does the job very well. The working volume is inside the glass cave. Four cameras are mounted in the top four corners of the construction and the remaining 12 cameras are mounted on the aluminum scaffold that encompasses the cave.

1. The mean image and the image of standard deviation is computed for from all images each camera.

2. The difference image between the mean one and the actual one is computed by using the appropriate color channel. When using a green laser pointer, than the green channel is used. A threshold is set to 4/5 of the maximum of the difference image. The image is discarded if any of the following conditions holds:

   a) The number of pixels in the thresholded difference image is much higher than the expected LED size.
   b) The maximum of the difference image is less than five times standard deviation in this pixel.
   c) The thresholded pixels are not connected, i.e. they compose more than one blob.
   d) The eccentricity of the detected blob exceeds a predefined threshold. This condition is against motion blur.

3. The neighborhood of the detected blob is resampled to a higher resolution by using bicubic interpolation in order to reach sub-pixel accuracy and robustness against irregular blob shapes.

4. A 2D Gaussian is fitted to this interpolated sub-image by the 2D correlation to get the final position of the LED projection.

The detection sequence above is very robust and works well in different multi-camera environments. The color of the LED and the approximate expected size of the LED may vary for different setups. However, in practice they turn out to be extremely stable. The desired sub-pixel accuracy may be also specified however, 1/3 pixel should suffice for most cases. Some of the validation steps above may be skipped when the imaging environment is more controlled. The 2D correlation in step 4 is the most computationally expensive operation. The steps 2–4 take together about 100 ms for one 640×480 image with LED size 7 pixels and 1/3 sub-pixel accuracy on a 2 GHz machine. The localization of the LEDs may run highly in parallel, which is the case at BlueC implementation.

3.2 Discarding wrongly detected points

Even though the procedure described in the previous section is fairly robust some false points may penetrate. When some reflecting surfaces are present in the scene, e.g. glass walls, a reflection of the laser light might be detected instead of the direct projection. Such false points, called outliers, would spoil the projective reconstruction and have to be discarded. The discarding step is twofold. First step is pairwise computation of epipolar geometry and removing points that lie too far from epipolar lines. This step clears the data at the very beginning of the whole process. The second step is an iterative removing of outliers by analyzing 2D reprojection error.

3.2.1 Finding outliers in image pairs

The image pairs are iteratively re-selected according to the highest number of visible mutual correspondences. The points that were already detected as outliers are removed from the list however, only for these two images. The other projections of the same 3D point may be correct in other cameras. The epipolar geometry is robustly computed via RANSAC 7-point algorithm [9]. The initial tolerated distance from epipolar lines has to be pre-set by user. We use ten pixels which works well for all our datasets and this value is then iteratively decreased as the camera models become more and more precise.

3.2.2 Finding outliers in reprojected points

Some wrongly detected point may survive the validation described above. It may happen quite often that the point is found at a wrong position however, along the proper epipolar line. We assume that the point is wrongly detected in most of the cameras and it is thus highly probably discovered by the pairwise validation or it is in most of the cameras detected correctly. In the latter case, the point is almost correctly reconstructed in 3D space. However, when projected back to the "bad" camera it exhibits a large reprojection error.

3.3 Euclidean stratification

The stratification works rather well when reliable projective structures were estimated in the previous steps. We assume that cameras are different, have orthogonal rows and columns, no skew, square pixels, and we initialize the principal points to be in the image centers. It follows, from the counting argument [9], that we need at least three cameras to perform the self-calibration. The resulting Euclidean projective matrices are decomposed into the internal and external parameters. The initial assumption about zero skew and known principal points is not used in the final decomposition. Hence, the final internal parameters slightly deviate from the initial simplified ones.

The stratification without assuming known aspect ratios is generally less robust and may occasionally fail in case of somehow unbalanced data. Camera looking in the same direction close to each other is a typical example of ill conditioned scene for which the self-calibration may fail for noisy data. We did not have any camera with non-square pixels to really test it. However, this case is implemented, too.

3.4 Alignment with a world coordinate system

The self-calibration yields the external camera parameters in an unknown world coordinate frame with the origin in the centroid of the point cloud. In practical applications, it is often desirable to have all parameters in some well–founded coordinate frame. For cave environments for instance, we would like to have the $z = 0$ plane coincident with the cave floor. Several different approaches might be applied. Scene objects with known dimensions and positions might be localized in image(s) and used for the alignment. We utilize the knowledge of the camera approximate positions. Since we know the physical dimensions of the construction, we can approximate the positions of the camera centers without actually measuring them. The precision in range of several centimeters or even more is enough for reliable alignment. We need to know at least three cameras positions while having more will increase the robustness. The used cameras must not lie on one line. The similarity transformation is computed by using the algorithm [2].

Even approximate positions of the cameras are not always known. However, we can often assume generally planar movement of the user. A bird eye view of the overall arrangement would be useful. A plane is fitted to the reconstructed point cloud and then rotated to a desired orientation. The similarity transformation is then directly composed from this plane rotation.

3.5 Issues in estimation of the non-linear distortion

The iterative estimate and refine process is surprisingly stable assuming reasonable data. More and more parameters of the non-linear distortion are computed during the iteration. This process may occasionally fail for cameras which have weak coverage of the image plane or too many outliers. Disabling the automatic increasing the number of free parameters could stabilize the whole process. The point of the zero distortion is the most unstable parameter. Its estimation becomes unstable if the points are only on one side of the image. It is better to disable the estimation of this point and put it into the image center in case of such uncomplete data. The final reprojection error may remain rather high, say about one pixel. However, wrongly estimated non-linear parameters could destroy the overall geometric consistency. The filled 3D points are also used for the estimation. New data acquisition is suggested if nothing else helps. The number of iterations is by default constrained to 10. Actually, according to our experience the whole refinement should converge within 5-6 iterations. If not, than the desired model precision is set too optimistically with respect to the quality of the data.

3.6 Validation of an existing calibration

Sometimes we would like to know if the calibration is still valid or not. We may always re-calibrate the system completely. However, this takes some time, and the resulting parameters will not be exactly the same as the old ones even the setup remained the same. We suggest the following practical approach:

1. Capture about 100 frames whilst waving the laser pointer.
2. Find points.
3. Perform a robust Euclidean reconstruction by trying all combinations of camera n-tuples.
4. Select the camera n-tuple with the lowest reprojection error and its variance.
5. Evaluate the reprojection errors of this most consistent reconstruction.

The first two steps are the same as for the self-calibration itself. However, essentially less points are required. The complete validation may be thus completed in really few minutes.

4 Experiments

We would like to demonstrate two major properties in which our solution outperforms competitors:

- The laser point acting as a calibration device need not to be visible in all cameras simultaneously.
- The parameters of the non-linear distortion are estimated without any additional information.

The ability of filling missing points significantly broadens the possible use of our algorithm. Multiple cameras for immersive environments or telepresence virtual rooms often encompass the whole volume posing thus challenges in visibility. Our Blue-C [8] setups with 16 cameras each have almost no occlusion because of relatively empty working volume. However, the calibration point is visible in all cameras in only fraction of all calibration frames. Worse, points which are visible in all cameras span usually small part of the possible working volume making thus estimation unstable. Occlusions and very different, or even disjoint, fields of view are common problems when using our mobile version of out **Vi**Room [6, 15] system. Calibration based only on the points visible in all cameras would be virtually impossible here. The filled points take also part in the estimation of the non-linear distortion.

We will show that our automatic estimation of the non-linear lens parameters is able to compensate huge distortion of fish-eye lenses. This feature is necessary for very precise shape reconstruction applications.

We have used our algorithm on several multi-camera setups scaling both quality and quantity of the cameras, used. We start with Blue-C experiments. The Blue-C setups have 16 cameras each. Firewire cameras are synchronized by an external sync signal, each camera has its own computer running under Linux for acquisition. The calibration sequences have been acquired at 3–5 frames per second. Lower capturing frequency allow to fill the working volume without accumulation unnecessary high number of points. The speed of the waving is dictated by the shuttering time of the cameras. It is desirable not to move very fast to avoid motion blur. The lenses span from 2.8 mm to 12 mm exhibiting often considerable radial distortion. Both setups are used for high quality reconstructions which calls for very high precision of the camera models. We show that we are able to calibrate the setups achieving reprojection error about 1/5 pixels.

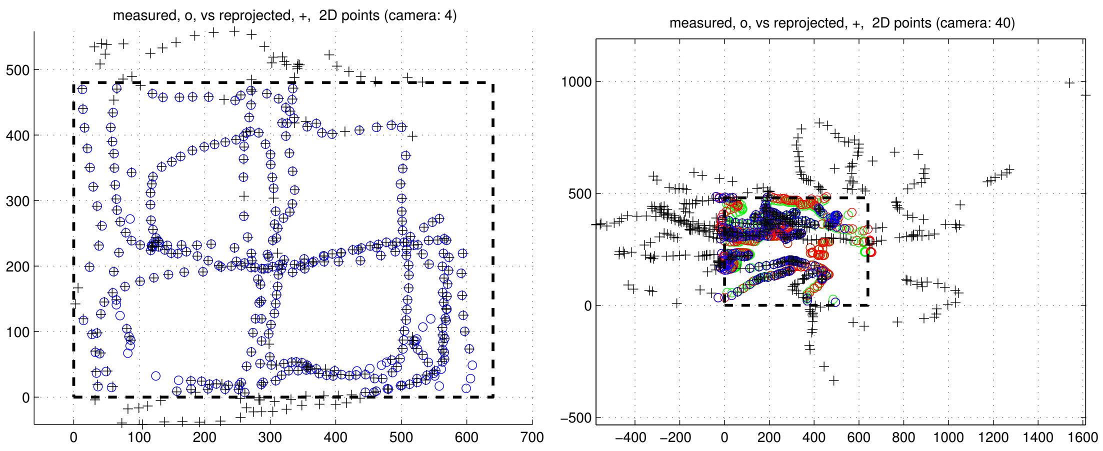

Figure 3: Filling of the points not visible in the cameras shown but in the others. On the left, a Blue-C camera mounted on the ceilings. On the right, a camera from a ViRoom installation. The camera have limited fields of view and do not see the whole working volume. The filling feature is clearly observable. Some points which have been reconstructed in 3D are clearly outside the image sensor (dashed line). They are visible in other cameras and filled into the measurement matrix.

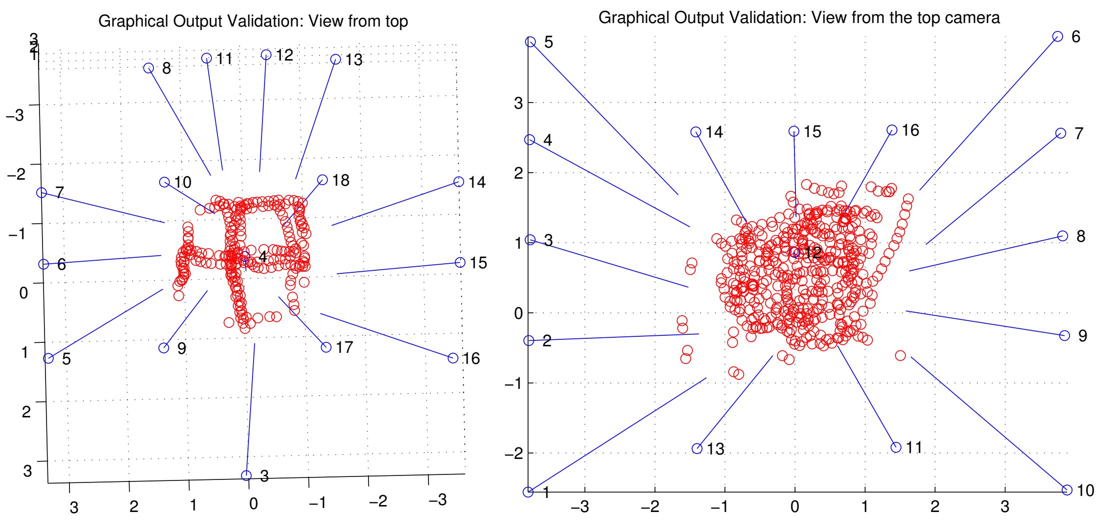

Figure 4: Results of the Euclidean stratification for the Blue-C 16-cameras setups. Small blue circles denote position of the camera centers, blue lines the orientation of the optical axes. The red circles show the reconstructed positions of the laser pointer.

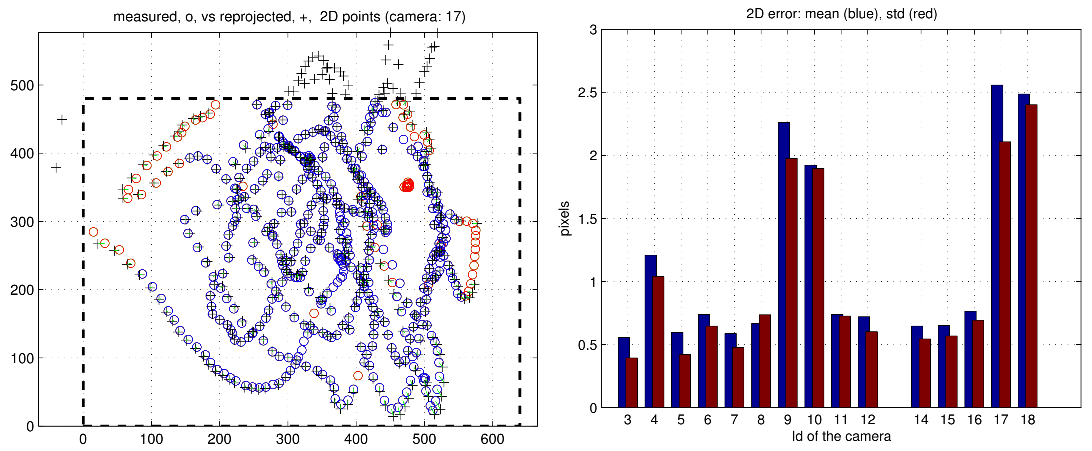

Figure 5: Results of the Linear model estimation. The left figure shows the point reprojections in one of the cameras with significant radial distortion. The small circles denote the detected points, red ones are tentative outliers which were detected in the pairwise RANSAC validation. The crosses are back-projected reconstructed calibration points. Right figure shows average reprojection errors and standard deviations in each camera. You can clearly distinguish the cameras No. 9,10,17,18 which are mounted inside the cave and have the shortest lenses and thus significant distortion.

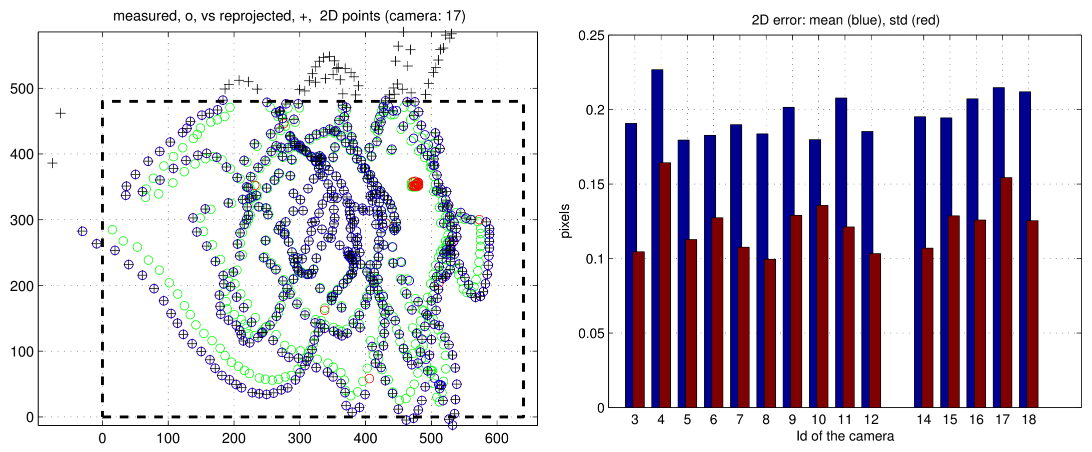

Figure 6: Complete projection model. The left figure shows the same camera as the left figure in Fig 4. The green circles are the originally detected points, the blue ones show points after undoing radial distortion. The right graph illustrates well–balanced reprojection error around 1/5 pixels. Compare with the reprojection of the linear model in Fig 4.

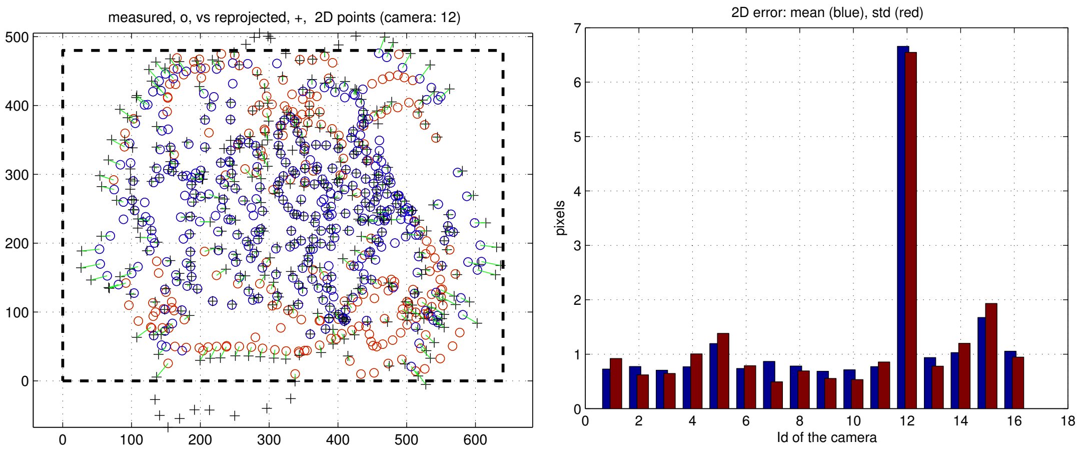

Figure 7: Results of the Linear model estimation. Example of an unbalanced multi-camera system. The camera 12 has a fish-eye lens with huge radial distortion.

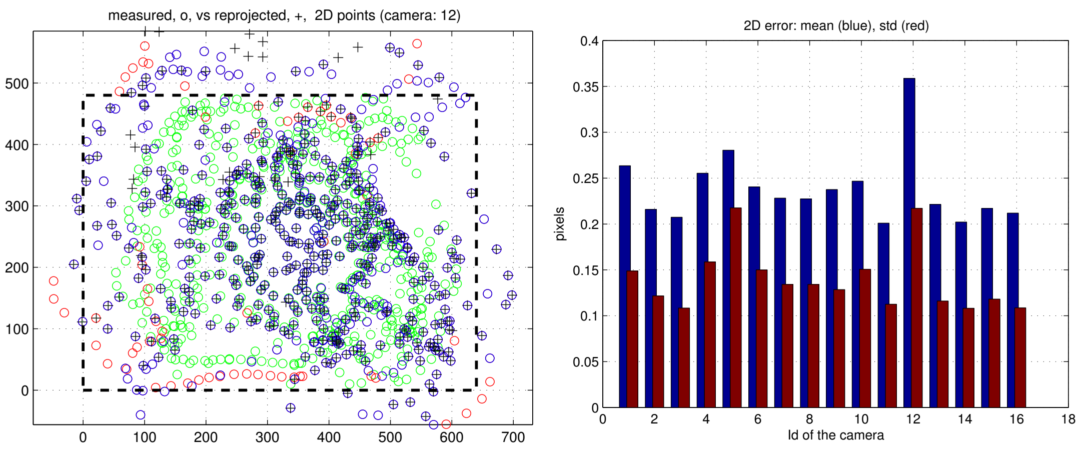

Figure 8: Results of the complete model estimation. The camera 12 still has higher reprojection error than the others. However, from the initial error about 7 pixels it decreased to less than 0.4 pixels. The extreme radial distortion can be clearly recognized in considerably different positions of the green (original points) and blue/red circles (undistorted points).

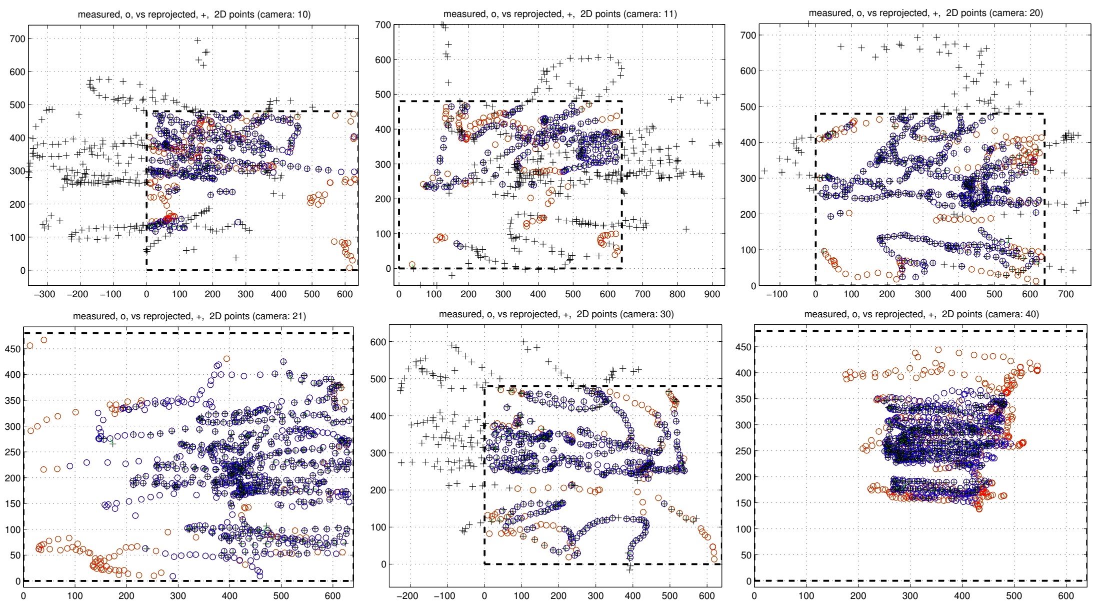

Figure 9: Example of a less controlled multi-camera setup. Note significant differences in camera fields of view. The filled points go essentially outside the image planes (denoted by dashed rectangle). The last camera (bottom-right) is quite far from the others and the points are clustered around the image center only. The first camera (top-left) has very unbalanced spread of points. Note also the considerable number of outliers caused by a very difficult imaging conditions. The cameras are synchronized based on TCP/IP communication only. Nevertheless, the 6-camera setup is calibrated reliable with less than two pixels reprojection error.

The **Vi**Room setups, both mobile one and static one pose slightly different challenges. The sub-pixel accuracy is not strictly required, shape 3D reconstruction is not the main application here. The setups are used for multi-camera tracking, activity monitoring, telepresence applications. The mobile version with six cameras and two laptops has been successfully used in a real factory environment. Both setups contain varying number of simple firewire cameras without external synchronization. One computer, standard PC or a laptop running on Linux, has often to serve more than just one camera. The acquisition is controlled via TCP/IP communication [6]. The working volume often contains furniture other computers and it cannot be completely darkened. Even worse, the camera fields of view frequently overlap only marginally. Still, our system is able to calibrate such setups with sufficient precision. The estimation of the non-linear distortion is difficult in such environments and may fail. It is mostly necessary to fix the center of the non-linear distortion which is quite brittle parameter for unbalanced spread of image points. The limited precision of the synchronization also plays a role. Simply speaking, you cannot get better precision of the calibration than your points have.

5 Conclusion

A reliable scheme for a complete and fully automatic calibration of a multi-camera network has been presented. A laser pointer or any similar bright spot object is the only required additional hardware. Waving this object through the working volume is the only hand work required. The object needs not to be visible in all cameras. The non-linear distortions are estimated from the same data set.

Experiments with different multi-camera setups scaling quality and quantity demonstrated the broad usability of our algorithm.

Acknowledgments

We thank Tomáš Pajdla and Daniel Martinec for the filling points procedures, for Ondřej Chum for providing RANSAC implementation, Tomáš Werner for his Bundle Adjustment routines, and Jean-Yves Bouguet for the non-linear distortion codes. Student Dejan Radovic implemented a very first version of the stratification.

References

[1] Open source computer vision library. http://www.intel.com/research/mrl/research/opencv/. Last visited 25 June 2003.

[2] K.S. Arun, T.S. Huang, and S.D. Blostein. Least-squares fitting of two 3-D point sets. *IEEE Transaction on Pattern Recognition and Machine Intelligence*, 9(5):698–700, September 1987.

[3] Patrick T. Baker and Yiannis Aloimonos. Calibration of a multicamera network. In Robert Pless, Jose Santos-Victor, and Yasushi Yagi, editors, *Omnivis 2003: Omnidirectional Vision and Camera Networks*, 2003.

[4] Jean-Yves Bouguet. Camera calibration toolbox for matlab. http://www.vision.caltech.edu/bouguetj/calib_doc/. Last visited June 17, 2003.

[5] German K.M. Cheung, Simon Baker, and Takeo Kanade. Shape-from-silhouette of articulated objects and its use for human body kinematics estimation and motion capture. In *Computer Vision and Pattern Recognition*, 2003.

[6] Petr Doubek, Tomáš Svoboda, and Luc Van Gool. Monkeys — a software architecture for ViRoom — low-cost multicamera system. In James L. Crowley, Justus H. Piater, Markus Vincze, and Lucas Paletta, editors, *3rd International Conference on Computer Vision Systems*, number 2626 in LNCS, pages 386–395. Springer, April 2003.

[7] M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, June 1981.

[8] M. Gross, S. Wuermlin, M. Naef, E. Lamboray, Ch. Spagno, Kunz. A., E. Koller-Meier, T. Svoboda, L. Van Gool, S. Lang, Strehlke K., A. Vande Moere, and O. Staadt. Blue-c: A spatially immersive display and 3D video portal for telepresence. In *Proceedings of ACM SIGGRAPH 2003*, July 2003.

[9] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, Cambridge, UK, 2000.

[10] I. Kitahara, H. Saito, S. Akimichi, T. Onno, Y. Ohta, and T. Kanade. Large–scale virtualized reality. In *Computer Vision and Pattern Recognition, Technical Sketches*, June 2001.

[11] L. Lee, R. Romano, and G. Stein. Monitoring activities from multiple video streams: establishing a common coordinate frame. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):758–767, August 2000.

[12] Daniel Martinec and Tomáš Pajdla. Structure from many perspective images with occlusions. In A. Heyden, G. Sparr, M. Nielsen, and P. Johansen, editors, *Proceedings of the European Conference on Computer Vision*, volume II, pages 355–369, Berlin, Germany, May 2002. Springer-Verlag.

[13] Simon Prince, Adrian David Cheok, Farzam Farbiz, Todd Williamson, Nik Johnson, Mark Billinghurst, and Hirokazu Kato. 3D live: Real time captured content for mixed reality. In *International Symposium on Mixed and Augmented Reality (ISMAR'02)*, pages 7–13. IEEE Press, September-October 2002.

[14] Peter Sturm and Bill Triggs. A factorization based algorithm for multi-image projective structure and motion. In *European Conference on Computer Vision*, pages 709–720. Springer - Verlag, 1996.

[15] Tomáš Svoboda, Hanspeter Hug, and Luc Van Gool. **ViRoom** — low cost synchronized multicamera system and its self-calibration. In Luc Van Gool, editor, *Pattern Recognition, 24th DAGM Symposium*, number 2449 in LNCS, pages 515–522. Springer, September 2002.

[16] Tomáš Svoboda, Daniel Martinec, and Tomáš Pajdla. A convenient multi-camera self-calibration for virtual environments. *Presence*, 2004. Submitted.

[17] Bill Triggs, Philip McLauchlan, Richard Hartley, and Andrew Fitzgibbon. Bundle adjustment – A modern synthesis. In W. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms: Theory and Practice*, LNCS, pages 298–375. Springer Verlag, 1999.

[18] Roger Y. Tsai. A versatile camera calibration technique for high-accurancy 3D machine vision metrology using off-the-shelf cameras and lenses. *IEEE Journal of Robotics and Automation*, RA-3(4):323 – 344, August 1987.

[19] Stephan Würmlin, Edouard Lamboray, and Markus Gross. 3D video fragments: Dynamic point samples for real-time free-viewpoint video. Technical Report CS #397, Computer Graphics Lab, Swiss Federal Institute of Technology, April 2003.

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

6 Appendix: Multi-camera self-calibration made easy

The following step-by-step procedure should work for most of the setups. However, if something goes wrong it may be necessary to consult the previous part of the report. A working multi-camera system with at least three cameras is assumed as well as a calibration laser pointer. During the time, the software package might get upgraded faster than the text hence, inconsistencies may appear. The more detailed but unorganized explanation may be found in `./MultiCamSelfCal/README.txt` file.

1. Arrange the cameras, fix them as much as possible.
2. Switch off the lights, draw the curtains if available. Ideally, the room should be relatively dark. If not possible, modify the camera parameters. If you have a strong laser pointer you may decrease the gain of cameras, close the iris, or shorten the shuttering time.
3. Capture a sequence whilst waving the pointer through the working volume. Try to fill the working volume uniformly.
4. Collect the captured images and check the config variables in the `./CommonCfgAndIO/configdata.m`: `config.paths.*, config.files.*, config.imgs.*`.
5. Start the `./MultiCamSelfCal/FindingPoints/im2points` script. The most frequent failure is the incorrect setting of config variables.
6. Check the `config.cal.*` variables and start `./MultiCamSelfCal/gocal`. Wait for the results. The `gocal` process may lead to bad results or even crash from several reasons. We will try summarize them later.

6.1 Failure of the `gocal`, what to do

We try to summarize possible reasons which may lead to a failure of `gocal` process and suggest eventual solution. Some failures certainly remain undiscovered, if you find one please let me know.

Inconsistent input data. The process then typically crashes at the very beginning complaining about uncomplete, not found or simply inconsistent data. → Check the config variables, access right to the files. Whatever it may sound strange, this is the most frequent reason for a failure. This error is especially likely if you use your own program for finding calibration points.

Ransac validation fails. The input matrices may have valid dimensions however, they still may be inconsistent. → The typical reason is inconsistent acquisition and(or) arrangement of images. The frames may not be consistently synchronized which causes huge errors in computations. The Ransac validation contains currently a c-code and it is not that stable. Try run `gocal` again. The RANSAC is stochastic, it may help.

Filling points fails. → The data matrix W_s is too sparse. Too few points survived the RANSAC validation. Try to increase `config.cal.INL_TOL` and(or) increase `config.cal.NUM_CAMS_FILL`. Visualize the point structure by `imshow(loaded.IdMat)`, black means point not detected. If the matrix is too black, something is wrong. → Run `showpoints` script which visualizes the detected points in images. This visualization may often discover some unexpected problems in point detection.

Reprojection error extremely high. The final validation step is iterating and the reprojection error is still extremely high in range of hundreds or even thousands. → The input data is highly probably frame-inconsistent. It means, frames captured at different time instants are considered as synchronized.

Reprojection error very high in only few cameras. Mostly because of two reasons. The camera(s) is(are) out of sync, or something weird is apparent in its(their) field(s) of view. → Disable this camera(s) by setting `config.cal.cams2use` variable and run the `gocal` again. Run the `showpoints` script to detect possible problems. If you have many cameras, it is rather probable that some of them may have some hardware/software problem. The `config.cal.cams2use` variable turned out to be very useful for tracking down problems.

Estimation of the non-linear parameters diverges. The final refinement through estimation of the non-linear parameters reaches the maximum number of allowed iterations without success or even making the reprojection errors worse. → Check the Matlab figures with the camera points. Are the points well spread, ideally uniformly, over the whole image planes? Try to disable some parameters by setting the `config.cal.NL_UPDATE` variable.

Euclidean parameters somehow suspicious. The reprojection error is relatively low, say below three pixels, but the graphical output validation presents somehow suspicious result. You would expect something like on Figure 4 however, it is not. Please note that the result may be still valid. The results on Figure 4 are already aligned with a well-founded world coordinate system. You can find some example of alignments functions in the `./MultiCamSelfCal/LocalAlignments` directory. Check the files `planarmove.m, planarcams.m, erlangen.m`. You may also try to rotate the 3D plot interactively to reach some orientation that is similar to the real arrangement of your cameras. Still, the results may remain really suspicious, like chaotic orientation of cameras, placement inside the point cloud, etc. → Check the spread of points in the image planes. The Euclidean stratification may become unstable if the points occupy only minor part of the image. Try to disable the assumption of square pixels by zeroing `config.cal.SQUARE_PIX`. If it does not help, force the bundle adjustment to be performed in each intermediate step by setting the `config.cal.START_BA` variable to 1.

6.2 Outputs and auxiliary files

The format of the output parameters has been mainly formed by the Blue-C shape reconstruction algorithm [19]. However, a general purpose form is also available. All the output files will be saved to the `config.paths.data` directory specified in the `configdata.m`. Let M denote number of cameras and N the number of frames captured. In the above mentioned directory, you can find the following files:

`*.tiff` **images** Output from the `im2points` procedure. Mean images and images of the standard deviations.

`*.dat` ASCII data files.

- `Res.dat` $M \times 2$ matrix containing the image resolutions. Product of `im2points.m`.
- `IdMat.dat` $M \times N$ boolean matrix. 1 or 0 at (i, j) position means point detected, resp. not, in the $i-$th camera and $j-$th frame. Product of `im2points.m`.
- `points.dat` $3M \times N$ matrix containing the $\mathbf{u}_j^i$ points. `NaN` (Non A Number) is used if the point is not detected. Product of `im2points.m`.
- `Cst.dat` $M \times 3$ matrix with the coordinates of the camera centers. Product of `savecalpar.m`.
- `Pmatrices.dat` $3M \times 4$ collection of the Euclidean projection matrices $\mathtt{P}^i$. Product of `gocal.m`. These Pmatrices are the raw output of the stratification. They are not aligned with the predefined world coordinate system.
- `Pst.dat` $3M \times 3$ collection of special matrices used in [19] for the back-projection. Product of `savecalpar.m`.
- `cam%d.points4cal.dat` contains 3D-2D correspondences used for the estimation of the non-linear distortion. Product of `gocal.m`.

`*.cal` Calibration parameters.

- `%s%d.cal` where `%s` is to be replaced by `config.files.basename`. It contains coordinates of the camera center and back-projection matrix. These files are directly used in [19]. Product of `savecalpar.m`.
- `camera%d.Pmat.cal` contains the $3 \times 4\mathtt{P}^i$ matrix. These matrices could be decomposed to the particular calibration matrix by using the function `./MultiCamValidation/CoreFunctions/P2KRtC.m`.
  Product of `savecalpar.m`.

`%s%d.rad` contains parameters of the non-linear distortion. Directly used in [19]. Product of `./CalTechCal/goradf.m`.

`*.eps` are various graphical outputs, mostly products of `gocal.m`.