Report

Towards a continuous, unified calibration of projectors and cameras

Author(s):
Cotting, Daniel; Nebel, I.; Gross, Markus H.; Fuchs, Henry

Publication Date:
2011

Permanent Link:
https://doi.org/10.3929/ethz-a-006787314

Rights / License:
In Copyright - Non-Commercial Use Permitted

This page was generated automatically upon download from the ETH Zurich Research Collection. For more information please consult the Terms of use.
Towards a Continuous, Unified Calibration of Projectors and Cameras

D. Cotting\textsuperscript{a}, I. Nebel\textsuperscript{a}, M. Gross\textsuperscript{a} and H. Fuchs\textsuperscript{b}

Technical Report 563, ETH Zurich

\textsuperscript{a}Computer Graphics Laboratory, ETH Zurich, Switzerland
\textsuperscript{b}Department of Computer Science, UNC-Chapel Hill, USA

Abstract

This article presents a novel approach for a unified calibration of both cameras and projectors. In addition to providing a solution for static settings, we introduce a continuous recalibration procedure allowing dynamic modifications of the extrinsic parameters during runtime. Generation of point correspondences for calibration can be performed concurrently with the main application of the system in a non-disturbing way. This can be achieved by using an imperceptible structured light approach enabling per-pixel surface light control. We aim at providing ready-to-use recipes for practitioners that have to use cameras and projectors in computer graphics applications, and share practical experiences and implementation-related details. Based on the information given in this article, a projector-camera system should be easy to install, should handle all components in a unified way and allow continuous dynamic setup changes.

Keywords: Projector, camera, calibration, homography, imperceptible structured light.

1. Introduction

With both cameras and projectors becoming affordable not only for research, but also for large-scale deployment, more and more systems are being developed, which capture the environment and at the same time augment it with projections. To register the devices to the same coordinate system, they have to be calibrated, i.e. their projection parameters have to be determined. While calibration of a single camera is regarded as a solved problem, no satisfactory solution for the calibration of multiple cameras and projectors has been presented yet. In this article, we introduce a unified calibration approach for both cameras and projectors. By first introducing a trivial calibration method, we illustrate the drawbacks of traditional approaches and define the design requirements for our novel unified handling of cameras and projectors. Dynamic changes will continuously be accounted for by a recalibration procedure based on imperceptible per-pixel light control.

2. Related work

Both image capture and projection is generally described by a standard pinhole camera model. Thus, projectors can be seen as duals of cameras, and projection alignment is naturally related to camera calibration. In wide-area environments, individual cameras often observe only a small fraction of the entire scene, making determination of reliable global relationships very difficult. Approaches based on self-calibration \cite{1} resolve this problem by observing a number of features in a set of successive images \cite{2, 3, 4}.

Since wide field-of-view cameras and projectors suffer significant radial distortion, non-linearity in the imaging process poses an additional challenge. Although radial distortion is not the only lens aberration altering projected images, it is certainly the most significant one in practice. It is often handled in a preprocessing step using traditional calibration methods \cite{5, 6}. More advanced self-calibration schemes perform estimation of distortion in iterative procedures, either minimizing error functions \cite{7, 8}, or solving directly for the parameters \cite{9, 10}. In the literature, several techniques are reported, which solve radial distortion separately without performing complete calibration calculations \cite{11, 12, 13, 14, 15}.

For appealing projection-based displays and augmented environments, a large variety of problems have to be addressed.\footnote{An extensive bibliography of projector-related publications can be found at http://www.cs.unc.edu/~raskar/Projector/projbib.html.} Creating high-quality images does not only require geometric alignment, but also smooth blending of multiple units and thorough color adjustment \cite{16, 17, 18}. Up to now, solutions for projector alignment have
mainly been devised for limited, application-specific cases, restricting the display geometry to planar, multi-planar or curved surfaces [19, 20, 21, 22]. More general results considering projections to arbitrary surfaces have been created by Raskar [23, 24, 25]. While most methods treat the alignment task as a one-time initialization step, Yang and Welch [26] have devised a procedure continuously updating parameters during runtime. Estimates of the display surface are iteratively refined based on correlation of a feedback camera and the projected imagery. Similarly, our newly developed approaches will allow for dynamic changes of arbitrary display surfaces, but will not rely on time-consuming correlation calculations.

Adaptivity of camera calibration to setup changes has been presented by Zhang and Schenk [27] in the context of robotic systems. The system is initially calibrated with a classical method using a calibration object. Then, at each time instant, the camera projection matrices are adjusted by minimizing the error between projections of 3D points reconstructed in the previous time step and the matching image points from the current time step. The method assumes a fixed scene structure as a source for the reference points. So it is not directly applicable to our goal. Furthermore, the approach does not consider radial distortion, which is often affecting wide field-of-view cameras. Shen and Menq [28] propose a dedicated projector calibration procedure and support projector location changes through rigid-body transformations, while camera positions are restricted to static positions. Similarly, Chen and Li [29] present an approach for self-recalibration confined to a single projector-camera pair.

In order to eliminate the drawbacks raised by the aforementioned methods, we will rely on a twofold approach. First, we extend a self-calibration procedure by Barreto et al. [10] to achieve initial calibration of a system consisting of multiple projectors and cameras. We preserve the option of simultaneous handling of radial distortion, intrinsic and extrinsic calibration in a reliable way. Second, dynamic changes are continuously accounted for with an efficient rigid-body transformation approach. While developing the final calibration approach, intermediate solutions and their limitations will be presented.

3. Basics

This section introduces the basic theoretical knowledge, the mathematical foundations and computational techniques required for our calibration approach.

3.1. Camera model

When considering the pinhole camera model, a point in space $X = (X, Y, Z)^T$ is mapped to a point on the image plane $Z = f$ by intersecting the plane with a line joining the point $X$ to the center of projection, which is assumed at the origin of the coordinate system. This results in a mapping from euclidean 3-space $\mathbb{R}^3$ to euclidean 2-space $\mathbb{R}^2$:

$$ (X, Y, Z)^T \rightarrow (f \cdot X/Z, f \cdot Y/Z)^T. \quad (1) $$

**Principal point.** The principal point is defined as the intersection of the image plane with the principal axis of the camera, i.e. the line through the center of projection perpendicular to the image plane. The previous equation assumes that the origin of coordinates in the image plane is at the principal point. For the more general case, the central projection can be expressed as a linear mapping in homogeneous coordinates:

$$
\begin{bmatrix}
X \\
Y \\
Z \\
1
\end{bmatrix}
\rightarrow
\begin{bmatrix}
f \cdot X + Z \cdot p_x \\
f \cdot Y + Z \cdot p_y \\
Z \\
1
\end{bmatrix}
= \begin{bmatrix}
f & 0 & p_x \\
0 & f & p_y \\
0 & 0 & 1
\end{bmatrix}
\cdot
\begin{bmatrix}
X \\
Y \\
Z \\
1
\end{bmatrix},
\quad (2)
$$

where $p = (p_x, p_y)^T$ are the coordinates of the principal point.

**Coordinate systems.** Since points in space are usually not directly expressed in the camera coordinate frame but in a world coordinate frame, the two coordinate frames have to be related via a rotation and a translation. We get the following mapping from world to camera coordinates:

$$
\begin{bmatrix}
X \\
Y \\
Z \\
1
\end{bmatrix}
\rightarrow
\begin{bmatrix}
R - R \cdot \hat{C} \\
0 & 1
\end{bmatrix}
\begin{bmatrix}
\hat{X} \\
\hat{Y} \\
\hat{Z} \\
1
\end{bmatrix},
\quad (3)
$$

where $\hat{C}$ represents the coordinates of the camera center in the world coordinate frame, and $R$ is a $3 \times 3$ rotation matrix representing the orientation of the camera coordinate frame.
Towards a Continuous, Unified Calibration of Projectors and Cameras

Therefore, the resulting general pinhole camera projection matrix is given as

\[ P = K \cdot R \begin{bmatrix} I & -\tilde{C} \end{bmatrix} = K \cdot [R | \tilde{J}], \]  

(4)

where \( K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \) and \( t = -R \cdot \tilde{C}. \)

(5)

We have 9 degrees of freedom: 3 for the so-called internal parameters contained in \( K \) (the elements \( f, p_x, p_y \)), and 6 for the so-called external parameters in \( R \) and \( \tilde{C} \).

**Non-square pixels.** If the number of pixels per unit distance in image coordinates are \( m_x \) and \( m_y \) in the \( x \) and \( y \) directions, then an additional final multiplication by \( \text{diag}(m_x, m_y, 1) \) is required. This multiplication allows for non-square pixels which are often found in CCD cameras. We get

\[ K = \begin{bmatrix} \alpha_x & 0 & m_x \cdot p_x \\ 0 & \alpha_y & m_y \cdot p_y \\ 0 & 0 & 1 \end{bmatrix}, \]  

(6)

where \( \alpha_x = f \cdot m_x \) and \( \alpha_y = f \cdot m_y \) represent the focal lengths of the camera in terms of pixel dimensions in the \( x \) and \( y \) direction respectively, resulting in 10 degrees of freedom.

**Skew parameter.** By adding a skew parameter \( s \), we can define a so-called finite projective camera with 11 degrees of freedom. This is the same number of degrees of freedom as a matrix, defined up to an arbitrary scale. We thus consider a calibration matrix of the form

\[ K = \begin{bmatrix} \alpha_x & 0 & s \cdot x_0 \\ 0 & \alpha_y & s \cdot y_0 \\ 0 & 0 & 1 \end{bmatrix}. \]  

(7)

Note that for most cameras, the skew parameter will be zero.

**Decomposition.** Note that the left hand \( 3 \times 3 \) submatrix of a pinhole camera projection matrix \( P = K \cdot R \begin{bmatrix} I & -\tilde{C} \end{bmatrix} \), equal to \( K \cdot R \), is non-singular. Conversely, any \( 3 \times 4 \) matrix \( P \) for which the left hand \( 3 \times 3 \) submatrix is non-singular is a camera matrix of some finite projective camera and \( P \) can be decomposed into its submatrices. Given a \( 3 \times 4 \) camera calibration matrix \( P = [B | \tilde{b}] \) with a \( 3 \times 3 \) submatrix \( B \) and a \( 3 \times 1 \) vector \( \tilde{b} \), we can use the following procedure based on an RQ-decomposition to get the \( 3 \times 4 \) upper triangular matrix of internal parameters \( K \), the \( 3 \times 3 \) orthonormal rotation matrix from the camera to the world coordinate system \( R \), the \( 3 \times 1 \) translation vector \( \tilde{C} \) from camera to world coordinates and a scaling constant \( c \), such that \( P = c \cdot K \cdot R \begin{bmatrix} I & -\tilde{C} \end{bmatrix} \):

\[ \tilde{C} = -B^{-1} \cdot \tilde{b}, \]

\[ [K, R] = \text{RQ}(B), \]

\[ c = K_{33}, \]

\[ K = K / c. \]

By negating appropriate columns of \( K \) and the corresponding rows of \( R \), a post processing step ensures that the intrinsic values \( K_{11} \) and \( K_{22} \) are positive. If required, negative values of \( c \) are avoided by negating \( R \).

**Distortion.** We can further enhance the model by taking non-linear lens properties into account. The most common approach is to model radial and translational distortion components \( \Delta x \) and \( \Delta y \) by using a polynomial with four parameters \( K_1, K_2, P_1 \), and \( P_2 \) [7]:

\[ \Delta x = \hat{x} \cdot (K_1 \cdot r^2 + K_2 \cdot r^4) + (P_1 \cdot (r^2 + 2 \cdot \hat{x}^2) + 2 \cdot P_2 \cdot \hat{x} \cdot \hat{y}), \]
and

\[
\Delta y = \hat{y} \cdot (K_1 \cdot r^2 + K_2 \cdot r^4) + (2 \cdot P_1 \cdot \hat{x} \cdot \hat{y} + P_2 \cdot (r^2 + 2 \cdot \hat{y}^2)),
\]

where \( \hat{x} = (x, y) \) is the projected point, \( p = (p_x, p_y) \) is the principal point, \( \hat{x} = x - p_x \), \( \hat{y} = y - p_y \), and

\[
r = \sqrt{\hat{x}^2 + \hat{y}^2}.
\]

Given the duality of projector and camera, the camera model introduced in this section applies to projectors as well [8].

### 3.2. Camera calibration

Camera calibration is the process of determining the parameters of the camera model of a certain device. Since the camera model describes the relationship between a point on a 3D scene and the corresponding pixel in the projected image, such (3D point, image pixel)-correspondences are the input to each calibration method.

### 3.3. Epipolar geometry

Basically, two kinds of epipolar geometries exist. On the one hand, the classical method, called calibrated route in computer vision, requires the devices to be calibrated with respect to a global coordinate system. This allows for the calculation of the so-called essential matrix, which can be used to calculate the 3D euclidean structure of the scene, whose projections are known. Uncalibrated route on the other hand derives the so-called fundamental matrix from an uncalibrated system. Like the essential matrix, this matrix can be used to determine the 3D structure. In the following, we will mainly consider this case of epipolar geometry not requiring a calibration with respect to a global coordinate system.

The epipolar geometry between two views is essentially the geometry of the intersection of the corresponding image planes with the pencil of planes having the baseline as a rotation axis. The baseline is the line joining the optical centers of the two cameras.

Considering a single camera \( i \), the 3D point \( X \) lies on the ray from the center of projection \( C_i \) of the camera with projection matrix \( P_i \) in the direction of the projection \( x_i \) on the image plane. With two cameras and the known projections of a point in both image planes, the point is located at the intersection of the two rays back-projected from the corresponding image points. The calculation of this intersection point is called triangulation. Figure 1 shows an illustration of the epipolar geometry. The image points \( x_i \), the 3D point \( X \) and the camera centers \( C_i \) are coplanar.

[Placeholder for Figure 1]

An epipole lies at the intersection of the baseline of the two cameras with the image plane of one of the cameras. Therefore, the epipole is the projection of the center of projection of one camera into the image plane of another camera, e.g.

\[
e_{21} = P_2^{-1} \cdot (C_1 - C_2).
\]  

(8)

An epipolar plane is defined by both centers of projection and a 3D point \( X \). Each plane containing the baseline is an epipolar plane, and intersects the image planes in corresponding epipolar lines, which also represent the projection of the ray from the center of projection of the other camera to the point \( X \). As the position of the 3D point \( X \) varies, the epipolar planes rotate around the baseline. This one-parameter family of planes is known as an epipolar pencil. The respective epipolar lines intersect at the epipole.

The benefit of epipolar geometry is that the search for a point corresponding to a point in another image plane need not cover the entire image plane, but can be restricted to an epipolar line.

### 3.4. Fundamental matrix

The epipolar geometry is the intrinsic projective geometry between two views. It is independent of scene structure, and only depends on the cameras’ internal parameters and relative pose.

The fundamental matrix \( F \) encapsulates this intrinsic geometry. It is a 3 x 3 matrix of rank 2. If a point in 3-space \( X \) is imaged as \( x_i \) in one view and as \( x_j \) in another one, then the image points satisfy the relation

\[
x_i^T \cdot F_{ij} \cdot x_j = 0.
\]  

(9)

As we will see later, the fundamental matrix can also be viewed as a mapping from a point in one image to its corresponding epipolar line in the other image.
Towards a Continuous, Unified Calibration of Projectors and Cameras

An intermediary matrix $M_{ij}$ is now set up as

$$M_{ij} = \begin{bmatrix} e_{ij}^y \quad e_{ij}^z \\ e_{ij}^z \quad e_{ij}^x \\ -e_{ij}^y \quad e_{ij}^x \end{bmatrix},$$

(10)

where $e_{ij}$ is the epipole of a camera pair $(i, j)$ lying in image plane $i$. $M_{ij}$ is a screw-symmetric matrix representation of the cross product defined such that

$$M_{ij} \cdot v = e_{ij} \times v,$$

(11)

for all arbitrary vectors $v$.

The fundamental matrix $F_{ij}$ can now be defined as

$$F_{ij} = M_{ij} \cdot P_i \cdot P_j^{-1}.$$

(12)

To get the parameters of the epipolar line $Ax + By + C = 0$ in the image plane $i$ with respect to a point $x_j$ in the image plane $j$, the equation

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix} = F_{ij} \cdot x_j,$$

(13)

has to be solved.

3.5. Essential matrix

Whereas the fundamental matrix can be used in an uncalibrated route setting to compute epipolar lines, the essential matrix is used for this task in a calibrated route setup. Even though we will focus on the fundamental matrix in this article, the essential matrix is described here for completeness.

Given are two cameras with their coordinate systems, which can be transformed into each other with the help of a translation $t$ and a rotation $R$:

$$X_i = R \cdot X_j + t.$$

(14)

The essential matrix $E$ defines the algebraic representation of the epipolar geometry for a known calibration. It describes the relation between image points in the camera coordinate systems:

$$E_{ij} = T \cdot R = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \cdot R.$$

(15)

Calibrated points $x_i$ and $x_j$ satisfy the following equation:

$$x_i^T \cdot E_{ij} \cdot x_j = 0.$$

(16)

The parameters of the epipolar line $Ax + By + C = 0$ in the image plane $i$ with respect to a point $x_j$ in the image plane $j$ can now be computed using the equation:

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix} = E_{ij} \cdot x_j.$$

(17)

3.6. Homography

Constraining points in space to a plane leads to a specialization of the epipolar geometry of two views. The mapping from such a plane to the first image $i$ can be represented with a collineation of a $3 \times 3$ non-singular matrix. The same is true for the second image $j$. By composing the inverse of the first collineation with the second one, a collineation from the first image plane to the second one can be achieved. Therefore, the plane $\Pi$
induces a collineation $H_{\Pi}$ between the views, which transfers the images of points $X \in \Pi$ from one view to the other:

$$x_j = H_{\Pi} \cdot x_i,$$

where $H_{\Pi}$ is a $3 \times 3$ non-singular matrix.

### 3.7 Null-space problems

Many of our proposed calibration solutions will rely on solving a null-space problem $A \cdot x = 0$. For overdetermined systems, one usually minimizes the norm $|A \cdot x|$, i.e. the norm of the so-called residual vector $e = A \cdot x$. The minimized scalar value is called the algebraic distance, and its disadvantage is that it is not geometrically or statistically meaningful. Nevertheless, methods relying on algebraic distances can achieve good results with an appropriate choice of preconditioning (see next sections). The computation of the linear (and thus unique) solution is computationally inexpensive, and is often used as a starting point for a non-linear minimization of a geometric or statistical cost function.

**Eckart-Young-Mirsky.** For overdetermined equation systems of null-space problems, we want to find the $x$ that minimizes $|A \cdot x|$ subject to the constraint $|x| = 1$. Let $A = U \cdot D \cdot V^T$ be the singular value decomposition (SVD) of $A$. Thus, one has to minimize $|U \cdot D \cdot V^T \cdot x|$. Since $U \cdot D \cdot V^T \cdot x = D \cdot V^T \cdot x$, we can minimize $|D \cdot V^T \cdot x|$ subject to the condition $|V^T \cdot x| = 1$. Substituting $x' = V^T \cdot x$, we minimize $|D \cdot x'|$ subject to $|x'| = 1$. Since $D$ is a diagonal matrix with its entries in descending order, it follows that the solution to this problem is $x' = (0, 0, ..., 0, 1)^T$. Therefore, the solution $x = V \cdot x'$ is simply the last column of the matrix $V$, which can also be obtained as the eigenvector of $A^T \cdot A$.

**Preconditioning.** Unfortunately, the presence of systematic and random measurement errors can adversely affect the accuracy of the estimated solution $x$ of such a linear system, making efficient countermeasures necessary. The lower the condition number of the system $A$, the less the input noise and error gets amplified, and the linear system becomes more stable. Therefore, in practical cases, we have to precondition the input data used to set up the matrix $A$. As proposed by [30], points are translated such that the centroid is at the origin, and the point cloud is scaled to achieve an average distance per dimension of 1. Therefore, the average euclidean distance from the origin equals $\frac{\sqrt{\sum_i x_i^2 + y_i^2}}{n}$ in the 2D case and $\frac{\sqrt{\sum_i x_i^2 + y_i^2 + z_i^2}}{n}$ in the 3D case. So, for example, homogeneous 2D image points are transformed through an isotropic scaling matrix

$$T = \begin{bmatrix} \frac{\sqrt{\sum_i x_i^2}}{\text{rms}} & 0 & -c_x \cdot \frac{\sqrt{\sum_i x_i^2}}{\text{rms}} \\ 0 & \frac{\sqrt{\sum_i y_i^2}}{\text{rms}} & -c_y \cdot \frac{\sqrt{\sum_i y_i^2}}{\text{rms}} \\ 0 & 0 & 1 \end{bmatrix}$$

to a point cloud with centroid $[0 \ 0 \ 1]^T$ and standard deviation $[1 \ 1 \ 0]^T$ by computing

$$c_x = \frac{1}{n} \sum_{i=1}^{n} x_i, \quad \text{and} \quad \text{rms} = \frac{1}{n} \sum_{i=1}^{n} x_i^2 + y_i^2.$$

This normalization transformation additionally nullifies the effect of the arbitrary selection of origin and scale in the coordinate frame. This means that the combined algorithm is invariant to a similarity transformation, as it is the case in the minimization of geometric distances (as opposed to algebraic distances).

### 3.8 Outlier handling using RANSAC

Wrong input data points (so-called outliers) can severely affect the estimation of a least-squares solution. Therefore, special care has to be taken to handle outliers correctly and to achieve a robust estimation. For a set $S$ of data points and a model that requires at least $p$ data points to estimate its unknowns $E$, the RANSAC procedure [31] handles outliers as follows:

- The unknowns $E$ are estimated based on a random subset of $p$ point of $S$.
- A set $C$ is built, which contains all the data points of $S$ that lie within an error tolerance $t$ of the model.
- If $C$ contains more than $T$ points (the expected number of inliers), then the final model is reestimated using the set of inliers $C$, and the computed value is output before exiting the procedure.
- If we have not yet reached a maximum number of iterations $N$, then the procedure is restarted from the first step, else the largest consensus set $C$ found so far is chosen, and the final model is reestimated based on it.

4. System overview

The setup used for validating our calibration procedures consists of movable units, called bricks, each containing several cameras and a DLP projector connected to a PC (see Figure 2). The networked units can be individually oriented to cover the desired working space, usually consisting of several projection walls and desks. A microcontroller unit (MCU) is used as a synchronization source for the cameras and the graphics boards driving the projectors. Figure 3 shows a single unit with its components.

5. Small-area calibration

In this section we will present a simple calibration solution suited for a small-area setting consisting of multiple projectors and cameras. These cameras and projectors must be calibrated intrinsically and extrinsically with relation to each other, and in case of setups consisting of more than one module, with respect to a common global world coordinate system. In a small-area static setting, a fairly simple procedure can be used to compute the calibration parameters of both cameras and projectors. In our first approach, both intrinsic and extrinsic parameter estimations of the cameras are based on Intel’s Open Computer Vision library (www.intel.com/research/mrl/research/opencv).

For the projector calibration, a checkerboard pattern (see Figure 4) is projected onto two previously calibrated planes, resulting in two sets of related 3D points that enable us to calculate projector position, orientation and frustum with adequate precision.

The procedure is summarized in Figure 5. We currently ignore lens distortion of the projector. Alternatively, the computed 3D positions of the projected checkerboards could directly be used to calibrate the projector using exactly the same algorithms as used for the cameras. Our procedure is simple, but has some drawbacks when calibrating a large setting consisting of multiple projectors:

- Two planes have to be carefully overlaid with a printed calibration pattern and then augmented with a projected pattern. In between, the planes must not move. Since the process cannot be automated, the calibration procedure is rather tedious and cumbersome.
- A simplified projector model is used, where the principal point is assumed to lie in the center of the image in the horizontal direction. Furthermore, the skew parameter is set to zero in the calibration matrix.
- As a severe limitation in the proposed simple approach, the checkerboards must always be entirely visible in the camera, a requirement which cannot be fulfilled for all settings.

In order to lift the restrictions of this simple small-area calibration procedure, we present some novel, more flexible alternative approaches.

6. Homography wide-area calibration

In this section we present a versatile calibration procedure suited for wide-area projections on planar surfaces, which for most applications do not require full calibration information. In such cases, a homography mapping is usually sufficient to relate image points.

6.1. Capturing point correspondences

In a first step, we create point correspondences between the devices (both cameras and projectors) by labeling discrete points on the planar surface. Labels are assigned by subsequently projecting two orthogonal Gray code sequences in each projection device.

If a relation to metric coordinates on the plane is required, we additionally capture a single camera image of a checkerboard on the plane to be calibrated.
6.2. Propagating homography

Using the normalized direct linear transformation given by Hartley and Zisserman [32] and the previously captured point correspondences \( x'_l \leftrightarrow x'_k \) of points \( X \) on the plane \( \Pi \), we are able to compute the pairwise homography \( H_{\Pi(l,k)} \) between two devices \( l \) and \( k \), such that \( x'_l = H_{\Pi(l,k)} \cdot x'_k \).

**Linear solution.** Using the cross product, the relation \( x'_l \times (H_{\Pi(l,k)} \cdot x'_k) = 0 \) can be rewritten as \( x'_l \times (H_{\Pi(l,k)} \cdot x'_k) = 0 \). With the properties of the Kronecker product \( \otimes \) and the vec-operator, we derive a null-space problem and achieve a linear solution:

\[
\begin{bmatrix}
0^T \\
-x'_{l,m} \cdot (x'_k)^T \\
x'_{l,m} \cdot (x'_k)^T \\
-x'_{l,n} \cdot (x'_k)^T \\
-\vec{H}_{\Pi(l,k)} \cdot (x'_k)^T \\
0^T \\
x'_{l,n} \cdot (x'_k)^T \\
x'_{l,m} \cdot (x'_k)^T \\
0^T \\
\end{bmatrix}
= 0.
\]

where only two of the three equations are linearly independent. For \( n \) point correspondences, we get a \( 2n \times 9 \) matrix \( A \) by stacking up two equations for each correspondence. For \( n \geq 4 \), the homography \( H_{\Pi(l,k)} \) can be computed by solving the resulting linear system of equations. As noted in Section 3.7, the least-squares solution for \( (\vec{H}_{\Pi(l,k)}) \) is the singular vector corresponding to the smallest singular value of \( A \). Potential outlier points are handled using a RANSAC procedure (see Section 3.8, [31]). If more than four point correspondences exist, the homography can be refined by iteratively minimizing the geometric reprojection error in the least-square sense.

**Normalization.** In order to improve the conditioning of the equations, the 2D point coordinates \( x'_l \) and \( x'_k \) are normalized to \( \hat{x}'_l \) and \( \hat{x}'_k \) beforehand using transformations \( T_I \) and \( T_J \), such that the origins of the point sets are at the centroid and the distances from the origins are \( \sqrt{2} \). Substituting these values into the previous equations, we get the homography \( \hat{H}_{\Pi(l,k)} \). The final homography is obtained by

\[
H_{\Pi(l,k)} = T_I^{-1} \cdot \hat{H}_{\Pi(l,k)} \cdot T_k.
\]

**Propagation tree.** Without loss of generality, we assume that device 1 has imaged the checkerboard on the plane to calibrate. Given the homography \( H_{\Pi(1,1)} \), which relates the metric coordinates on the plane \( \Pi \) to the image coordinates in device 1, the homography \( H_{\Pi(l)} \) of device \( l \) can be computed as

\[
H_{\Pi(l)} = H_{\Pi(l,1)} \cdot H_{\Pi(1,1)}.
\]

This equation can be solved as long as enough point correspondences exist between device 1 and device \( l \). If this is not the case, device \( l \) is ignored in this first homography propagation round. Subsequently, for any uncalibrated device \( l \) and calibrated device \( k \), we try to propagate the homography

\[
H_{\Pi(l)} = H_{\Pi(l,k)} \cdot H_{\Pi(k)}.
\]

until all devices are calibrated, resulting in a tree-like propagation structure appropriate for wide-area settings.

6.3. Homography projector matrix

The reduced information gained from the homography can be used to create projection matrices which in practice can substitute full calibration matrices in projector simulations as long as the 3D points are constrained to lie on the specified plane. For example in OpenGL, the projection matrix gained from full intrinsic calibration information \( P \).
Towards a Continuous, Unified Calibration of Projectors and Cameras

can be substituted using the plane to image homography \( H_{\Pi} \) by the following matrix:

\[
\begin{bmatrix}
\frac{2 \cdot P_{11}}{\text{Res}_x} & 0 & -\frac{2 \cdot P_{13}}{\text{Res}_x} + 1 & 0 \\
0 & \frac{2 \cdot P_{22}}{\text{Res}_y} & \frac{2 \cdot P_{23}}{\text{Res}_y} - 1 & 0 \\
0 & 0 & \frac{Z_{\text{near}} + Z_{\text{far}}}{Z_{\text{near}} - Z_{\text{far}}} & \frac{2 \cdot Z_{\text{far}}}{Z_{\text{near}} - Z_{\text{far}}} \\
0 & 0 & -1 & 0
\end{bmatrix}
\]

where \( \text{Res}_x \) corresponds to the horizontal resolution of the projector, and \( \text{Res}_y \) to its vertical resolution. Before setting up the projection matrix, the homography is normalized to make sure that \( H_{\Pi13} \) is non-negative. The model-view matrix is set to the identity matrix.

7. Full wide-area calibration

In more advanced application scenarios, a homography-based calibration is not sufficient, but complete knowledge of the intrinsic and extrinsic parameters of the devices involved is required. For that purpose, we introduce a novel calibration approach suited for wide-area settings and continuous dynamic changes of the locations of the devices. During an initial calibration step, which is run once at the beginning of operation, projection properties of all components are determined in a static setting. As was shown in Section 5, methods built on traditional calibration procedures [5, 6] provide fast and reliable parameter estimation, but they are quite tedious for wide-area setups and are not designed for a unified handling of cameras and projectors, which are better included in self-calibration approaches. In an effort to combine the advantages of both traditional calibration procedures and self-calibration, we extend an approach presented by Barreto and Daniilidis [10] to setups containing multiple projectors and cameras.

One of the biggest challenges in wide-area calibration generally is the lack of an overlapping field of view of the imaging devices involved. Therefore, metric reference objects like a dotted floor or a checkerboard cannot be used. As noted in [33], previous works, like [34] generally solve the problem in a three step procedure: a) The setup is calibrated up to a collineation \( H \), using multi-view factorization [35, 36] or pair wise fundamental matrices; b) Euclidean stratification is applied to estimate \( H \); c) The radial distortion is computed using non-linear optimization.

Compared to these approaches, the method we use [10, 33] provides an option to precalibrate two or more cameras in order to avoid euclidean stratification. The precalibration is performed using standard techniques [5, 6]. Additionally, a two step multi-view factorization can simultaneously recover the projection matrices and radial distortion. This method is computationally efficient since the solution is computed by solving an eigenvproblem and no non-linear minimization is required.

The method requires synchronous image acquisition and can be used to calibrate any network where every camera has an overlapping field of view with at least another camera. The unified procedure, which propagates the euclidean structure of traditionally calibrated cameras by self-calibration, is described in detail in the following sections.

7.1. Precalibration

Propagation of euclidean structure requires precalibrated components as a basis in order to provide a euclidean world coordinate system.
Automatic precalibration. An automatic precalibration of two components labelled 1 and 2 can be obtained from imaged point correspondences \((x^1_i, x^2_i)\) by relying on euclidean stratification. Given at least 8 points, one can easily compute the fundamental matrix \(F\), such that \(x^1_i \cdot F \cdot x^2_i = 0\). The projection matrices can be defined up to a general \(4 \times 4\) collineation \(H\) as \(P_1 = \begin{bmatrix} I & 0 \end{bmatrix} \cdot H\) and \(P_2 = \begin{bmatrix} e_{21} \cdot F \cdot e_{21}^T \cdot H \end{bmatrix}\), where \(e_{21}\) and \(e_{21}^T\) are the epipole and the corresponding skew symmetric matrix. The process of finding the projective transformation \(H\) is called euclidean stratification [32] and is usually based on geometrical constraints and assumptions, which can have an impact on stability and accuracy.

Manual precalibration. Two cameras belonging to a single brick are calibrated using conventional methods by capturing common views of a reference pattern [5, 6]. While not being appropriate for the entire calibration of a wide-area setup, such a method is perfectly suitable to calibrate a small number of cameras for which the field of view overlaps. At the same time, this practical procedure registers calibration to a predefined physical environment, therefore avoiding the stratification process commonly used in self-calibration approaches and leading to more accurate results in general.

7.2. Point correspondences

In the subsequent computation steps, similarly to many self-calibration approaches, the procedure by Barreto and Daniilidis [10] is based on a set of point correspondences between views. In contrast to the original method, where the correspondences are generated by moving an LED in hundreds of unknown positions in front of the cameras, our approach relies on imperceptible projection of structured light as shown in Figure 6 [37].

Binary patterns introduced by Vuylsteke and Oosterlinck [38] are embedded in a fast sequence, both to encode column and row indices of a discrete grid of points (see Figure 7).

This approach not only allows us to create reliable point correspondences, but also permits the extraction of the geometry as soon as the system is calibrated. As opposed to the LED approach, our system has many advantages: It is not restricted to dark environments, includes the projectors into a unified calibration approach, and can run continuously during system operation in a non-disturbing way.

To ensure reliable calibration, a temporary projection surface is moved during pattern projection to fill the entire working volume with point correspondences (see Figure 8). If no movable surface is desired, a person walking through the projection volume can serve as a reflector for the patterns (see Figure 9).

7.3. Propagation of euclidean structure

The point correspondences generated imperceptibly are used to propagate calibration parameters from already calibrated components to the remaining cameras and projectors. In order to achieve a complete, consistent calibration, all devices must be connected to the precalibrated cameras over a chain of point correspondences calibrated components to the remaining cameras and projectors. In order to achieve a complete, consistent calibration, all devices must be connected to the precalibrated cameras over a chain of point correspondences calibrated components to the remaining cameras and projectors. In order to achieve a complete, consistent calibration, all devices must be connected to the precalibrated cameras over a chain of point correspondences calibrated components to the remaining cameras and projectors.

Overview. Consider a set of \(K\) devices \(D_1, \ldots, D_K\), where the first \(M \geq 2\) devices are the precalibrated cameras. In homogeneous coordinates, the image of a specific point \(X' = (X', 1)\) on a device \(D_i\) is given by

\[
\lambda_i \cdot x'_i = P_i \cdot X',
\]

where \(\lambda_i\) is a scalar depth value and \(P_i\) the \(3 \times 4\) projection matrix, which can be written as \(P_i = [A_i | a_i]\). \(A_i\) is a full rank \(3 \times 3\) matrix and \(a_i\) is a \(3 \times 1\) vector. For each device \(D_i\) we know which of the points \(X'\) have been imaged at what location \(x'_i\). Based on this input information, we want to estimate the corresponding projection properties, including the projection matrices \(P_i\). Computation of this calibration propagation is performed by first reconstructing the structure, i.e. calculating the depths of points relative to calibrated components, and then determining the projection matrices of uncalibrated components by taking the depths and the imaged point coordinates into account. Both steps are formulated as null subspace problems of matrices derived from known data according to Barreto and Daniilidis [10]. Solutions are given by the Eckart-Young-Mirsky theorem [39] as the right singular vectors corresponding to the smallest singular values of the matrices.
Towards a Continuous, Unified Calibration of Projectors and Cameras

considered (see Section 3.7). Contrary to other approaches [40, 41, 34], this approach is a one-shot method able to recover the parameters with minimum computational effort. It does not require euclidean stratification or non-linear minimization. Furthermore, all multiple-view constraints are considered simultaneously, and not only the pairwise fundamental matrices [40, 41].

**Structure reconstruction.** Without loss of generality, we assume that a certain point \( X^j \) is visible in the first \( K_R \) devices, which all have known calibration matrices \( P_i \). By multiplying Equation 22 by the skew symmetric matrix

\[
\tilde{x}^j = \begin{bmatrix}
0 & -x_{iw}^j & x_{iv}^j \\
x_{iw}^j & 0 & -x_{iu}^j \\
-x_{iv}^j & x_{iu}^j & 0
\end{bmatrix}
\]

we get \( \tilde{x}^j \cdot P_i \cdot X^j = 0, \) or

\[
\begin{bmatrix}
\tilde{x}_1^j \cdot P_1 \\
\tilde{x}_2^j \cdot P_2 \\
\vdots \\
\tilde{x}_{K_R}^j \cdot P_{K_R}
\end{bmatrix} \cdot X^j = 0,
\]

(24)

Only two of the three equations per device are linearly independent, so every calibrated camera \( i \) that has made an image of the 3D point contributes two equations. Therefore, two image points suffice to determine the four unknown homogenous coordinates of \( X^j \): If there are more points, we have an overdetermined system. The cross product achieved by the skew symmetric matrix expresses the fact that the image point \( \tilde{x}^j \) and the unnormalized projection \( P_i \cdot \tilde{X}^j \) lie on the same ray that runs through the camera center. As can be seen in the resulting equation, the point \( \tilde{X}^j \) lies in the null subspace of \( R \) and its coordinates can be computed from the imaged coordinates \( \tilde{x}^j \) using a singular value decomposition. As noted in Section 3.7, the point coordinates \( \tilde{x}^j \) should be normalized using a transformation \( T \) in a preconditioning step to get a reliable estimate of \( \tilde{X}^j \):

\[
\begin{bmatrix}
T \cdot \tilde{x}_1^j \cdot T \cdot P_1 \\
T \cdot \tilde{x}_2^j \cdot T \cdot P_2 \\
\vdots \\
T \cdot \tilde{x}_{K_R}^j \cdot T \cdot P_{K_R}
\end{bmatrix} \cdot \tilde{X}^j = 0.
\]

(25)

If desired, the initial guess of the reconstruction can be refined further by an iterative method, e.g. Levenberg-Marquardt.

**Depth estimation.** Once again, without loss of generality, we assume that a certain point \( \tilde{X}^j \) is visible in the first \( K_R \geq 1 \) devices, which all have known calibration matrices \( P_i \). Considering the first camera, it follows from Equation 22 that

\[
\tilde{X}^j = A_1^{-1} \cdot (\lambda_1 \cdot \tilde{x}_i^j - a_i).
\]

(26)

Replacing \( \tilde{X}^j \) in the projection equation of another camera yields

\[
\lambda_i \cdot \tilde{x}_i^j = \lambda_i \cdot A_i \cdot A_1^{-1} \cdot \tilde{x}_i^j + (a_i - A_i \cdot A_1^{-1} \cdot a_i),
\]

(27)

which multiplies with the skew symmetric matrix \( \tilde{x}_i^j \) results in

\[
\lambda_i \cdot \tilde{x}_i^j \cdot A_i \cdot A_1^{-1} \cdot x_i^j + \tilde{x}_i^j \cdot (a_i - A_i \cdot A_1^{-1} \cdot a_i) = 0,
\]

(28)
or

\[ M \cdot \begin{bmatrix} \lambda_1 \\ x_1 \end{bmatrix} = 0. \]  

(28)

where

\[
M = \begin{bmatrix}
\hat{x}_2^\prime \cdot A_2 \cdot A_1^{-1} \cdot x_1^\prime & \hat{x}_2^\prime \cdot (a_2 - A_2 \cdot A_1^{-1} \cdot a_1) \\
\hat{x}_3^\prime \cdot A_3 \cdot A_1^{-1} \cdot x_1^\prime & \hat{x}_3^\prime \cdot (a_3 - A_3 \cdot A_1^{-1} \cdot a_1) \\
\vdots & \vdots \\
\hat{x}_{K_D}^\prime \cdot A_{K_D} \cdot A_1^{-1} \cdot x_1^\prime & \hat{x}_{K_D}^\prime \cdot (a_{K_D} - A_{K_D} \cdot A_1^{-1} \cdot a_1)
\end{bmatrix}
\]

Ideally, the matrix \( M \) should be rank 1, but due to measurement errors, the measurement vectors generally occupy a higher dimensional manifold. Thus, \( M \) usually is full rank and the underlying null subspace needs to be estimated in order to determine the depth \( \lambda_1 \), i.e., the direction which is most likely to be the null space of the unperturbed matrix needs to be determined. Once again, according to the Eckart-Young-Mirsky theorem [39], the solution is obtained by computing the singular value decomposition \( M = U \cdot D \cdot V^T \) and selecting the column of \( V \) corresponding to the smallest singular value (see Section 3.7).

**Projection matrices.** When computing the projective matrix \( P_i \) for a device \( D_i \) with index \( K_D < i \leq K \), we assume that \( N \geq 6 \) points \( x' \) are simultaneously viewed as \( x_1^i \) by the first camera with known matrix \( P_1 \) and as \( x_j^i \) by device \( D_i \). We define \( \Psi_j = A_j \cdot A^{-1}_1 \) and \( \phi_j = a_j - A_j \cdot A^{-1}_1 \cdot a_1 \). By concatenating the columns of the \( 3 \times 3 \) matrix \( \Psi_j \) as a \( 9 \times 1 \) vector \( \psi_j \) and by introducing \( \otimes \) as the Kronecker product [39], Equation (28) can be transformed into

\[
\begin{bmatrix}
\lambda_1 \cdot \hat{x}_1^j \otimes x_1^i \\
\lambda_2 \cdot \hat{x}_2^j \otimes x_1^i \\
\vdots \\
\lambda_N \cdot \hat{x}_N^j \otimes x_1^i
\end{bmatrix} \cdot \begin{bmatrix} \psi_j \\ \phi_j \end{bmatrix} = 0.
\]

(29)

Given all \( N \) points, we derive

\[
\begin{bmatrix}
\lambda_1 \cdot \hat{x}_1^j \otimes x_1^i \\
\lambda_2 \cdot \hat{x}_2^j \otimes x_1^i \\
\vdots \\
\lambda_N \cdot \hat{x}_N^j \otimes x_1^i
\end{bmatrix} \cdot \begin{bmatrix} \psi_j \\ \phi_j \end{bmatrix} = 0.
\]

(30)

Once again the vectors \( \psi_j \) and \( \phi_j \) are estimated by applying the Eckart-Young-Mirsky theorem to the matrix \( G_j \). Finally, we get

\[
A_j = \psi_j \cdot A_1 \quad \text{and} \quad a_j = \phi_j - \psi_j \cdot a_1,
\]

(31)

and the projective matrix \( P_i \) is given as

\[
P_i = [\psi_j \cdot A_1 | \phi_j - \psi_j \cdot a_1].
\]

(32)

To summarize, the calibration algorithm performs as follows in a setup of \( K \) devices, where we assume, without loss of generality, that the projection matrices of the first \( M \geq 2 \) views are known and that the projection matrices of the remaining \( K - M \) cameras have to be determined:

- Only consider the points which are visible in the first device \( D_1 \) and in at least another calibrated device.
- For each such point, compute the depth \( \lambda_1 \) with respect to device \( D_1 \) based on Equation 28.
- For each uncalibrated device \( D_i \), determine the points viewed both in \( D_i \) and \( D_1 \), then use them to compute the projection matrix \( P_i \) according to Equations 30, 31 and 32.

**Iterative refinement.** Since neither the measured points nor the initial projection matrices are noiseless and perfect, a single run of the algorithm is usually not sufficient. We therefore iterate the process until a pre-defined
number of iterations is reached or until the reprojection error is below a predefined threshold. At each iteration, updated matrices are used to recover the depth values, which themselves are used to refine the projection matrices, therefore gradually enforcing the detected point correspondences.

**Discussion of accuracy.** It is remarkable, that the calibration propagation does not rely on a single non-linear minimization, leading to reliable solutions without convergence issues. Using proper equilibration of the design matrices \([42, 30, 43]\), reprojection errors approaching the lower theoretical bound in the order of the measurement errors are achieved. During calibration computation, outliers whose reprojection error exceeds a predefined threshold are continuously removed from the data set, leading to a procedure, which is robust against erroneous point locations. In our settings, we have found error accumulation of propagation across multiple bricks to be negligible.

**Radial distortion.** An extended version of the propagation algorithm by Barreto and Daniilidis [10] also simultaneously handles radial distortion in an elegant and unified way by estimating the radial distortion over multiple views based on a rough estimation of the distortion parameters of at least two cameras in the set. The algorithm avoids any non-linear minimization and is based on the computation of the closest rank deficient matrix to a full-rank matrix using singular value decomposition.

When modeling the radial distortion using the so-called division model [9], the image point \(x'\) without distortion and the image point \(u'\) with distortion are related by a non-linear mapping function \(f\) as

\[
x' = f^{-1}(u', \xi) = u' + \xi \cdot \left[ \begin{array}{c} (u'_u)^2 + (u'_v)^2 \\ (u'_v)^2 \\ (u'_u)^2 \\ 0, 0, 0, 1 \end{array} \right] \cdot (\psi, \phi, \alpha, \lambda),
\]

where the radial distortion is parameterized by \(\xi\). This model approximates the radial distortion curve as well as the traditional polynomial first order model (see Section 3.1). Note that the coordinates are expressed with the origin at the distortion center, which, if no better information is available, is set to the image center without significantly affecting the correction [44]. Setting \(x'_i = u'_i + \xi_i \cdot v'_i\) in Equation 30 yields

\[
A_i \cdot \begin{bmatrix} \psi_i \\ \phi_i \\ \alpha_i \end{bmatrix} + \xi_i \cdot B_i \cdot \begin{bmatrix} \psi_i \\ \phi_i \end{bmatrix} = 0,
\]

where

\[
A_i = \begin{bmatrix} \lambda_i^1 \cdot u^1_i \otimes x^1_1 \cdot u^1_1 \\ \lambda_i^2 \cdot u^2_i \otimes x^2_1 \cdot u^2_1 \\ \ldots \ldots \\ \lambda_i^N \cdot u^N_i \otimes x^N_1 \cdot u^N_1 
\end{bmatrix}
\]

and

\[
B_i = \begin{bmatrix} \lambda_i^1 \cdot v^1_i \otimes x^1_1 \cdot v^1_1 \\ \lambda_i^2 \cdot v^2_i \otimes x^2_1 \cdot v^2_1 \\ \ldots \ldots \\ \lambda_i^N \cdot v^N_i \otimes x^N_1 \cdot v^N_1 
\end{bmatrix}
\]

By multiplying with a \(12 \times 12\) permutation matrix \(I_P\), we move the four null columns of \(B_i\) to the right of the matrix.
Towards a Continuous, Unified Calibration of Projectors and Cameras

By writing the equation as a single matrix multiplication, we get

\[
\begin{bmatrix}
\alpha_i' \\
\alpha_i''
\end{bmatrix}
\begin{bmatrix}
\alpha_i' \\
\alpha_i''
\end{bmatrix} + \begin{bmatrix}
\beta_i \\
0
\end{bmatrix} \begin{bmatrix}
\alpha_i'
\alpha_i''
\end{bmatrix} = \mathbf{0}. \quad (34)
\]

A solution vector \( \beta_i = \begin{bmatrix} \beta_{i1} \beta_{i2} \ldots \beta_{i20} \end{bmatrix}^T \) is obtained by applying the Eckart-Young-Mirsky theorem to the matrix \( C_i \). In general, the resulting solution will not lie on the manifold of camera calibrations, which present a specific structure and form a non-linear subspace. We compute the nearest point on the manifold in the least-square sense. Given the singular value decomposition of the \( 8 \times 2 \) matrix

\[
\begin{bmatrix}
\beta_{i1} & \beta_{i8} & \ldots & \beta_{i16} \\
\beta_{i2} & \beta_{i9} & \ldots & \beta_{i17} \\
\beta_{i3} & \beta_{i10} & \ldots & \beta_{i18} \\
\beta_{i4} & \beta_{i11} & \ldots & \beta_{i19} \\
\beta_{i5} & \beta_{i12} & \ldots & \beta_{i20}
\end{bmatrix} = U \cdot D \cdot V^T, \quad (36)
\]

we zero the smallest diagonal element of \( D \) to obtain \( D_0 \). The solution is then given by

\[
\begin{bmatrix}
\alpha_i' \\
\alpha_i''
\end{bmatrix} = \begin{bmatrix} \beta_{i11} \ldots \beta_{i20} \end{bmatrix}, \quad (37)
\]

As an alternative, radial distortion can also either be ignored if negligible or precomputed for all devices using traditional calibration procedures.

8. Continuous adaptivity

In this section, we present our calibration adaptation approach which guarantees that at any time all devices are calibrated intrinsically and extrinsically with relation to each other. To achieve this goal, we have chosen a procedure, which can be split into two distinct parts: The initial calibration of the setup as proposed in the previous section and a continuous dynamic recalibration of modified components.

After the initial calibration has been computed, components can be moved and reoriented to accommodate new user requirements. Our recalibration procedure detects such changes during system operation and recalibrates accordingly in a very efficient manner. For simplicity, we expect intrinsic parameters to remain constant, an assumption which is often valid in practice, where components are mostly moved or reoriented.

The use of permanent autocalibration has advantages over other methods with only an initial calibration:

- The quality of the calibration is independent of the accuracy and stability of the system assemblies.
- The machinery is not sensitive to rough handling.
- Frequent alignment, which is time-consuming and may render the extended use of the equipment impossible is not required.
- Modifications of the system are easy

8.1. Design decisions

This section describes the rationale behind some of the design decisions we made for achieving calibration adaptivity.

Input data. We assume that point correspondences between the devices can be created during system operation. The imaged coordinates and labels are the only input to the continuous calibration procedure.
Constant internal parameters. The internal calibration parameters are assumed to remain constant after their determination by the initial calibration. This assumption is justified. First, because the system components often do not have zoom or other functions that would alter their internal parameters or they do not use them during operation. And second, the inevitable modifications of the internal parameters caused by temperature changes etc. are negligible.

Architecture. The system setup is a collection of equivalent peers. Therefore, a distributed solution for the calibration adaptivity seems compelling. The ease of implementation however favored a client-server solution. One peer plays the role of the “brain”, which gathers all the information from the various units, makes the necessary computations and then sends the updates back to the nodes. There are as many clients as there are units (see Section 4). The tasks of the client software are:

- Storing the calibration information of all the devices of the unit and providing this information for the other services on the unit system and the server.
- Gathering the point correspondences from the devices of the unit and sending them to the server.
- Updating the calibration with information obtained from the server.

The server software runs on a brick system or on a dedicated machine. It provides the following functionality:

- Initial calibration of the system.
- Detection of modifications in the calibrations of the system and invalidation of the corresponding calibrations at the client side.
- Recalibration of the altered devices and update of the client under the assumption that the internal calibration parameters remain constant.

Figure 11 gives a schematic overview of the dataflow between client and server.

8.2. Detection of changes

With the help of imperceptible point correspondences collected during system operation on the projection surface, the validity of calibration parameters is verified continuously. Although the checking frequency is adjustable, a new verification round usually makes sense as soon as enough new point correspondences are available to detect a modification.

Overview. For verification purposes, we rely on a pairwise comparison of components by checking whether common point correspondences are still consistent with calibration. In each camera we compute the distance between a point and the epipolar line corresponding to the respective point in the other camera. If for a component and all remaining components the distance exceeds a predefined threshold for a certain percentage of pairwise common points, we assume this component to be modified. In case an entire brick is moved at a time, this procedure will still recognize the brick as calibrated. Therefore, we have to apply the same distance test on a brick level to correctly detect rigid displacements of entire bricks. As long as a single component or brick is moved at a time and multiple elements remain static, the detection is unambiguous. Note that changes in projection geometry do not affect calibration and will properly be ignored by the verification procedure.

Epipolar line. The server maintains the information needed to compute the epipolar lines. Actually, it stores the fundamental matrices for all possible device pairs. The fundamental matrix is easily computed from the calibration matrices $P = K \cdot [I|0]$ and $P' = K' \cdot [R|R']$:

$$ F = \left[ P \cdot \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \right] \cdot \left[ P' \cdot \begin{bmatrix} K^{-1} \end{bmatrix} \right] $$

If $x$ is a point in the image that belongs to $P$, then $l = F \cdot x$ is the corresponding epipolar line in the image belonging to $P'$ (see Figure 12).

Usually, $P$ is not given in the form shown above. However, it can be transformed using the following formula, which is equivalent to a transformation of all 3D points into the camera coordinate frame of $P$ before projecting.

$$ P_1 = K_1 \cdot [R_1 | t_1], $$

$$ P_2 = K_2 \cdot [R_2 | t_2]. $$
Deviation computation. To compute the distance of a point $x' = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}^\top$ from a line $r = \begin{bmatrix} a \\ b \\ c \end{bmatrix}^\top$ the following equation (see Figure 13) is used, where the length of the projection of $r$ onto $v$ is computed:

$$d = \|\text{proj}_r r\| = \frac{|v \cdot r|}{|v|}$$

$$= \frac{|a(x-x_0) + b(y-y_0)|}{\sqrt{a^2 + b^2}}$$

$$= \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

Pairwise verification. The detection is based on the epipolar geometry of pairs of brick elements. More precisely, we use the epipolar constraint to predict the locations of corresponding points. The actual positions are then compared to the predictions, and the point correspondence is rejected if the distance of one image point from the corresponding epipolar line exceeds a certain threshold (see Figure 14). Every point correspondence yields two comparisons. If either one fails, the correspondence is rejected.

This test is done for an appropriately big number of point correspondences. An additional threshold is introduced to tolerate a certain amount of outliers. The two compared elements are considered uncalibrated if the fraction of rejected to total point correspondences exceeds the threshold (see Table 1).

The element pairs are tested in this way in a two step fashion. First each brick and its components is tested separately. Then each brick that has at least two calibrated elements participates in an inter-brick test. If one match is found, the involved bricks are considered calibrated definitely. All elements of an uncalibrated brick are marked as uncalibrated in addition to the elements already marked in the first step.

The distance threshold depends on the accuracy of the point detection method. In our setup we used a value of $\sim 5$ pixel. The point threshold also depends on the quality of the point detection algorithm. In our setup a value of $1/3$ proved to be useful. The following pseudo code shows how the verification of an element pair using these two thresholds (named accuracy and pointThreshold) is performed.

This method works well in practice. There were some considerations about false positives - wrong points that pass the test. This can happen when the wrong point lies in the plane defined by the baseline and the projection ray. A special event when this could be a problem is the displacement of an element relative to another element along their baseline. However, if there is at least one other element present at an arbitrary location (baseline does not coincide) then the displacement is detected. The test could also be implemented using the trifocal tensor to alleviate this problem. However, the epipolar constraint proved to be sufficient.

Verification algorithm. In this section we explain what happens when the verification procedure is invoked and where the Pair Verification function fits in. Please take the logical flow-diagram (Figure 15) as a guideline and consult the code snippets for algorithmical details. The verification consists of two parts, which are explained in the following.

1) First, for all possible pairs of brick-elements a pair-verification is performed (see number (1) in Figure 15). The output of the verification is stored in the following variables:

- couldBeVerified: indicates if a verification could be performed (there were enough point correspondences available).
- newCalibrated indicates if the verification was successful, i.e. if the element is still calibrated.
Towards a Continuous, Unified Calibration of Projectors and Cameras

• calibrated: captures the calibration status of an element.

Let’s consider the possible outcomes of the verification (the numbers refer to Figure 15). There are two cases where the status of the element does not change: in (2) the element was verified to still be calibrated, in (4) the verification confirmed the uncalibrated status of the element. When the calibration could not be verified (5), we have no information about the status of the element. We optimistically assume that the element is calibrated. The only case where the state of the element changed is when the former valid calibration of an element is found to be invalid (3). An invalidation-message is broadcasted. The algorithm (see Table 2) conforms to the data-flow diagram shown earlier (Figure 15).

[Placeholder for Table 2]

2) Second, the calibration is verified across the bricks. This step is incorporated to detect the movement of an entire brick whose elements (at least two of them) are still in the same position relative to each other. In this case the modification would pass the verification step undetected. For each brick pair all possible element pairings - each pair has one element from each brick - are checked using the PairVerification procedure. As soon as a match is found, the two bricks are both marked as calibrated. If no match was found over all brick pairs, then one brick is searched with a calibrated element and marked as calibrated if found. All the elements of new uncalibrated bricks are marked as not calibrated and for each an invalidate message is multicasted (see Table 3).

[Placeholder for Table 3]

8.3. Recalibration

Components, which have been detected as displaced or reoriented, need to be recalibrated extrinsically. For that purpose, imaged points are reconstructed in 3D, provided that they are at least visible in two components with valid calibration. Using this information a gradient descent method is applied to a 6-dimensional reprojection error minimization problem taking into account translational and rotational degrees of freedom. For all practical cases, previously valid extrinsic parameters have proven to be good starting values for the minimization process. Note that this procedure works well even in case of coplanar 3D point correspondences, which often occur in projections onto flat surfaces. Our efficient recalibration procedure can also be used as a substitute for the initial calibration as long as the intrinsic parameters have not been changed from the last execution of euclidean calibration propagation. Even if no component is moved, newly generated point correspondences can be used for continuously refining calibration and for compensating potential changes, which are too small and slow to be detected, but nevertheless adversely affect calibration. The steps involved in the recalibration procedure are described in detail in the following paragraphs.

Point collection. As soon as a modification is detected, a recalibration is issued. For this recalibration procedure, previously collected point correspondences are discarded and a new point set is collected. This deals with the possibility that the point set used for the verification was captured during an ongoing modification of a device. We assume that the element is no longer moved during the capture of the new point set. However, even if devices are still moved, the procedure will converge to a correct steady solution in a later verification and recalibration cycle.

The recalibration comprises the following steps:

• Reconstruction of the 3D points - the structure - using the image points and the calibration data of the valid elements.

• Computation of the external calibration parameters - the motion - of the uncalibrated elements using the image points and the reconstructed structure.

Reconstruction of the structure. The reconstruction step computes the 3D points from the image points and the calibration data of the valid elements. For this computation the DLT-method is used, as described in Section 7.3.

Update extrinsics. A first approach to recomputing the calibration parameters is using the DLT-method as well. Given the image points and the reconstructed 3D points the calibration parameters are obtained by solving the following equation:

\[
s \cdot x = s \cdot \begin{bmatrix} u \\ v \end{bmatrix} = P \cdot X = P \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix},
\]
where \( x \) is the image point of the element with projection matrix \( P \), and \( X \) is the reconstructed 3D point. This is a system of three equations in the twelve unknown elements of \( P \):

\[
\begin{align*}
s \cdot u &= P_{11} \cdot X + P_{12} \cdot Y + P_{13} \cdot Z + P_{14}, \\
s \cdot v &= P_{21} \cdot X + P_{22} \cdot Y + P_{23} \cdot Z + P_{24}, \\
s &= P_{31} \cdot X + P_{32} \cdot Y + P_{33} \cdot Z + P_{34}.
\end{align*}
\]

The scaling factor \( s \) can be eliminated by making the following transformations:

\[
(Eq1) \leftarrow (Eq1) - u \cdot (Eq3),
\]

\[
(Eq2) \leftarrow (Eq2) - v \cdot (Eq3),
\]

which leads to a system of two equations:

\[
\begin{align*}
0 &= (P_{11} - u \cdot P_{31}) \cdot X + (P_{12} - u \cdot P_{32}) \cdot Y + (P_{13} - u \cdot P_{33}) \cdot Z + (P_{14} - u \cdot P_{34}), \\
0 &= (P_{21} - v \cdot P_{31}) \cdot X + (P_{22} - v \cdot P_{32}) \cdot Y + (P_{23} - v \cdot P_{33}) \cdot Z + (P_{24} - v \cdot P_{34}).
\end{align*}
\]

This can be written in matrix form:

\[
\begin{bmatrix}
X & Y & Z & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -u \cdot X & -u \cdot X & -u \cdot X & -u \cdot X \\
0 & 0 & 0 & 0 & X & Y & Z & 1 & -v \cdot X & -v \cdot X & -v \cdot X & -v \cdot X
\end{bmatrix}
\begin{bmatrix}
P_{11} \\
P_{12} \\
\vdots \\
P_{34}
\end{bmatrix}
\]

\[
= A \cdot P = 0.
\]

Every (image point, 3D point)-pair contributes two rows to the matrix \( A \). Therefore, six point pairs would be enough to determine the values of the twelve unknowns. But like in the reconstruction step above, the noise in the point-capturing and detection process, the limited accuracy of the reconstructions and the presence of a certain amount of outliers favors an overdetermined system. The solution is again obtained by computing the singular value decomposition of \( A \).

This linear solution would have been used as a starting value for further improvement of the calibration parameters by an optimization procedure. But the eleven degrees of freedom and the possibly planar structure of the 3D points allowed solutions to the problem that were not the expected ones (see Figure 16). Tests of the procedure in a simulator and in practice introduced solutions that were the point-mirror-images (green) of the real solutions (black). Additionally, another mirror image (red) with the projection wall as a mirror-plane was obtained as an alternative solution, together with the point mirrored image of that one.

[Placeholder for Figure 16]

This erratic behavior led us to discard the DLT-solution and take the last valid calibration data as a starting point for an optimization process instead. We chose to adapt the existing method cvFindExtrinsicCameraParams_64d contained in OpenCV. The parameters of the method are the image points captured by the camera, which is to be calibrated, and the corresponding 3D points as well as the last valid external calibration parameters. For the input, the rotation matrix has to be converted to a rotation vector. As output, the method provides the updated external calibration parameters. From the method body, we used only the minimization loop. It searches the calibration parameter values that minimize the reprojection error. The minimization is done with the Newton method. The starting values of the parameters are taken from the old calibration data.

With one parameter the Newton method works as follows: \( e \) is the function to be minimized, \( \kappa \) is the step number. In each step an update \( \Delta p \) is computed which brings us from \( p(\kappa) \) to \( p(\kappa + 1) \):
Towards a Continuous, Unified Calibration of Projectors and Cameras

\[
\frac{\delta \varepsilon(\rho(\kappa))}{\delta \rho} \cdot \Delta \rho(\kappa) = \varepsilon(\rho(\kappa)),
\]

where

\[
\Delta \rho(\kappa) = \rho(\kappa) - \rho(\kappa + 1),
\]

and

\[
\rho(\kappa + 1) = \rho(\kappa) - \frac{\varepsilon(\rho(\kappa))}{\bar{\rho}\rho(\kappa)}. \nabla \varepsilon(\rho(\kappa))^\top.
\]

We have six parameters \( \rho_j \) : three for the orientation and three for the translation. Every image point contributes one error \( \varepsilon_j \). This extends the above univariate version along the parameter and the error dimension. In matrix form, the equation looks almost the same as Equation 38, but \( \varepsilon \) and \( \Delta \rho \) now are vectors:

\[
\nabla \varepsilon(\rho(\kappa)) \cdot \Delta \rho(\kappa) = \varepsilon(\rho(\kappa)),
\]

where

\[
\Delta \rho(\kappa) = \begin{bmatrix} \Delta \rho_1 \\ \Delta \rho_2 \\ \vdots \\ \Delta \rho_6 \end{bmatrix} \quad \text{and} \quad \varepsilon(\rho(\kappa)) = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}.
\]

\( \nabla \varepsilon(\rho(\kappa)) \) is the Jacobian with dimension \( n \times 6 \), where \( n \) is the number of image points:

\[
\nabla \varepsilon(\rho(\kappa)) = \begin{bmatrix} \delta \varepsilon_1(\rho(\kappa)) & \delta \varepsilon_1(\rho(\kappa)) & \cdots & \delta \varepsilon_1(\rho(\kappa)) \\ \delta \varepsilon_2(\rho(\kappa)) & \delta \varepsilon_2(\rho(\kappa)) & \cdots & \delta \varepsilon_6(\rho(\kappa)) \\ \vdots & \vdots & \ddots & \vdots \\ \delta \varepsilon_n(\rho(\kappa)) & \delta \varepsilon_n(\rho(\kappa)) & \cdots & \delta \varepsilon_n(\rho(\kappa)) \end{bmatrix}.
\]

The Equation 38 is transformed to

\[
\Delta \rho = [(\nabla \varepsilon(\rho(\kappa)))^\top \cdot \nabla \varepsilon(\rho(\kappa))]^{-1} \cdot (\nabla \varepsilon(\rho(\kappa)))^\top \cdot \varepsilon(\rho(\kappa)).
\]

Now that all the ingredients of the loop have been defined, the pseudo code can be found in Table 4.

[Placeholder for Table 4]

While the motion is iteratively refined using feedback by reprojection, the structure is computed as a one-step solution of a linear method. Since the structure is an input of the motion computation, but is not refined in the minimization loop, its accuracy is essential to get a good recalibration solution. Another approach that uses the feedback from the reprojections to adjust the camera parameters as well as the 3D points in a global minimization step is proposed by [45].

**Fallback.** The calibration procedure from Section 7 is the last resort to fall back on, in case the automatic recalibration fails. Failure occurs when there are less than two remaining calibrated elements, because a minimum of two calibrated elements are required to reconstruct 3D points.

9. Results

Our unified, continuous calibration has been tested both in a simulation and in real-world settings.

9.1. Simulator

The simulator takes the role of a calibration client. But instead of interacting with the server over a network, it has direct access to the data structures of the server. The simulator generates a few brick elements by choosing
certain calibration parameters. It then registers these elements at the server. The simulator has the advantage over the real setup to be able to control some additional parameters. These parameters include:

- The set of known 3D points. This allows to check the reconstruction step.
- The noise level in the image point positions. This parameter allows to model the inaccuracy of the camera capturing and the point detection processes.
- The fraction of outliers. This allows to introduce an exact percentage of wrongly detected points.
- The element modifications. The movement of an element - which decomposes into a rotational and a translational part - can be exactly determined. This allows to perform test series where one parameter at a time is modified.

**Verification.** The important step in the verification procedure is the computation of the error as the geometric distance between image point and epipolar line. Tests showed that even minor element movements are detected with a fairly high accuracy threshold of about 10 pixels. Experiments with different noise levels suggested a minimal value for the accuracy threshold that is about as high as the accuracy of the detection method. If a point can be detected with an accuracy of 0.5 pixels, then the minimal accuracy threshold should be in the order of 0.5 pixels as well. It is important to note that although the distances are measured in pixels, the points can be detected with subpixel accuracy. On the other hand, the point threshold should be high enough to tolerate a certain fraction of outliers (wrongly detected points). Typically, the fraction of outliers is very small, i.e., below 1%. This is mainly because outliers are removed during point detection. On the other hand, the point threshold should not be too high, so that an element modification does not remain undetected. In our tests, when we dislocated an element, the fraction of points that did not pass the verification step was always over 90%. We used a point threshold of 30% in our tests. And this choice proved to be useful.

**Recalibration (structure).** The following figures show the error in reconstruction depending on a changing parameter. The relative error is defined as the absolute error - which is the euclidean distance of the reconstruction from the original 3D point - divided by the centroid of the generated 3D points. In our test world we used millimeters as distance metric. The centroid has a distance from the camera center of 3 meters. Therefore, when the relative error is $10^{-3}$, then the absolute error is 3 millimeters.

[Placeholder for Figure 17]

In Figure 17 the relative error of the reconstruction is plotted against different levels of noise. The noise level is the radius of a circle around the exact image point. The noised point can lie anywhere in this circle with equal probability. The most important information here is that the relative error does not depend significantly on the number of points used for the reconstruction. The noise clearly is dominant. Figure 18 shows the dependence of the relative reconstruction error on the fraction of outliers. This error is significantly bigger than the error introduced by noise: the error with only 1% of outliers is bigger than the error with 2 pixels of noise. Additionally, the error grows with the number of elements.

[Placeholder for Figure 18]

This behavior is introduced by the method used to model the outliers. A fraction one third of outliers is obtained by assigning random values to every third detected point. If in the reconstruction of a 3D point only one outlier is present, then the reconstruction is totally different from the original 3D point. The probability that there is no outlier among the detected points used for reconstructing one 3D point, diminishes with every additional brick element. With 16% of outliers and 4 elements, only 50% of the 3D points are reconstructed correctly. In practice, the wrongly detected points are mostly correlated. For example, a reflection is captured by all cameras simultaneously. So all image-point correspondences of this reflection point are false. This is a deficiency of the chosen model of outliers. But nevertheless, these results identified the reconstruction method as the weakest link in the calibration chain. However, the low percentage of outliers makes the implemented method useful and adequate in practice.

**Recalibration (motion).** The first recalibration approach used the DLT method as in the reconstruction. This yielded among the expected result also unexpected ones, which were identified as symmetric solutions to the original one. This is not due to an implementation error, but is rather problem-inherent. The second approach used the Newton method to minimize the reprojection error. The result of the DLT method was used as a starting point for the minimization. Again, the expected result was obtained only occasionally. The final solution was to use the last known calibration as a starting point for the minimization. The minimization converges to the right solution as long as the element modification does not exceed a certain range. Figure 20, Figure 21 and Figure 22 show some test results to confirm this statement. The test setup was a single camera facing a $100 \times 100$ regular grid of generated 3D points, as shown in Figure 19. There were thresholds determined for the translation along
the camera axes and for the rotation around them. For the translation, no thresholds could be determined. The minimization always converged to the right solution (Figure 20). For the rotation, several thresholds could be determined (Figure 21 and Figure 22). The important fact here is not the exact value of the threshold, but the fact that in all the tests, the right solution was obtained if the rotation angle was smaller than some threshold angle. So instead of error measures, the figures contain only qualitative regions.

The source of the wrong solutions lies in the nature of global minimization. The reprojection error function is a function in six variables. It has many local minima. Which minimum is reached by the Newton method depends on the starting point. Additionally, a chaotic behavior could be noticed. Some solutions had very big values (~$\pm 10^{20}$) that changed dramatically with only small camera parameter changes. This behavior might be produced by the method used to compute the calibration parameter update in the iteration of the recalibration. The multiplication of the Jacobian with its transpose to avoid inverting a big matrix is known to be unstable. The actual solution works if the element movement, especially the rotational component, is not too big. In practical use, the element movements are smaller than the computed thresholds. In any case the elements can be moved across a long distance by several small movements and in-between recalibrations.

9.2. Real-world setting

Based on the methods presented in this paper, we have implemented a projector-camera system using off-the-shelf components in a standard office environment. Our setup operates with three bricks (see Section 4), each consisting of a standard PC with an NVIDIA Quadro FX3000G graphics board, a projector, a greyscale camera and two color cameras. The system is complemented by a custom-made synchronization device and a network hub. The projectors and cameras are mounted on a portable aluminum rig. We use NEC LT240K projectors with XGA resolution ($1024 \times 768$) and Point Grey Dragonfly cameras with the same resolution. Various projections containing imperceptible patterns and the corresponding camera captures are shown in Figure 23. Although the cameras show smearing artifacts due to short shutter times, the patterns are clearly visible.

**Full wide-area calibration.** With the help of imperceptible point correspondences extracted from the patterns, the initial calibration computes the arrangement of the projection setup and the intrinsic parameters of its components. In our example, the radial distortion of the camera lenses is precomputed using traditional calibration procedures. The resulting brick layout and the reconstructed calibration points in the working volume are shown in Figure 24. For all components, root-mean-square reprojection errors below three pixels are achieved. In our current implementation, which has not yet been optimized for speed, our point collection for initial calibration takes approximately three minutes. We expect at least a tenfold increase in speed in a tuned system.

Once the initial calibration has been computed, a projected display can be created over multiple overlapping projections.

It is beyond the scope of this article to analyze the accuracy of the euclidean propagation. Please refer to [10]. In addition to real imagery results, the authors of the original approach have analyzed the behavior of the algorithm in simulations, achieving very satisfactory results.

**Continuous adaptivity.** The continuous calibration verification and the subsequent recalibration step are illustrated in Figure 25. After detecting displaced components, in our case a projector, the prototype system automatically recalibrates using the imperceptibly generated point correspondences.

Since complete calibration information and knowledge of the brick layout are available, screen redefinition is not restricted to a single projection surface. Therefore, displays can also be moved to new surfaces as shown in Figure 26.
Towards a Continuous, Unified Calibration of Projectors and Cameras

10. Applications
We have successfully applied our calibration technique in a wide range of applications, ranging from a 3D painting prototype supporting head-parallax [37], over 3D video acquisition [46] to ubiquitous displays on walls [47] and tabletops [48]. In general, our system is applicable to any system, which includes both projectors and cameras.

11. Conclusions and future work
In this article, we have introduced several calibration approaches suited for a large number of projectors and cameras. Each approach has its own advantages and drawbacks as shown in Table 5. [Placeholder for Table 5]
In our most elaborate calibration version relying on self-calibration, a projector-camera system is easy to install, handles all components in a unified way and allows continuous dynamic setup changes. Generation of point correspondences for calibration can be performed concurrently with the main application of the system in a non-disturbing way. This can be achieved by using an imperceptible structured light approach enabling per-pixel surface light control.
The auto-calibration method for a camera-projector system can replace an existing traditional calibration method of a system which usually requires a person to wave about a calibration object; a procedure that often has a duration of several minutes and that has to be repeated every time an element is modified, i.e. relocated or moved.

Possible extensions. There are some alternatives to the methods used for the calibration verification and the recalibration. One could use the 3-view-geometry for the calibration verification step instead of the epipolar geometry. The other is the usage of a global minimization of the reprojection error that adjusts both the 3D points and the calibration parameters as implemented by [45]. The calibration method could be extended to use the point captures to iteratively refine the computed calibration, when the element was not modified. For this a method like [27] could be used.

Acknowledgments
We thank Joao P. Barreto for sharing his implementation of euclidean structure propagation.

References
[13] T. Pajdla, T. Werner, V. Hlavac, Correcting Radial Lens Distortion without Knowledge of 3D Structure,
Towards a Continuous, Unified Calibration of Projectors and Cameras


[45] M. I. Lourakis, A. A. Argyros, The design and implementation of a generic sparse bundle adjustment soft-


Figure captions

Figure 1: Epipolar geometry with epipoles $e_{12}$ and $e_{21}$.
Figure 2: Configuration of the setup to be calibrated.
Figure 3: Single unit with projector and cameras.
Figure 4: Calibration pattern.
Figure 5: Calibration of a projector using precalibrated planes. a) The location of the back plane is determined with respect to a calibrated camera. b) A checkerboard projected onto the back plane is recorded and the coordinates of the edges are determined. c) The location of the front plane is determined with respect to a calibrated camera. d) A checkerboard projected onto the front plane is recorded and the coordinates of the edges are determined. e) The center of projection is determined as the least-squares intersection of the rays through the projected edges on the back and front planes. f) By shifting planes delimiting the frustum and then intersecting them, we generate a possible image plane, defining the intrinsic parameters of the projector. The principal point is obtained as the perpendicular projection of the center of projection. The image plane also defines the projector coordinate system and therefore the extrinsic parameters.
Figure 6: Projected image and capture of imperceptible structured light (see inset).
Figure 7: Patterns for encoding column and row indices.
Figure 8: Temporary projection surface for point generation.
Figure 9: Person as a reflector in the projection volume.
Figure 10: Propagation using point correspondences.
Figure 11: Dataflow-diagram.
Figure 12: Epipolar line $i$ in an image plane corresponding to the image point $x$ in another image plane.
Figure 13: Computation of the distance between a point $x' = \begin{bmatrix} x'_0 & y'_0 \end{bmatrix}^T$ and a line $v = \begin{bmatrix} a & b & c \end{bmatrix}^T$.
Figure 14: Epipolar constraint. The detected point lies outside the range around the epipolar line. Therefore, it is rejected.
Figure 15: Logical flow-diagram of the verification of elements in one brick.
Figure 16: Solutions for the camera position and orientation obtained using the DLT-method.
Figure 17: Relative error of the reconstruction depending on different noise levels.
Figure 18: Relative reconstruction error depending on different fractions of outliers. The noise level is set to 0.5 pixels.
Figure 19: Test setup for determining the motion thresholds.
Figure 20: Convergence of the recalibration step when the element is translated along the x-axis.
Figure 21: Convergence of the recalibration step when the element is rotated about the y-axis.
Figure 22: Convergence of the recalibration step when the element is rotated about the z-axis.
Figure 23: Projected images and captured patterns.
Figure 24: Computed 3D layout of the brick setup and its components. Three projectors were imaging patterns for imperceptible point correspondences.
Figure 25: Adaptivity of calibration. a) Misaligned display resulting from projector displacement during system operation. b) Projection after automatic recalibration.
Figure 26: Instant displays can be redefined with a laser pointer, allowing a user to move projections to new surfaces.
Towards a Continuous, Unified Calibration of Projectors and Cameras

---

### Tables

#### Table 1: Pairwise calibration verification.

```cpp
def PairVerification(BrickElement brickElement1, BrickElement brickElement2):
    F = getFundamentalMatrix(brickElement1, brickElement2);
    F_ = getFundamentalMatrix(brickElement1, brickElement2);
    # compare the image points of the two elements */
    acceptedPoints = rejectedPoints = comparedPoints = 0;
    while (isPointCorrespondenceLeft()) {
        getNextPointCorrespondence(x, x_);
        e_ = F * x;
        e = F_ * x_
        d_ = distance(e_, x_);
        d = distance(e, x);
        if ((d < accuracy) && (d_ < accuracy)) {
            acceptedPoints++;
        } else {
            rejectedPoints++;
        }
        comparedPoints++;
    }
    if (comparedPoints < minimalNumberOfPoints) {
        # we did not have enough points for comparison */
        returnValue = false;
    } else {
        # verification was performed */
        brickElement1.calibrationCouldBeVerified = true;
        brickElement2.calibrationCouldBeVerified = true;
        if ((rejectedPoints / comparedPoints) > pointThreshold) {
            returnValue = false;
        } else {
            returnValue = true;
        }
    }
    return returnValue;
```

#### Table 2: Intra-brick calibration verification.

```cpp
def CCalibrationServerVerification::VerifyCalibrationInsideBricks(Bricks bricks) {
    /* we first update the status of the brick elements relative to the brick */
    for each brick in bricks {
        /* reset for next round */
        for each brickElement in brick.elements {
            brickElement.newCalibrated = false;
            brickElement.calibrationCouldBeVerified = false;
        }
        /* compute the new calibration status and */
        for each brickElement1 in brick.elements {
            for each brickElement2 after brickElement1 in brick.elements {
                if (brickElement1.calibrated && brickElement2.calibrated) {
                    match = PairVerification(brickElement1, brickElement2);
                    if (match) {
                        // default value of is false
                        brickElement1.newCalibrated = true;
                        brickElement2.newCalibrated = true;
                    }
                }
            }
        }
        /* copy new calibration status values */
        for each brickElement in brick.elements {
            /* if the element is not marked as calibrated but
            has never been verified then we assume that it still
            is (we are nice). invalidation is therefore only
            performed if we are sure that something has
            changed */
            if (!brickElement.calibrationCouldBeVerified
                && !brickElement.newCalibrated) {
                brickElement.newCalibrated = true;
            }
            /* invalidate brick elements for which the status has
            changed from calibrated to not calibrated */
            if (brickElement.calibrated && !brickElement.newCalibrated) {
                multicastInvalidateCalibrationMessage(brickElement);
            }
            /* copy the status value */
            brickElement.calibrated = brickElement.newCalibrated;
            /* reset for next round */
            brickElement.newCalibrated = false;
            brickElement.calibrationCouldBeVerified = false;
        }
    }
```
Table 3: Inter-brick calibration verification.

```cpp
bool CCalibrationServerVerification::VerifyCalibrationAcrossBricks(Bricks bricks) {
    /* first reset the brick calibration status */
    for each brick in bricks {
        brick.calibrated = false;
    }
    brickMatchFound = false;
    for each brick1 in bricks {
        for each brick2 after brick1 in bricks {
            for each pair(brickElement1, brickElement2) with brickElement1 in brick1.elements
                and brickElement2 in brick2.elements
                and brickElement1.calibrated == true
                and brickElement2.calibrated == true
            {
                if PairVerification(brickElement1, brickElement2) {
                    brick1.calibrated = true;
                    brick2.calibrated = true;
                    /* break out of the inner loop,
                       continue with next brick */
                    break;
                }
            }
        }
    }
    calibrationFound = false;
    if (!brickMatchFound) {
        /* we have to find the brick with calibrated components
           we assume the right one is always the first one */
        for each brick in bricks {
            for each element in brick.elements {
                if (element.calibrated) {
                    brick.calibrated = true;
                    calibrationFound = true;
                    break;
                }
            }
            if (calibrationFound) { break; }
        }
    } else {
        calibrationFound = true;
    }

    /* set the elements of the uncalibrated bricks to uncalibrated */
    for each brick in bricks {
        if (!brick.calibrated) {
            for each element in brick.elements {
                if (element.calibrated) {
                    element.calibrated = false;
                    /* invalidate brick elements for which the
                       status has changed from calibrated to not
                       calibrated */
                    MulticastInvalidateCalibrationMessage(element);
                }
            }
        }
    }
    return calibrationFound;
}
```
Table 4: Update of the extrinsic calibration parameters.

```c
/* initialization of variables used in the loop condition */
iter = 0;
change = 1.0f;

/* initialization of the parameter vector */
icvCopyVector_64d( rotVect, 3, vect_Param );
icvCopyVector_64d( transVect, 3, vect_Param + 3 );

/* start of the loop, at most 20 iterations are made */
while( (change > 1e-10) && (iter < 20) )
{
    /* projection of the object points */
icvProjectPoints( // input */
        numPoints, objectPoints,
        rotVect, transVect,
        focalLength, principalPoint,
    // output */
        projectImagePoints,
        derivPointsRot, derivPointsTrans);

    /* computation of the error vector ex_points */
icvSubVector_64d( (double *) imagePoints,
        (double *) projectImagePoints,
        (double *) ex_points, numPoints * 2 );

    /* computation of the Jacobian matrJJ */
    for( t = 0; t < numPoints * 2; t++ )
    {
        icvCopyVector_64d( derivPointsRot + t * 3, 3,
            matrJJ + t * 6 );
        icvCopyVector_64d( derivPointsTrans + t * 3, 3,
            matrJJ + t * 6 + 3 );
    }

    /* preparing the transformations of the Jacobian to eventually */
    /* compute the update as in Equation */
icvMulTransMatrixR_64d( matrJJ, 6, 2 * numPoints, matrJJtJJ );
icvInvertMatrix_64d( matrJJtJJ, 6, invmatrJJtJJ );
icvTransposeMatrix_64d( matrJJ, 6, 2 * numPoints, matrJJt );

    /* computation of the update vectParam_innov */
icvMulMatrix_64d( matrJJt, 2 * numPoints, 6,
        tmp6 );
icvMulMatrix_64d( invmatrJJtJJ, 6, 6, tmp6, 1, 6,
        vectParam_innov );

    /* computation of the updated parameter vector vect_Param_up */
icvAddVector_64d( vect_Param, vectParam_innov,
        vect_Param_up, 6 );
norm1 = icvNormVector_64d( vect_Param_up, 6 );
norm2 = icvNormVector_64d( vect_Param, 6 );

    /* computation of the relative change of the parameter vector */
    change = norm1 / norm2;

    icvCopyVector_64d( vect_Param_up, 6, vect_Param );
icvCopyVector_64d( vect_Param, 3, rotVect );
icvCopyVector_64d( vect_Param + 3, 3, transVect );
    iter++;
}
```

Table 5: Advantages and drawbacks of the proposed calibration approaches

<table>
<thead>
<tr>
<th>CALIBRATION</th>
<th>ADVANTAGES</th>
<th>DRAWBACKS</th>
</tr>
</thead>
</table>
Figures

Figure 1: Epipolar geometry with epipoles $e_{12}$ and $e_{21}$.

Figure 2: Configuration of the setup to be calibrated.

Figure 3: Single unit with projector and cameras.

Figure 4: Calibration pattern.
Figure 5: Calibration of a projector using precalibrated planes. a) The location of the back plane is determined with respect to a calibrated camera. b) A checkerboard projected onto the back plane is recorded and the coordinates of the edges are determined. c) The location of the front plane is determined with respect to a calibrated camera. d) A checkerboard projected onto the front plane is recorded and the coordinates of the edges are determined. e) The center of projection is determined as the least-squares intersection of the rays through the projected edges on the back and front planes. f) By shifting planes delimiting the frustum and then intersecting them, we generate a possible image plane, defining the intrinsic parameters of the projector. The principal point is obtained as the perpendicular projection of the center of projection. The image plane also defines the projector coordinate system and therefore the extrinsic parameters.
Figure 6: Projected image and capture of imperceptible structured light (see inset).

Figure 7: Patterns for encoding column and row indices.

Figure 8: Temporary projection surface for point generation.

Figure 9: Person as a reflector in the projection volume.
Figure 10: Propagation using point correspondences.

Figure 11: Dataflow-diagram.

Figure 12: Epipolar line $i$ in an image plane corresponding to the image point $x$ in another image plane.

Figure 13: Computation of the distance between a point and a line $v = \begin{bmatrix} a & b & c \end{bmatrix}$.
Figure 14: Epipolar constraint. The detected point lies outside the range around the epipolar line. Therefore, it is rejected.

Figure 15: Logical flow-diagram of the verification of elements in one brick.

Figure 16: Solutions for the camera position and orientation obtained using the DLT-method.
Towards a Continuous, Unified Calibration of Projectors and Cameras

![Figure 17: Relative error of the reconstruction depending on different noise levels.](image17)

![Figure 18: Relative reconstruction error depending on different fractions of outliers. The noise level is set to 0.5 pixels.](image18)

![Figure 19: Test setup for determining the motion thresholds.](image19)
Towards a Continuous, Unified Calibration of Projectors and Cameras

**Figure 20:** Convergence of the recalibration step when the element is translated along the x-axis.

**Figure 21:** Convergence of the recalibration step when the element is rotated about the y-axis.

**Figure 22:** Convergence of the recalibration step when the element is rotated about the z-axis.
Figure 23: Projected images and captured patterns.

Figure 24: Computed 3D layout of the brick setup and its components. Three projectors were imaging patterns for imperceptible point correspondences.

Figure 25: Adaptivity of calibration. a) Misaligned display resulting from projector displacement during system operation. b) Projection after automatic recalibration.

Figure 26: Instant displays can be redefined with a laser pointer, allowing a user to move projections to new surfaces.