Extrinsic recalibration in camera networks

Extrinsic Recalibration in Camera Networks

Chris Hermans[1], Maarten Dumont[1,2], Philippe Bekaert[1]
{chris.hermans, maarten.dumont, philippe.bekaert}@uhasselt.be

[1]Hasselt University - Expertise Centre for Digital Media
transnationale Universiteit Limburg
Wetenschapspark 2, 3590 Diepenbeek, Belgium

[2]Interdisciplinary institute for BroadBand Technology
IBBT - Expertise Centre for Digital Media
Diepenbeek, Belgium

Abstract

This work addresses the practical problem of keeping a camera network calibrated during a recording session. When dealing with real-time applications, a robust calibration of the camera network needs to be assured, without the burden of a full system recalibration at every (un)intended camera displacement. In this paper we present an efficient algorithm to detect when the extrinsic parameters of a camera are no longer valid, and reintegrate the displaced camera into the previously calibrated camera network. When the intrinsic parameters of the cameras are known, the algorithm can also be used to build ad-hoc distributed camera networks, starting from three calibrated cameras. Recalibration is done using pairs of essential matrices, based on image point correspondences. Unlike other approaches, we do not explicitly compute any 3D structure for our calibration purposes.

1 Introduction

When recording images for real-time applications, a robust camera calibration needs to be guaranteed during the entire recording session. It is often not possible to interrupt the capture at every occurrence of (un)intended camera movement, in order to perform a full system recalibration. Even when we can shut down the system and completely recalibrate the entire camera network, this proves to be a time consuming operation. Moreover, in case of a single moving camera, there is no immediate need to recalibrate the *entire* camera network. Compared with the previous calibration information, there is only a small subset of changed variables, namely the extrinsic parameters of the moved camera.

Therefore, we have developed a simple and elegant algorithm that detects camera movement, and consequently computes a robust extrinsic recalibration of the moved camera. During this computation, we use available information of the remaining calibrated cameras and the known intrinsic calibration of the moved camera, in order to reintegrate the camera in the network coordinate frame.

2 Related Work

2.1 Calibration of Camera Networks

Calibration algorithms for multi-camera setups are generally divided in four steps [9]: (1) detect feature points in each camera and calculate correspondences between the different views, (2) perform an initial reconstruction, (3) apply bundle adjustment and (4) upgrade to a metric reconstruction.

For the purpose of initial calibration, the first step is usually facilitated by the use of a calibration object. This object can come in many different flavors, usually either a planar surface with an imprinted binary pattern (such as a checkerboard, parallel lines [2] or evenly spaced circles or boxes [2, 20, 22]), or a single moving bright spot (such as a laser pointer) [18, 19]. All these methods provide point-point and/or line-line constraints, whose accuracy is critical to the success of the rest of the algorithm.

In comparison to these methods, our algorithm is applied during capturing time, and therefore does not have the luxury of being able to use a specifically designed calibration object. Even though in some controlled environments, the calibration object can be an explicit part of the scene (e.g. a checkerboard pattern on the floor, etc...), we make no such assumptions.

As our technique is intended for a wide variety of applications, we make no assumptions about the nature of the camera baselines. Therefore we employ SIFT [12], the current state-of-the-art feature detector and descriptor. More recently, an interesting alternative for real-time applications has emerged in the form of SURF [3].

The second step (the initial reconstruction method) is the key difference between most of the multi-camera algo-

rithms. These techniques can roughly be labeled as either top-down or bottom-up methods.

Algorithms that employ a top-down approach are generally employed in controlled environments, such as a lab or a recording studio, calibrating the cameras all at once. A widely used approach of this category is the projective factorization proposed by Martinec & Pajdla [14]. This method requires the estimation of projective depths for proper initialization, which is achieved using epipolar geometry as done by Sturm & Triggs [17]. Occlusion handling is solved by an extension of the method by Jacobs [11] to fill in missing data. This extension can exploit the geometry of the perspective camera so that both points with known and unknown projective depths are used.

Algorithms of the second category are often more suitable for less controlled environments, distributed applications [13] and video sequences [5] (a dynamic scene viewed by a single camera, as opposed to multiple viewpoints of the same static scene). The general strategy here is to first perform a local calibration for one or more small clusters of cameras, and then gradually converge to a solution using the previously calculated building blocks.

The method used by Sinha *et al.* [16] shows some resemblance to our own. They first resolve for a set of three cameras with non-collinear centers, for which the three fundamental matrices F_{12}, F_{13}, F_{23} have been computed. Given these, they calculate a corresponding triplet of camera matrices P_1, P_2, P_3. This provides a general projective frame for the rest of the reconstruction. To complete the N-view camera network, they then inductively add each of the remaining cameras. The key observation in this method is that, given camera matrices P_1, P_2 and fundamental matrices F_{12}, F_{13}, the third camera matrix P_3 spans a 4D subspace of $\mathbb{P}^8$. This can be solved linearly, calculating a $\overline{F}_{23} = [e_{32}]_x P_3 P_2^+$ that most closely approximates the given F_{23}.

The similarity with our work is that we also use information from neighboring camera matrices and fundamental (essential) matrices to restore the missing camera parameters. However, the differences between this method and ours are twofold. First and foremost, we assume the intrinsic parameters of our cameras to be known. Therefore, we are dealing with essential matrices instead of fundamental matrices. Secondly, unlike Sinha *et al.*, our building block does not need to be a camera triplet. We can use information from two or more neighbors to find a solution.

The third step consists of applying bundle adjustment to the previously calculated results, usually by minimizing the error using the Levenberg-Marquardt algorithm. This procedure results in a projective reconstruction $\{P^i, X_j\}$ of the detected feature points $\{x_j^i\}$.

In case a metric calibration is needed (step four), a rectifying homography H can be constructed from auto-calibration constraints, to upgrade the reconstruction to a metric one $\{P^i H, H^{-1} X_j\}$. To attain this homography, we can choose between direct and stratified methods [9]. The direct auto-calibration methods involve computing the absolute conic or its image, whilst the stratified methods solve the reconstruction in two steps: first solving for the plane at infinity, then using this to solve for the absolute conic.

2.2 An Alternate Approach

Besides the works referenced above, one alternative algorithm stems from the following theorem.

Theorem 1 *[Hartley Zisserman [[9], p.385]] Given three compatible fundamental matrices F_{21}, F_{31} and F_{32} satisfying the non-collinearity condition, the three corresponding camera matrices P, P' and P'' are unique up to the choice of a 3D projective coordinate frame*

The first two camera matrices P and P' may be determined from the fundamental matrix F_{21}. The third camera matrix P'' can then be determined in the same projective frame as follows.

1. Select a set of matching points $x_i \leftrightarrow x_i'$ in the first two images, satisfying $x_i'^T F_{21} x_i = 0$, and use triangulation to determine the corresponding 3D points X_i.
2. Use epipolar transfer to determine the corresponding points x_i'' in the third image, using the fundamental matrices F_{31} and F_{32}.
3. Solve for the camera matrix P'' from the set of 3D-2D correspondences $X_i \leftrightarrow x_i''$.

This is essentially the approach of *triangulation* (determining the 3D points X_i) and *localization* (determining camera pose from 2D-3D correspondences, more commonly known as camera resectioning) in the work of Mantzel *et al.* [13]. It should be noted that they employ the described algorithm on essential matrices, rather than fundamental matrices. In order to ensure the validity of the computed camera matrix in step three, they also have the need to ensure the orthogonality of the rotation component. This leads to an iterative algorithm that alternates between optimization of the camera rotations and translations.

An essential issue of the described method consists of the fact that it does not use all the constraints available. We chose to apply a different approach, making direct use of the available information in the form of the intrinsic camera calibration, reducing the available degrees of freedom.

Another issue of this approach consists of the cumulative error that is introduced at every stage of the algorithm. Even though the approach can be extended to multiple cameras, there is still the need at every stage to remove outliers

Algorithm 1 Overview of our Central Algorithm

CAMERA MOVEMENT DETECTION

Continuous loop:

1. Initialize new views since last iteration:
   (a) Calculate the background image.
   (b) Determine primary and secondary hotspots.
2. For each view that currently is not being recalibrated:
   (a) Cross-correlate patches around primary hotspots.
   (b) If number of good patches lies below threshold:
      i. Cross-correlate patches around secondary hotspots.
      ii. If number of good patches lies below threshold, recalibrate view.

CAMERA RECALIBRATION

1. Find N nearest neighbor cameras. Assign index 0 to the moved camera.
2. For each neighboring camera i:
   (a) Compute the essential matrix E_{0i}.
   (b) Compute the canonical camera pair $P_i^{L_i} = [I|0]$ and $P_0^{L_i} = [R_0^{L_i}|t_0^{L_i}]$.
3. Compute the mean rotation matrix R_0^W
4. Using the baseline estimates connecting t_i^W with t_0^W, compute the translation vector t_0^W.

in the feature point matches. The accuracy of the camera resectioning in the third stage is also very dependent on the success of the previous stages (3D reconstruction and point-transfer), and has the risk of becoming inaccurate in case of a low amount of point correspondences.
Our approach needs to only perform a single feature matching step, and only in image space. This reduces the chance of introducing unwanted errors to a minimum.

3 The Algorithm

We will now explain the basic building blocks of our method, together with our assumptions about the host system. Next we shall describe the two main algorithms of our method: movement detection and recalibration. An overview of our method can be found in algorithm listing 1.

3.1 Notation

In this section the basic principles and notations used in this paper are introduced. Unless noted otherwise, our notation is analogous to the one used in Hartley's work [9]. Projective geometry and homogeneous coordinates are used.

3.1.1 Cameras

The following equation is used to describe the perspective projection of the scene onto the images

$$m \propto PM \tag{1}$$

where P is a 3×4 projection matrix describing the perspective projection, and $M = [X, Y, Z, 1]^T$ and $m = [x, y, 1]^T$ are vectors containing the homogeneous coordinates of the world points respectively image points. The symbol $\propto$ indicates equality up to a scale factor.

In the metric case, the matrix factorizes as

$$P = K[R|t] \tag{2}$$

where the 3×3 matrix K encodes the intrinsic parameters of the camera, and (R, t) denotes a rigid transformation which indicate the position and orientation of the camera. If the calibration matrix K is known, then we may apply its inverse to the point m to obtain $\widehat{m} = [u, v, 1]^T = K^{-1}m$. We shall refer to these coordinates as *normalized coordinates*. The camera matrix $K^{-1}P = [R|t]$ is called a *normalized camera matrix*. During the rest of this paper, when we refer to camera matrices, we assume them all to be of this form.

3.1.2 Coordinate Frames

During the rest of the paper, we shall refer to the Euclidean coordinate frame of the previous calibration as the *world coordinate frame* W, e.g. the camera matrix of the first camera in this coordinate frame shall be denoted by P_1^W.

The coordinate frames derived from essential matrices are referred to as *local*. For example $P_1^{L_2}$ denotes the matrix of the first camera in the local coordinate frame of the second camera.

3.1.3 Cross Product

If $a = (a_1, a_2, a_3)^T$ is a 3-vector, then one defines a corresponding skew-symmetric matrix as follows:

$$[a]_\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \tag{3}$$

The cross product of two 3-vectors $a \times b$ is then related to skew-symmetric matrices according to:

$$a \times b = [a]_\times b = \left(a^T [b]_\times\right)^T \tag{4}$$

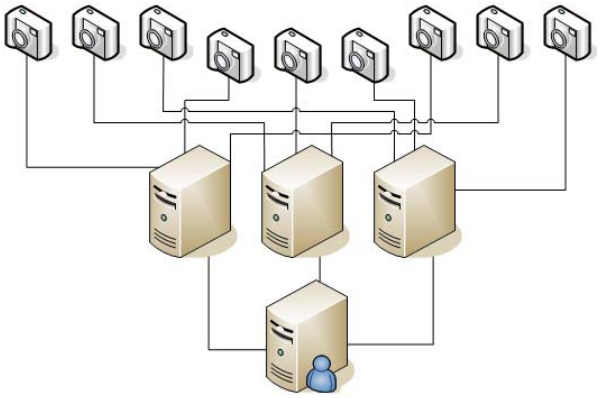

Figure 1. We define our algorithm for a centralized network structure, in which its system has access to all the nodes containing cameras.

3.2 System Assumptions

We define our algorithm for a centralized network structure. More specifically, we make the following assumptions: (1) the network is synchronized and (2) the station on which the algorithm is performed, has access to the neighboring cameras of the moved camera and their synchronized images. Thus we assume the algorithm to be running on a system that has access to all the nodes containing cameras.

The algorithm can also be applied to a distributed camera network, running an instance of the algorithm at each node, assuming that the conditions mentioned above are met.

Network delays have little to no impact on the system: as long as we have a means to ensure synchronous image pairs between two cameras, all requirements for essential matrix computation are met. The actual time between each acquired image pair is irrelevant. We end acquisition when we have enough point correspondences to commence computation.

3.3 Movement Detection

The goal of our system is to ensure that all cameras in the network remain calibrated. Therefore we need to detect when a camera has 'moved': when its known external calibration is no longer valid. This movement detection algorithm loops continuously during the entire application, and can be divided into two parts.

3.3.1 Preprocessing

The preprocessing step is called at each iteration in order to initialize new views, corresponding to cameras previously unknown to the network. When a new camera is introduced to a running application, we need to prepare it for calibration and recalibration.

First, we estimate a background from a predefined number of frames from the new view. There are several approaches to this estimation, and some are better suited for our purposes than others, e.g. we observed that applying a median filter on the frameset results in unnecessary sharp transitions in areas where the background is only exposed during a very limited time. This has undesirable consequences for the rest of our algorithm. Instead, we employed an averaging filter. It is not unlikely that more sophisticated approaches can be devised to improve the initial conditioning of the detection phase.

Our next step is to detect two layers of *hotspots* (Figure 2). We define a hotspot as an interesting feature that remains immobile during the entire recording process. This could be in any form, varying from special-purpose markers in the scene to distinctive features in the background. For the rest of the paper, we have chosen the latter approach.

As we are not interested in advanced aspects of feature detection (scale-space, affine invariance, etc...), but only in their *cornerness*, we have chosen to apply feature detection as done by Shi and Tomasi [15]. This cornerness allows us to impose a rank-order on the feature set. From these detected features, we sample two subsets which we will label the *primary* and *secondary hotspots*. The primary set will contain a small set of features, which lie at a large distance from each other, whilst the secondary set will be larger and denser. It is possible that a feature exists in both the primary as the secondary hotspot set: they are completely independent. We add a predefined number of features to each set, starting with the best features (highest minor eigenvalues, see Shi and Tomasi [15] for more details) and iteratively adding new features to the respective sets that are not too close to the features already added.

In case of a large change in the scene condition (e.g. completely different lighting conditions), the background needs to be refreshed, and the hotspots need to be recalculated. As this is a computationally expensive operation, it should only occur when, for *every* camera in the network, a large portion of the calculated background is different from the current frame. As an alternative, we can simply let the user trigger the background recomputation when the scene contents are altered.

3.3.2 Detection

For each view that currently is not being recalibrated, we perform a double checking procedure.

First, we verify that each of the primary hotspots is still

Figure 2. Given a set of detected features with an established rank-order, we can extract two subsets of with respectively high (red markers) and low (green markers) intra-element distances, which we label the primary and secondary hotspots.

in place. For a hotspot located at pixel (x, y), we perform a normalized cross-correlation between the $n \times n$ windows centered around (x, y) in both the current frame and the previously calculated background. If the computed value is below a certain threshold, this indicates that either the camera has moved, or the feature point has been occluded. When a predefined number of primary hotspots has been validated, we move on to the next view. If we reach the end of the list, this means not enough 'good' hotspots have been found, and we move on to the secondary layer of hotspots.

The presence of a secondary layer of hotspots is necessary to avoid unneeded recalibration attempts. We repeat the algorithm mentioned above for this additional set of feature points. If it returns a number of validated hotspots below a predefined threshold, we assume the camera needs to be recalibrated.

While our primary check is a quick process, slightly more time is needed for the secondary hotspots ($\approx 10^{-4}$ to 10^{-3} sec.), but this amount is still significantly less than the time-consuming feature detection and matching performed in the recalibration phase ($\approx$ a second per view). This approach reduces computation time to a level where the detection algorithm can work in parallel to other real-time algorithms, where overtaxing the processor would be unacceptable.

3.4 Recalibration

When camera movement is detected, the moved camera is flagged as inactive until the recalibration process is completed. The first step in the recalibration algorithm is to find the N nearest neighbors of the moved camera, with respect to its previous location and orientation. If the cameras are part of nodes equipped with radio sensors and transmitters, the *current* nearest neighbors can be determined from broadcast intervals instead.

Once we have determined these neighbors, we calculate the essential matrices between the moved camera and each of its neighbors. We employ a standard robust algorithm (appendix A)[9], based on RANSAC [4] and the normalized eight-point algorithm [8]. On these matrices, we apply the following theorem:

Theorem 2 *[Hartley Zisserman [[9], p.259]] For a given essential matrix $E = Udiag(1, 1, 0)V^T$, and first camera matrix $P = [I|0]$, there are four possible choices for the second camera matrix P', namely*

$$P' = \begin{array}{lll} [UWV^T|+u_3] & or & [UWV^T|-u_3] \quad or \\ [UW^TV^T|+u_3] & or & [UW^TV^T|-u_3] \end{array}$$

Testing with a single point to determine if it is in front of both cameras is sufficient to decide between the four different solutions for the camera matrix P'. Since this information is already available from our initial essential matrix estimation, we now have a metric local coordinate frame for each essential matrix.

3.4.1 Comparing Coordinate Frames

We will now compare the projections of a scene point $M = [X\ Y\ Z\ 1]^T$ onto a pair of cameras, both in world and local frame coordinates. The normalized image coordinates are identical in both (metric) coordinate frames, thus they are used for comparison.

The world coordinates are our reference frame, hence we dub them to be exact. This gives us the following equations:

$$\begin{array}{rcll} [u_0\ v_0\ 1]^T & = & [R_0^W|t_0^W] & [X^W\ Y^W\ Z^W\ 1]^T \\ [u_1\ v_1\ 1]^T & = & [R_1^W|t_1^W] & [X^W\ Y^W\ Z^W\ 1]^T \end{array} \quad (5)$$

Alternatively, the local coordinate frame is known only up to scale. As we know it to be metric as well, the camera matrices are of the following form:

$$\begin{array}{rcll} [u_0\ v_0\ 1]^T & = & [\pm R_0^L|\lambda t_0^L] & [X^L\ Y^L\ Z^L\ 1]^T \\ [u_1\ v_1\ 1]^T & = & [I|0] & [X^L\ Y^L\ Z^L\ 1]^T \end{array} \quad (6)$$

Putting these equations together gives us:

$$\begin{aligned} M^L &= \begin{bmatrix} I & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} R_1^W & t_1^W \\ 0 & 1 \end{bmatrix} M^W \\ M^L &= \begin{bmatrix} \pm R_0^L & \lambda t_0^L \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} R_0^W & t_0^W \\ 0 & 1 \end{bmatrix} M^W \end{aligned} \tag{7}$$

This provides us with two equations which define the rigid transformation (R, t):

$$R_0^W = R_0^L \, R_1^W \tag{8}$$

$$t_0^W = R_0^L \, t_1^W + \lambda t_0^L \tag{9}$$

3.4.2 Computing the Rotation Matrix

Equation (8) provides us with N estimates of the moved cameras orientation. Using these approximate solutions, we then compute an average rotation.

There are several approaches to this computation. Very often the barycenter of quaternions or matrices that represent the rotations are used as an estimate of the mean. One could point out that these methods neglect that rotations belong to a non-linear manifold, and that in order to obtain proper rotations, renormalization or orthogonalization must be applied. However, Gramkow [7] showed that using the simpler linear methods gives us a very good approximation of the non-linear average. Therefore, we employed the barycenter of rotation matrices as our method of choice.

The eigenvalues of an orthogonal matrix R_{ort} are of the form $(1, e^{i\theta}, e^{-i\theta})$, where the eigenvector corresponding to the unit eigenvalue represents the rotation axis [6]. Based on this, the mean rotation is defined by:

$$\overline{R} = \arg\min_{R_{ort}} \sum_i \theta^2 (R^{-1} R_i) \tag{10}$$

$$\approx \arg\max_{R_{ort}} tr\left(R^{-1} \sum_i R_i \right) \tag{11}$$

The solution to the latter optimization problem is actually the core of the 3D-3D pose problem, which has been solved by Horn [10], Arun *et al.* [1] and Umeyama [21]. It is most easily obtained by Singular Value Decomposition (SVD), $\sum_i R_i = UWV^T$, where the singular values are ordered in descending order. Introducing the matrix $S = diag(1, 1, det(U)det(V^T))$, the mean rotation in the above problem is simply given by

$$\overline{R} = USV^T \tag{12}$$

One preprocessing step has been omitted in the previous computation. As we have two valid possibilities for each $R_0^{W_i}$ (each rotation matrix and their inverse), we have to designate one orientation as the dominant orientation. To achieve this, we first compute $R_0^{W_1}$. Next, we apply the following transformation for all remaining $R_0^{W_i}$, in order to make sure all orientations are located within the same hemisphere:

$$\forall i \in [2, N] : R_0^{W_i} = \arg \min_{R \in \{\pm R_0^{W_i}\}} \|R_0^{W_1} - R\|^2 \tag{13}$$

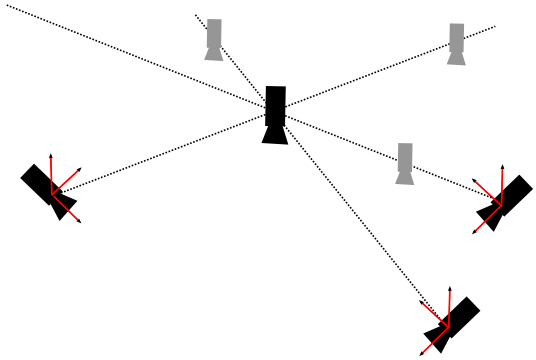

Figure 3. The local coordinate frames, obtained from essential matrices between the moved camera and its previously neighboring cameras, provide us with a baseline estimate (fig: dotted lines) for each camera pair. These estimates should have a common intersection point, reducing the problem to finding a point X with a minimized distance to each baseline. This will be the location of the new camera center.

3.4.3 Computing the Translation

Each local coordinate frame defines the new position of the moved camera up to scale (as depicted in Figure 3). This translates into equation (9), which defines the baseline connecting the camera centers of the camera pair in the world coordinate frame. Theoretically, the moved camera should be located at the intersection of all these lines. In practice, this boils down to a least-squares minimization problem, in which we minimize the 3D Euclidean point-line distance.

The distance between the line parameterized by eq.9 and a point X is defined as:

$$d = \frac{\|t_0^L \times (R_0^L \, t_1^W - X)\|}{\|t_0^L\|} \tag{14}$$

$$= \left\| \left(\frac{-[t_0^L]_\times}{\|t_0^L\|} \right) X - \left(-\frac{[t_0^L]_\times R_0^L \, t_1^W}{\|t_0^L\|} \right) \right\| \tag{15}$$

Although this definition provides us with three linear equations per neighboring camera, only two of them are linearly

independent. The reason for this is that $[t_0^L]_\times$ is a rank two matrix.

Combining the rows of m lines gives us a minimization problem of the form $\|Ax - b\|$, where A is a $2m \times 3$ matrix and b is a $2m$ vector. We solve this problem using the normal equations $A^T Ax = A^T b$. If $A^T A$ is invertible, then the solution is $x = (A^T A)^{-1} A^T b$.

4 Results

We implemented our system on a clustered network of workstations (2.8GHz Xeons, 2Gb RAM), connected by firewire to four Point Grey Flea cameras at each node, counting 20 cameras in total. The recalibration algorithm was running on one of the nodes, acquiring images trough network connections when needed. Camera calibration information was stored on the respective workstations.

Uniformly colored backgrounds contain little to no useful intensity information. This is a common problem for feature matching algorithms, and our movement detection is no exception to this rule. When we employ feature point detection as the basis for our hotspots, movement detection is dependent on a sufficient amount of background texture.

A possible work-around to this problem is to provide the scene with markers, which will serve as traceable hotspots. In a dense camera networks, these markers could be assigned to the cameras themselves, allowing the cameras to track each other.

In practice, we have observed the algorithm to give accurate results, comparable to those of state-of-the-art (de)centralized full-network recalibration algorithms. Using the newly computed camera matrix, the essential matrices associated with its neighbors resulted in a reprojection error of less than a pixel.

Computation times for the detection phase of our method were in the order of 10^{-4} to 10^{-3} seconds per iteration, making it attractive for real-time applications.

Recalibration times were also severely reduced when compared to full system recalibrations, even though there is still the matter of the time consuming SIFT detector. The time needed to detect and describe SIFT features in each camera frame proves to be the main bottleneck of our system ($\approx$ a second per view). In comparison, we also implemented the algorithm for small-baseline setups, using a normalized cross-correlation of RGB intensities as our matching criterium. This resulted in a significant reduction in computation time, often to less than a second. For unknown scene configurations however a wide-baseline setup must be assumed, and SIFT is currently still the state-of-the-art detector/descriptor available. Because we are aiming for real-time applications, we are currently investigating the use of SURF. We believe this should provide us with speed improvements comparable to small-baseline implementation we mentioned above.

A final note should be made for practical purposes. If the cameras are very close to the recorded scene, it is not uncommon for an actor to completely occlude the background. In order to avoid unnecessary recalibration attempts, it is recommended to let the camera perform an additional detection phase, a predefined amount of time after the first 'movement' was detected. This way, a passing actor will not trigger the recalibration algorithm.

5 Conclusions

We have demonstrated a working algorithm which successfully assures the robust calibration of a camera network, without the need for a full system recalibration after the event of (un)intended camera displacement. The algorithm detects when the extrinsic parameters of a camera are no longer valid, and reintegrates the displaced camera into the previously calibrated camera network.

This detection is done by the use of *hotspots*, traceable features in the background image. A sparse and a dense layer of such features are matched with the current frame, in order to establish if they are still present and located at the same pixel coordinates. If a large enough quantity of these hotspots remain in place, the camera is stable. Otherwise we commence recalibration of the displaced camera.

Recalibration is achieved only using image point correspondences, without the need to compute the underlying 3D structure. We compute essential matrices from the displaced camera to its neighboring cameras, which provide us with local coordinate estimates for each camera pair. These canonical pairs are related to the real world coordinates, up to a similarity transformation. From these estimates, a mean rotation and translation is deduced in the coordinate framework of the previous full system calibration.

It is important to note that our calibration algorithm is based on fitting camera pair estimates, rather than fitting 2D-3D correspondences. This means our computations have a lower dimensionality (two rotations and translations instead of a large set of point correspondences), and avoid the unneeded step of computing the 3D estimates.

The results of our algorithm are comparable to those of state-of-the-art (de)centralized full-network recalibration algorithms.

6 Acknowledgments

The authors acknowledge financial support on a structural basis from the European Regional Development Fund (ERDF) and the Flemish Government. Part of this research was funded by the BOF-projectfund of Hasselt University. Furthermore we would like to thank our colleagues for their help and inspiration.

Appendix A: Robust Essential Matrix Computation

1. **Feature point detection:** Compute feature points in M views for N frames.
2. **Putative correspondences:** Compute a set of feature points matches based on the SIFT detector/descriptor.
3. **RANSAC robust estimation:** Until a certain threshold has been reached, use random samples of feature point matches to find an essential matrix with E a large support of inliers.
4. **Non-linear estimation:** Re-estimate E from all correspondences classified as inliers by minimizing a cost function using the Levenberg-Marquardt algorithm.
5. **Guided matching:** Further interest point correspondences are now determined using the estimated E.

The last two steps are iterated until the number of correspondences is stable.

References

[1] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of 2 3-D point sets. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, 9(5):698–700, 1987.

[2] P. T. Baker and Y. Aloimonos. Calibration of a multicamera network. *Conference on Computer Vision and Pattern Recognition Workshop*, 07:72, 2003.

[3] H. Bay, T. Tuytelaars, and L. J. V. Gool. SURF: Speeded up robust features. In *Proceedings of the 9th European Conference on Computer Vision*. Springer-Verlag, may 2006.

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

[5] A. W. Fitzgibbon and A. Zisserman. Automatic camera recovery for closed or open image sequences. In *Proceedings of the 5th European Conference on Computer Vision-Volume I (ECCV '98)*, pages 311–326, London, UK, 1998. Springer-Verlag.

[6] G. H. Golub and C. F. V. Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[7] C. Gramkow. On averaging rotations. *Journal of Mathematical Imaging and Vision*, 15(1-2):7–16, 2001.

[8] R. I. Hartley. In defense of the eight-point algorithm. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, 19(6):580–593, 1997.

[9] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[10] B. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society A*, 4(4):629–642, April 1987.

[11] D. Jacobs. Linear fitting with missing data: Applications to structure-from-motion and to characterizing intensity images. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, page 206, Washington, DC, USA, 1997. IEEE Computer Society.

[12] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[13] W. Mantzel, H. Choi, and R. Baraniuk. Distributed Camera Network Localization. In *Asilomar Conference on Signals, Systems, and Computers*, volume 2, Pacific Grove, CA, November 2004.

[14] D. Martinec and T. Pajdla. Structure from many perspective images with occlusions. In *Proceedings of the 7th European Conference on Computer Vision-Part II (ECCV '02)*, pages 355–369, London, UK, 2002. Springer-Verlag.

[15] J. Shi and C. Tomasi. Good features to track. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593 – 600, 1994.

[16] S. N. Sinha, M. Pollefeys, and L. McMillan. Camera network calibration from dynamic silhouettes. *Proceedings of the 2004 Conference on Computer Vision and Pattern Recognition (CVPR'04)*, 01:195–202, 2004.

[17] P. F. Sturm and B. Triggs. A factorization based algorithm for multi-image projective structure and motion. In *Proceedings of the 4th European Conference on Computer Vision-Volume II (ECCV '96)*, pages 709–720, London, UK, 1996. Springer-Verlag.

[18] T. Svoboda, H. Hug, and L. J. V. Gool. Viroom - low cost synchronized multicamera system and its self-calibration. In *Proceedings of the 24th DAGM Symposium on Pattern Recognition*, pages 515–522, London, UK, 2002. Springer-Verlag.

[19] T. Svoboda, D. Martinec, and T. Pajdla. A convenient multicamera self-calibration for virtual environments. *PRESENCE: Teleoperators and Virtual Environments*, 14(4):407–422, August 2005.

[20] R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal of Robotics and Automation*, 3(4):323–344, 1987.

[21] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–380, 1991.

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