Estimating Physical Camera Parameters based on Multi-Sprite Motion Estimation

Dirk Farin[a] and Peter H.N. de With[a,b]

[a]Eindhoven University of Technology, PO Box 513, 5600 MB, The Netherlands
[b]LogicaCMG, PO Box 7089, 5600 JB Eindhoven, The Netherlands
d.s.farin@tue.nl

ABSTRACT

Global-motion estimation algorithms as they are employed in the MPEG-4 or H.264 video coding standards describe motion with a set of abstract parameters. These parameters model the camera motion, but they cannot be directly related to a physical meaning like rotation angles or the focal-length. We present a two step algorithm to factorize these abstract parameters into physically meaningful operations. The first step applies a fast linear estimation method. In an optional second step, these parameters can be refined with a non-linear optimization algorithm. The attractivity of our algorithm is its combination with the multi-sprite concept that allows for unrestricted rotational camera motion, including varying of focal-lengths. We present results for several sequences, including the well-known *stefan* sequence, which can only be processed with the multi-sprite approach.

Keywords: camera motion estimation, camera calibration, 3-D reconstruction.

1. INTRODUCTION

In current video-coding algorithms like MPEG-4 or H.264, global camera-motion is described as a small set of parameters. The most frequently applied model is the projective motion model with eight parameters. This model can describe the image motion that results from a rotating camera. For this reason, the projective motion model is often applied to assemble images of the input video into so-called background *sprites*. By transmitting only these static background images and the camera motion parameters to the decoder, it is possible to reconstruct the sequence showing the scene background with the moving camera.

The parameters of the projective motion model describe the camera motion in an abstract way without a direct correspondence to physically meaningful operations. However, for some applications, it is advantageous or even required to describe the camera motion in terms of rotation angles or zoom (change of focal-length). One of these applications is content analysis, where the camera motion can be used as a feature to help in analysing the video content. For example, a zoom-in operation should direct the attention of the viewer to a specific detail in the scene, where the important object is usually located at the center of the zoom.

While a qualitative analysis of camera motion can be sufficient for content analysis, there are other applications which require parameters with a high accuracy. An example is a augmented reality application, where virtual 3-D object should be added to the recorded video such that they are seamlessly integrated. This can only be achieved if the view onto the synthetic 3-D object is synchonized with the current camera view. For the camera control, the current camera rotation angles and the focal-lengths have to be known. Another application is the creation of cylindrical or spherical panoramic images[1] from video sequences, where especially the focal-length of the camera needs to be known. These parameters are usually not known and have to be estimated from the video sequence itself.

In this paper, we propose an algorithm that factorizes projective motion parameters into a set of physical parameters. More clearly, it takes a sequence of projective motion parameters as input and generates a corresponding sequence of camera rotation angles and focal-lengths for each of the input frames. What makes our algorithm different from previous approaches is its ability to support arbitrary camera rotations. It has been shown earlier[2] that the projective motion model cannot describe camera rotations of more than 180 degrees. As a solution, we combined the algorithm with a multi-sprite partitioning of the input sequence such that each partition does not violate the 180 degrees limitation. The parameter estimation is carried out in two steps, where

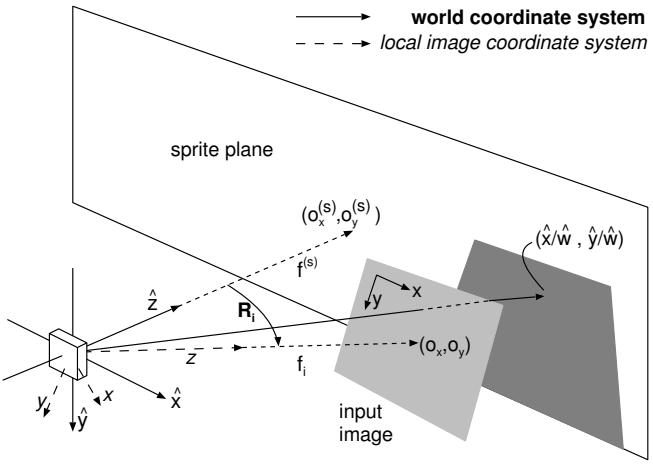


Figure 1. The rotating camera is located at the origin of the world coordinate system. The sprite plane is assumed to be orthogonal to the z-axis. Input images are at a distance to the origin that is equal to the focal-length when the image was taken. A point $(x/w, y/w)$ on the image is projected onto the sprite position $(\hat{x}/\hat{w}, \hat{y}/\hat{w})$.

the first is a fast linear calibration algorithm based on the *image of the absolute conic*. While the accuracy of this first step can be sufficient for content analysis applications, we also describe an optional refinement step. This refinement step uses a non-linear optimization algorithm to yield an increased accuracy of the obtained parameters.

The paper is organized as follows. We begin with a brief introduction to camera motion estimation and we show the necessity of a multi-sprite partitioning in Section 2. Section 3 describes the linear camera calibration algorithm for the single sprite case, which is generalized to the multi-sprite case in Section 4. The non-linear calibration is described in Section 5. In Section 6, we show how a rotation matrix can be factorized into elementary Euler angles. The paper closes with some example results in Section 7 and conclusions.

2. CAMERA MOTION ESTIMATION

2.1. Geometry of Background Image Generation

We start with a brief review of the geometry of rotational cameras and global motion estimation. Let us define the 3-D world coordinate system such that the camera is located at its origin. The camera captures a number of images i with different rotations $\mathbf{R}_i$ and focal-lengths f_i. The optical axis of the camera intersects the image plane at the principal point o_x, o_y. Focal length and principal point are collected in an intrinsic camera parameters matrix

$$\mathbf{K}_i = \begin{bmatrix} f_i & 0 & o_x \\ 0 & f_i & o_y \\ 0 & 0 & 1 \end{bmatrix}. \quad \text{Or, for the general case,} \quad \mathbf{K}_i = \begin{bmatrix} f_i & \tau & o_x \\ 0 & \eta f_i & o_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{1}$$

where η denotes the pixel aspect ratio and τ is the image skew. However, for typical CCD cameras, we can assume zero skew $\tau = 0$ and square pixels $\eta = 1$, so that we take these parameters as fixed.

Using homogeneous coordinates $\mathbf{p} = (x, y, w)^\top$ for 2-D image positions, we can obtain the projection of a 3-D point onto the image as $\mathbf{p} = \mathbf{K}_i \cdot (x, y, z)^\top$. Conversely, we can use the inverse $\mathbf{K}_i^{-1}$ to map 2-D points back to 3-D direction vectors. Concatenating these transforms with an intermediate 3-D rotation gives the transformation

between two images or, similarly, between an input image and the background sprite, as

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = \underbrace{\begin{pmatrix} f_i & 0 & o_x \\ 0 & f_i & o_y \\ 0 & 0 & 1 \end{pmatrix}}_{\substack{\text{intrinsic camera} \\ \text{parameters: } \mathbf{K}_i}} \underbrace{\begin{pmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{pmatrix}}_{\substack{\text{extrinsic parameters} \\ \text{(camera rotation): } \mathbf{R}_i}} \underbrace{\begin{pmatrix} 1/f^{(s)} & 0 & -o_x^{(s)}/f^{(s)} \\ 0 & 1/f^{(s)} & -o_y^{(s)}/f^{(s)} \\ 0 & 0 & 1 \end{pmatrix}}_{\substack{\text{intrinsic cam-} \\ \text{era parameters:} \\ (\mathbf{K}^{(s)})^{-1}}} \begin{pmatrix} \hat{x} \\ \hat{y} \\ \hat{w} \end{pmatrix} = \mathbf{H}_i \begin{pmatrix} \hat{x} \\ \hat{y} \\ \hat{w} \end{pmatrix}. \quad (2)$$

This is also illustrated in Figure 1. Multiplying the intrinsic and extrinsic transformation matrices together, we obtain the combined matrix $\mathbf{H}_i$, describing the projection of coordinates on the background plane onto image i. Writing with inhomogeneous coordinates and normalizing $\mathbf{H}_i$ to $h_{22} = 1$ gives the well known projective motion model with eight parameters

$$x = \frac{h_{00}\hat{x} + h_{01}\hat{y} + h_{02}}{h_{20}\hat{x} + h_{21}\hat{y} + 1}, \quad y = \frac{h_{10}\hat{x} + h_{11}\hat{y} + h_{12}}{h_{20}\hat{x} + h_{21}\hat{y} + 1}. \quad (3)$$

Taking the background-to-image transformations $\mathbf{H}_i, \mathbf{H}_k$ for two images i,k, we can obtain the transformation from image k to i by first mapping the point of image k onto the background and then mapping it back onto image i. We denote this inter-image transform as $\mathbf{H}_{i;k} = \mathbf{H}_i \mathbf{H}_k^{-1}$.

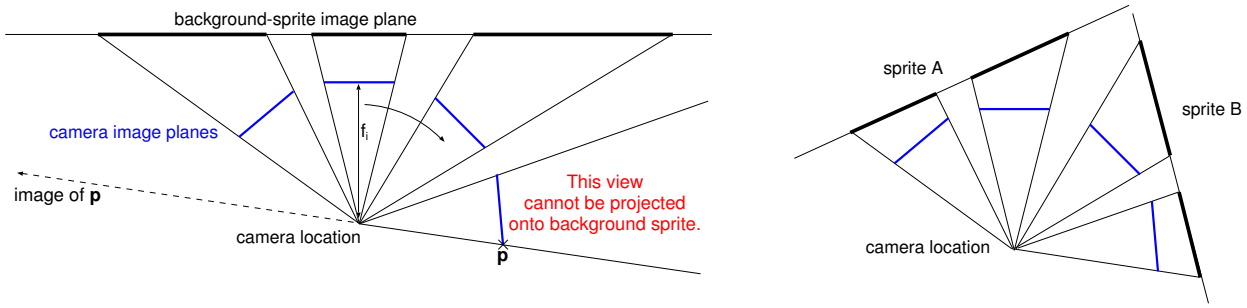
2.2. Global Motion Estimation

The camera calibration algorithm that we are going to describe in the next section requires accurate motion parameters, which we compute with a two-step parametric motion estimator from the observed video sequence,[3] We start with a feature-based motion estimator to compute the motion $\mathbf{H}_{i+1,i}$ between successive frames i and $i+1$. After choosing a reference frame r for the sprite background and a corresponding anchor transformation $\mathbf{H}_r$, we can concatenate the inter-image transforms to obtain the transforms between the sprite and each of the input images. Let, for example, frame 4 be the reference frame, then we can get $\mathbf{H}_7$ as $\mathbf{H}_7 = \mathbf{H}_{7,6}\mathbf{H}_{6,5}\mathbf{H}_{5,4}\mathbf{H}_4$ while we have to take inverse transforms for frames $i < r$: $\mathbf{H}_1 = \mathbf{H}_{2,1}^{-1}\mathbf{H}_{3,2}^{-1}\mathbf{H}_{4,3}^{-1}\mathbf{H}_4$.

The disadvantage of this chaining of transformations is that small errors in the motion parameters will accumulate when the distance between the reference transform and the current input frame increases. For this reason, we apply a second motion estimation step which registers the input images to a common background sprite.

This refinement algorithm synthesizes a background sprite image by iteratively adding input frames to the background image. For each of these images, the motion parameters obtained from the feature-based estimator are refined by minimizing the difference image between the background sprite and the camera-motion compensated input frame. This results in *long-term* motion parameters, describing the transform between the input frames and the background sprite. Since all images are registered to the same sprite image, independent of the temporal distance, error accumulation is prevented. The result are sprite-to-image transformations $\mathbf{H}_i$ for all input images i.

When chaining the inter-sprite transformations, we have to be careful about the parameterization of the transformations. It is common practice to normalize the projective transformations $\mathbf{H} = \{h_{ik}\}$ to $h_{22} = 1$, and we also apply this normalization in our motion estimation algorithms. While this normalization is valid for small camera rotation angles, it imposes two problems. First, if the camera rotation angle is about 90 degrees, such that the horizont line crosses the origin of the coordinate system, we get transformations with $h_{22} = 0$. Consequently, the normalization to unity must fail. If the camera rotates further to larger angles, the sign of h_{22} becomes negative. The normalization is still applicable, but we lose the information about the sign, what again has consequences when we want to recover the camera rotation angles later. The simple but important consequence is that transformations should not be normalized to $h_{22} = 1$ when they are chained together to span larger fields of view. Note that it is still possible to carry out the inter-image motion estimation using normalized parameters, since the multi-sprite partitioning that will be introduced in the next section ensures that the transforms do not degenerate.



(a) The use of ordinary sprites limites the field of view to 180 degrees.

(b) Multi-sprites allow to cover arbitrary camera rotation angles.

Figure 2. (a) Top-view of projecting the input frames onto the background sprite plane. For large camera rotations, the projection ray does not intersect the sprite plane. Hence, only 180° field of view can be covered with one sprite. (b) Using several sprites allows to cover larger viewing angles.

2.3. Generalization to a Multi-Sprite Approach

In geometric terms, the transformation model that we used in the camera motion estimation is a plane-to-plane mapping. Thus, the background sprite image can be envisioned as the projection of the input images onto a plane. This is illustrated in Fig. 2(a), which shows a top-view of a camera that rotates around its vertical axis. If the camera rotates away from the frontal view position, a point will be reached, where image pixels cannot be projected onto the plane anymore (see, e.g., the pixel position **p**). In this situation, the image will be projected upside down on the opposite side of the background.

Consequently, a planar background sprite has the direct limitation that only 180° field of view can be represented in a single sprite image. In practice, the usable viewing angle is even smaller since the perspective deformation increases rapidly when the camera rotates away from its frontal view position. Because we make use of this planar background sprite to obtain accurate motion parameters, we are confronted with the limitation of the maximum camera rotation angle.

As a solution, we have proposed in[2] to partition the input sequence into several independent sub-sequences and generate independent sprites for each of them (Fig. 2(b)). In the original work, the intention was to minimize the size of the generated sprite images for increased coding efficiency, but we can also use the same principle to prevent the limitation on the allowed camera rotations. The result of this multi-sprite partitioning is a partitioning of the input sequence of length N into ranges

$$P = \{(1, p_2 - 1)\ ,\ (p_2, p_3 - 1)\ ,\ (p_3, p_4 - 1)\ ,\ \ldots\ ,\ (p_M, N)\}\,, \tag{4}$$

where p_k denotes the first frame used in sprite k ($k \in [1; M]$). Instead of computing motion parameters to a global reference sprite (which is not possible), we compute the motion parameters for each image relative to the sprite that it has been assigned to. We denote this transformation from sprite k to image i as $\mathbf{H}_i^{(k)}$ (Fig. 6).

3. LINEAR CAMERA CALIBRATION

The camera calibration is carried out in two steps, where the first is a linear calibration algorithm, providing an approximate solution. This solution is further refined in a second non-linear optimization. The linear calibration will be discussed in this section, while the non-linear optimization will be presented in the Section 5.

The linear calibration algorithm uses the method based on the transformation of the absolute conic.[4,5] The absolute conic Ω_∞ is defined as the points $(x, y, z, w)^\top$ satisfying

$$(x, y, z)\mathbf{I}(x, y, z)^\top = 0 \quad \text{and} \quad w = 0, \tag{5}$$

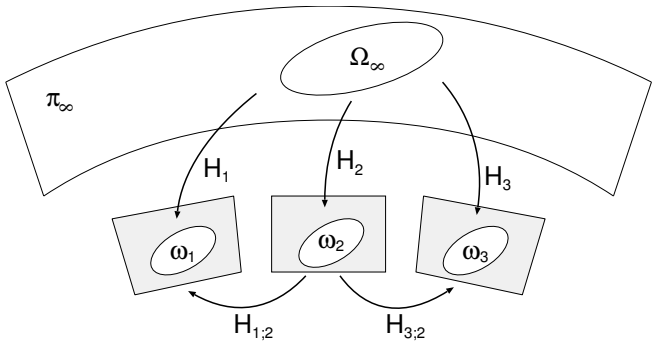


Figure 3. The absolute conic Ω_∞ is transformed to the input frames and shows there as the IAC $\boldsymbol{\omega}_i$.

with $\mathbf{I}$ being the 3×3 identity matrix. Hence, because of $w = 0$, the absolute conic lies in the plane at infinity $\boldsymbol{\pi}_\infty$ (see Fig. 3). Transformed with the camera transformation $\mathbf{H}_i = \mathbf{K}_i\mathbf{R}_i$ for view i, the image of the absolute conic (IAC) $\boldsymbol{\omega}^{(i)}$ is

$$\boldsymbol{\omega}^{(i)} = \mathbf{H_i}^{-\top}\mathbf{I}\mathbf{H_i}^{-1} = \mathbf{K_i}^{-\top}\mathbf{R_i}^{-\top}\mathbf{R_i}^{-1}\mathbf{K_i}^{-1} = \mathbf{K_i}^{-\top}\mathbf{K_i}^{-1}, \tag{6}$$

where the last equality holds since $\mathbf{R_i}^\top = \mathbf{R_i}^{-1}$. Assuming zero skew and square pixels, $\boldsymbol{\omega}^{(i)}$ has the form

$$\boldsymbol{\omega}^{(i)} = \begin{bmatrix} 1/f_i^2 & 0 & -o_x/f_i^2 \\ 0 & 1/f_i^2 & -o_y/f_i^2 \\ -o_x/f_i^2 & -o_y/f_i^2 & o_x^2/f_i^2 + o_y^2/f_i^2 + 1 \end{bmatrix}. \tag{7}$$

Consequently, zero skew leads to the constraint $\omega_{01}^{(i)} = 0$ and the square pixel assumption leads to $\omega_{00}^{(i)} = \omega_{11}^{(i)}$.

Now, let us consider several views. The constraints that we derived previously are true for each of the views. Let us select one of the views as the reference view r. By concatenating two transformations $\mathbf{H}_i$ and $\mathbf{H}_r$, we obtain the inter-image transforms $\mathbf{H}_{i;r}$, mapping pixels from view r to view i as $\mathbf{H}_{i;r} = \mathbf{H}_i\mathbf{H}_r^{-1}$. With these parameters, we can transform the IAC from the reference view to the other views according to

$$\boldsymbol{\omega}^{(i)} = \mathbf{H}_{i;r}^{-\top}\boldsymbol{\omega}^{(r)}\mathbf{H}_{i;r}^{-1}. \tag{8}$$

This allows us to express the IAC also in coordinates for the other views. Since the constraints for zero skew and square pixels must be fulfilled for all IACs $\boldsymbol{\omega}^{(i)}$, we can also impose the constraints in these views. The plan for the following is to stack these constraints into an equation system, from which we can subsequently estimate the parameters of $\boldsymbol{\omega}^{(r)}$. Because we express the constraints of all views in the coordinate system of view r, we have enough equations to solve for the parameters of $\boldsymbol{\omega}^{(r)}$. When we know $\boldsymbol{\omega}^{(r)}$, it is easy to obtain the intrinsic parameters using a Cholesky decomposition.

To exploit the constraints from all the views, we collect the parameters of $\boldsymbol{\omega}^{(r)}$ in a vector $\mathbf{c}$. Note that $\boldsymbol{\omega}^{(r)}$ is symmetric so that $\mathbf{c}$ only holds six parameters. Based on Equation (8), we can derive a linear equation, which expresses every component of $\boldsymbol{\omega}^{(i)}$ as a linear combination of the parameters of $\mathbf{c}$. Imposing the zero skew assumption $\omega_{01}^{(i)} = 0$ and the square pixel assumption $\omega_{00}^{(i)} - \omega_{11}^{(i)} = 0$, we get one constraining equation for each constraint. Stacking all the equations into a matrix $\mathbf{A}$, we get the overdetermined, homogeneous equation system $\mathbf{Ac} = \mathbf{0}$, which we can use to calculate $\mathbf{c}$. A least-squares solution for this equation system can be obtained by carrying out a Singular Value Decomposition and taking the singular vector that corresponds to the smallest singular value. The parameter vector $\mathbf{c}$ directly gives the IAC $\boldsymbol{\omega}^{(i)}$. Remember that $\boldsymbol{\omega}^{(i)} = \mathbf{K}^{-\top}\mathbf{K}^{-1}$, where $\mathbf{K}$ is an upper triangular matrix. To get $\mathbf{K}_i$ from $\boldsymbol{\omega}^{(i)}$, we carry out a Cholesky decomposition.*

Repeating the above algorithm to iterate the reference frame through each of the views gives us the intrinsic parameters $\mathbf{K}_i$ for all views. Once all the intrinsic parameters are known, we can obtain the camera rotation.

*The Cholesky decomposition $\boldsymbol{\omega} = \mathbf{K}\mathbf{K}^\top$ is often implemented to give lower triangular matrices $\mathbf{K}$. We use the trick to transpose the matrices along the diagonal ($\omega_{00} \leftrightarrow \omega_{22}, \omega_{01} \leftrightarrow \omega_{12}, \omega_{10} \leftrightarrow \omega_{21}$) before and after the decomposition to get a decomposition into upper triangular matrices.

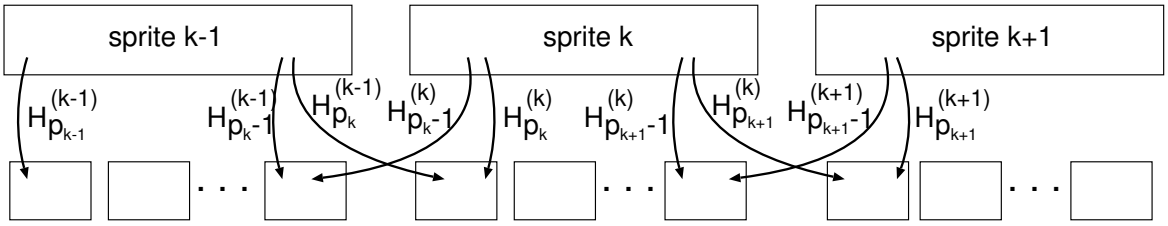


Figure 4. Transformations between sprites and input frames. Every image i is connected to its assigned sprite k the transform $\mathbf{H}_i^{(k)}$. The first and last images from each partition are also connected to the previous or next sprite, respectively.

We start with the transformation between two views i and k, given by

$$\mathbf{H}_{k;i} = \mathbf{H}_k \mathbf{H}_i^{-1} = \mathbf{K}_k \mathbf{R} \mathbf{K}_i^{-1}, \tag{9}$$

where $\mathbf{R}$ is the rotation between the two views. Since we know the homographies $\mathbf{H}_k, \mathbf{H}_i$ and the intrinsic parameter matrices, we can recover the rotation matrix as

$$\mathbf{R} = \mathbf{K}_k^{-1} \mathbf{H}_k \mathbf{H}_i^{-1} \mathbf{K}_i. \tag{10}$$

4. LINEAR CALIBRATION WITH MULTI-SPRITE MOTION ESTIMATION

The algorithm that we presented so far assumed that accurate inter-image transforms between all pairs of views are available. However, as described in the introduction, the chaining of inter-image transforms between successive frames leads to an accumulation of errors, which subsequently also leads to an inaccurate parameter estimation. The solution was that instead of chaining inter-image transforms, we determined long-term motion parameters relative to a global background sprite image. This provides high accuracy parameters between this background image and a current view. Inter-image transforms between a pair of views i,k can be obtained from these by just concatenating the two sprite transforms $\mathbf{H}_i \mathbf{H}_k^{-1}$. Consequently, there is less accumulation of errors even for transforms between distant views.

A consequence of using a global background sprite is that the supported camera motion is limited. Since the background sprite is a planar manifold, only a (theoretical) maximum of 180 degrees field of view can be covered. Practically, the maximum camera rotation angles are much lower. As a solution to this problem, we used the multi-sprite approach to partition the video sequence into sub-sequences and to synthesize a separate background sprite for each of these sub-sequences. This means that we cannot apply the above calibration algorithm directly, because the transforms of two views can be relative to different background sprites.

To obtain inter-image transformations for views that were assigned to different sprites, we have to determine the motion transform between successive sprites to connect them. In the original multi-sprite algorithm, motion parameters where only computed between the current view and the sprite to which they were assigned to. We extend this by computing also the transforms to the previous or successive sprites for the first and last frame of a sub-sequence, respectively (see Fig. 4). This means that for frame $p_k - 1$ (the last frame of sprite $k-1$), we compute the long-term motion parameters $\mathbf{H}_{p_k-1}^{(k-1)}$ to sprite $k-1$, but also the parameters $\mathbf{H}_{p_k-1}^{(k)}$ relative to the successive sprite k. Similarly, we also compute the motion parameters $\mathbf{H}_{p_k}^{(p_k-1)}$ to the previous sprite $k-1$ for the first frame p_k of a sprite k. These transforms allow to chain the transformations between sprites to allow for large rotation angles. As an example, we can get the transformation from a view i_1 that is assigned to sprite k to a view i_2 on sprite $k+1$ with

$$\mathbf{H}_{i_2}^{(k+1)} \left(\mathbf{H}_{p_{k+1}}^{(k+1)}\right)^{-1} \mathbf{H}_{p_{k+1}}^{(k)} \left(\mathbf{H}_{i_1}^{(k)}\right)^{-1}. \tag{11}$$

If more than one sprite boundary has to be crossed, more chaining transforms are required. However, note that the number of chained transforms is significantly less than when chaining transforms between successive images. Typically, we have only about five sprites, which usually cover up to several thousands of frames. Hence, the error accumulation is neglegible. Moreover, notice that the motion parameters have been obtained with the long-term motion estimator, such that the parameters have *a-priori* a higher accuracy.

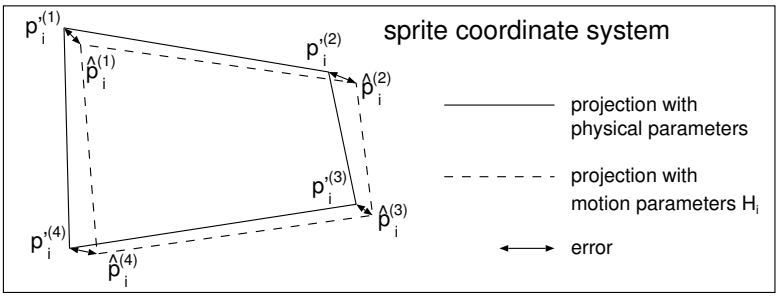


Figure 5. The corners of an input image are projected into the sprite coordinate system once with the global motion parameters, and once with the physical camera parameters. The Euclidean distance between the respective corner positions define the error function for the physical camera parameters.

5. NON-LINEAR CAMERA CALIBRATION

In this section, we apply a bundle-adjustment algorithm[6] to increase the accuracy of the camera parameters that we obtained with the linear calibration algorithm. The reason why the parameters obtained with the linear algorithm are not optimal is that we obtained them with a least-squares approximation over an algebraic error. The accuracy of the camera parameters can be increased by replacing this semantically weak error definition with a physically meaningful measure. Furthermore, the non-linear estimation algorithm has the advantage that it allows to integrate additional constraints, which are difficult to express in the linear parameter estimation.

We start again with the high accuracy motion parameters that we computed in the long-term motion estimation. Let us first consider the case in which we have only one sprite. For each view i, there is one set of motion parameters $\mathbf{H}_i$, mapping coordinates on the sprite to coordinates in image i. This transformation can be decomposed in a sequence of physically motivated transformations as

$$\mathbf{H}_i = \mathbf{K}_i \cdot \mathbf{R}_i \cdot {\mathbf{K}^{(s)}}^{-1}. \tag{12}$$

If the sprite is constructed based on an input view r as reference coordinate system, then $\mathbf{K}^{(s)}$ will equal the intrinsic parameters $\mathbf{K}_r$ of view r. However, since our sprite construction algorithm can change the focal-length for the sprite plane (see[2]), we use an independent set of parameters for the sprite projection.

The purpose of the camera calibration algorithm is to factorize the motion parameters $\mathbf{H}_i$ to the intrinsic parameters $\mathbf{K}_i$ and the rotations $\mathbf{R}_i$. Usually, it will not be possible to find an exact solution, so that a solution minimizing some error function is desired. We define the error as the Euclidean distance between the pixel positions on the sprite when they are projected with the abstract motion parameters, and their position when the physical parameters are used. Using the measured motion parameters $\mathbf{H}_i$, a pixel $\mathbf{p}$ on view i is projected to the sprite position

$$\hat{\mathbf{p}} = \mathbf{H}_i^{-1}\mathbf{p}. \tag{13}$$

With the estimated physical camera parameters, we project the same pixel to the sprite position

$$\mathbf{p}' = \mathbf{K}^{(s)}\mathbf{R}_i\mathbf{K}_i^{-1}\mathbf{p}. \tag{14}$$

This lets us define the estimation error $e_i(\mathbf{p})$ for view i and pixel position $\mathbf{p}$ as $e_i(\mathbf{p}) = d(\hat{\mathbf{p}}, \mathbf{p}')^2$, where $d(\cdot,\cdot)$ denotes Euclidean distance.

5.1. Parameterization

The parameters that we are going to estimate are the intrinsic camera parameters and the camera rotation. The intrinsic parameters for a camera comprise the variable focal length f_i, and the principal point o_x, o_y which is assumed unknown but constant during the observed sequence[†]. Additionally, we include parameters for the

[†]Note that the constant principal point constraint was not applied in the linear calibration algorithm.

projection into the sprite coordinate system, comprising the focal length $f^{(s)}$ and the principal point in the sprite $o_x^{(s)}, o_y^{(s)}$.

For the subsequent optimization, we need a suitable parameterization of the camera rotation. A camera rotation has three free parameters, but previously, we have expressed camera rotation with a 3×3 rotation matrix, comprising nine parameters. Compared to the three free parameters, this is an unnecessary over-parameterization. An alternative parameterization with Euler angles has only three parameters, but it shows singularities that lead to low numerical stability. Better numerical stability can be obtained with a parameterization using quaternions[7] $\mathbf{q} = (q_w, q_x, q_y, q_z)$. A quaternion $\mathbf{q}$ can be associated with a rotation, when $||\mathbf{q}|| = 1$. For these unit quaternions, a rotation matrix can be derived from the quaternion parameters as

$$\mathbf{R} = \begin{bmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_xq_y - 2q_wq_z & 2q_xq_z + 2q_wq_y \\ 2q_xq_y + 2q_wq_z & 1 - 2q_x^2 - 2q_z^2 & 2q_yq_z - 2q_wq_x \\ 2q_xq_z - 2q_wq_y & 2q_yq_z + 2q_wq_x & 1 - 2q_x^2 - 2q_y^2 \end{bmatrix}, \quad \text{where} \quad ||\mathbf{q}|| = 1. \tag{15}$$

Quaternions show no singularities and have only four parameters. With the unit norm constraint for rotations, the number of free parameters is reduced to three.

We collect all camera parameters into a parameter vector $\mathbf{x}$. In particular, each image has a quaternion for the rotation and the focal-length parameter. An exception is the first image, which does not have rotation parameters, since this image serves as the reference view. Hence, we get the parameter vector

$$\mathbf{x} = (\underbrace{f_1,\ \mathbf{q}_2, f_2,\ \ldots,\ \mathbf{q}_N, f_N,\ o_x, o_y}_{\text{image view parameters}}\ ,\ \underbrace{\mathbf{q}^{(s)}, f^{(s)},\ o_x^{(s)}, o_y^{(s)}}_{\text{sprite view parameters}})^\top. \tag{16}$$

With a vector of physical camera parameters $\mathbf{x}$, we can compute the projection of the four corners of each input image on the sprite. Let us collect the inhomogeneous coordinates of these projections of the image corners in a data vector

$$\mathbf{y} = (\underbrace{\underline{\mathbf{p}'}_1^{(1)}, \underline{\mathbf{p}'}_1^{(2)}, \underline{\mathbf{p}'}_1^{(3)}, \underline{\mathbf{p}'}_1^{(4)}}_{\text{first view}}, \underbrace{\underline{\mathbf{p}'}_2^{(1)}, \underline{\mathbf{p}'}_2^{(2)}, \underline{\mathbf{p}'}_2^{(3)}, \underline{\mathbf{p}'}_2^{(4)}}_{\text{second view}}, \ldots, \underbrace{\underline{\mathbf{p}'}_N^{(1)}, \underline{\mathbf{p}'}_N^{(2)}, \underline{\mathbf{p}'}_N^{(3)}, \underline{\mathbf{p}'}_N^{(4)}}_{\text{view } N})^\top, \tag{17}$$

where $\underline{\mathbf{p}}_i^{(c)}$ denotes the inhomogeneous coordinates of the projection of the c^{th} corner of the image in view i. We can also generate a measurement data vector $\hat{\mathbf{y}}$ of similar layout, for which we obtain the projections of the image corners from the motion parameters $\mathbf{H}_i$:

$$\hat{\mathbf{y}} = (\underbrace{\underline{\hat{\mathbf{p}}}_1^{(1)}, \underline{\hat{\mathbf{p}}}_1^{(2)}, \underline{\hat{\mathbf{p}}}_1^{(3)}, \underline{\hat{\mathbf{p}}}_1^{(4)}}_{\text{first view}}, \underbrace{\underline{\hat{\mathbf{p}}}_2^{(1)}, \underline{\hat{\mathbf{p}}}_2^{(2)}, \underline{\hat{\mathbf{p}}}_2^{(3)}, \underline{\hat{\mathbf{p}}}_2^{(4)}}_{\text{second view}}, \ldots, \underbrace{\underline{\hat{\mathbf{p}}}_N^{(1)}, \underline{\hat{\mathbf{p}}}_N^{(2)}, \underline{\hat{\mathbf{p}}}_N^{(3)}, \underline{\hat{\mathbf{p}}}_N^{(4)}}_{\text{view } N})^\top, \tag{18}$$

These three vectors $\mathbf{x}, \mathbf{y}$, and $\hat{\mathbf{y}}$ will be crucial in the optimization process. In that process, we will optimize the parameter vector $\mathbf{x}$ such that the sum of squared Euclidean distances between the projected corner coordinates $||\hat{\mathbf{y}} - \mathbf{y}||^2$ is minimized (see Fig. 5).

5.2. Generalizing to Multi-Sprites

Like in the linear calibration algorithm, we have to generalize the algorithm to allow for multiple sprites, if we want to support arbitrary camera rotations. To do so, we extend the parameter vector $\mathbf{x}$ with calibration parameters for each for the sprites, such that we get a vector of the following layout:

$$\mathbf{x} = (\underbrace{f_1,\ \mathbf{q}_2, f_2,\ \ldots,\ \mathbf{q}_N, f_N,\ o_x, o_y}_{\text{image view parameters}}\ ,\ \underbrace{\mathbf{q}_1^{(s)}, f_1^{(s)},\ o_{x;1}^{(s)}, o_{y;1}^{(s)},\ \ldots,\ \mathbf{q}_M^{(s)}, f_M^{(s)},\ o_{x;M}^{(s)}, o_{y;M}^{(s)}}_{\text{sprite view parameters}})^\top. \tag{19}$$

The involved parameters are depicted in Figure 6.

The measurement vector $\mathbf{y}$ is constructed similarly as before, except for two changes. First, the corners of all the images are not projected onto the same sprite. Instead, the corners of each image are projected into the coordinate system of the sprite that is assigned to the image. Second, the corners of the images that link two sprites together (e.g., image 2 in Figure 6) are projected onto both sprites. Consequently, the projected positions of these image corners each lead to two entries in the data vector.

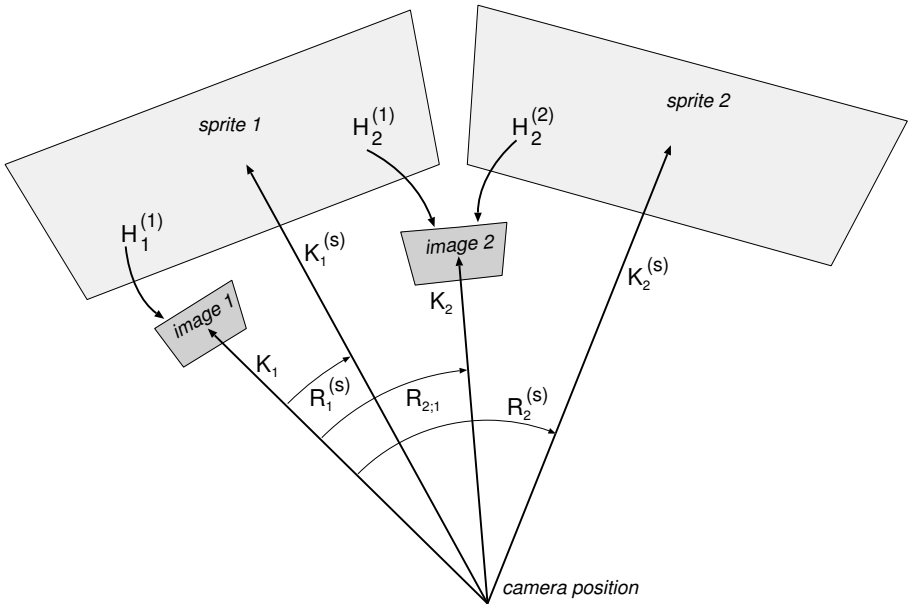


Figure 6. The transformations involved in the non-linear camera calibration algorithm for multi-sprite sequences.

5.3. Optimization Algorithm

For the optimization, we apply the Levenberg-Marquardt (LM) algorithm.[8] This algorithm is a combination of a steepest gradient descent algorithm and a Gauss-Newton method. It attempts to find the parameter vector $\mathbf{x}$, for which the data vector $\mathbf{y}$ is closest to the observed measurement vector $\hat{\mathbf{y}}$ in terms of minimizing $(\hat{\mathbf{y}}-\mathbf{y})^\top(\hat{\mathbf{y}}-\mathbf{y})$. Starting with a first estimate of the parameter vector $\mathbf{x}$, the LM algorithm iteratively updates the parameter vector, where a steepest gradient descent is carried out while the current estimate is far from the optimum. When the optimum is approached, the algorithm operates more like a Gauss-Newton algorithm. In each step, the LM algorithm updates the parameter vector $\mathbf{x}$ with an update $\boldsymbol{\delta}_\mathbf{x}$ such that

$$(\lambda\mathbf{I}+\mathbf{J})^\top((\lambda\mathbf{I}+\mathbf{J})\boldsymbol{\delta}_\mathbf{x}-(\hat{\mathbf{y}}-\mathbf{y}))=\mathbf{0}, \tag{20}$$

where $\mathbf{J}$ is the Jacobian matrix $\partial\mathbf{y}/\partial\mathbf{x}$. The parameter λ is controlled by the LM algorithm to switch between steepest descent and Gauss-Newton. To maintain the constraint that the quaternions have unit norm, they are normalized after each iteration of the LM algorithm. Since the camera parameters for a view i only have influence on the position of the four corner points of that view, the Jacobian matrix $\mathbf{J}$ is very sparse (see Fig. 7). Therefore, an optimized implementation[6,9] can be used to exploit this spare matrix structure for a lower computational complexity.

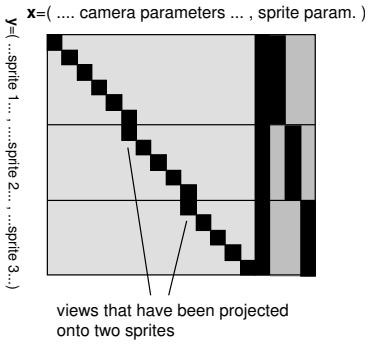


Figure 7. Non-zero entries of the Jacobian matrix $\mathbf{J}$.

6. RECOVERING ROTATION ANGLES

In the calibration algorithm, we have used rotation matrices and quaternions to describe camera rotation. However, while both approaches offer advantages for the numerical implementation, they cannot be interpreted intuitively. Hence, we desire to convert the parameterization to a sequence of elementary Euler angle rotations. According to *Euler's rotation theorem*, an arbitrary rotation can be decomposed into three successive rotations around predefined axes. Rotations are not commutative and hence, the order in which we perform the rotations is important. We chose to use this sequence of elementary rotations:

$$\mathbf{R}(\alpha,\beta,\gamma)=\mathbf{R}_\mathbf{y}(\beta)\mathbf{R}_\mathbf{x}(\alpha)\mathbf{R}_\mathbf{z}(\gamma), \tag{21}$$

since the obtained angles correspond well to the human intuition about rotations. In this equation $\mathbf{R_x}(\alpha)$ denotes rotation around the x-axis by an angle of α. The other rotation matrices stand for the y- and z-axes. Multiplying these elementary rotation matrices together, we obtain

$$\mathbf{R} = \begin{bmatrix} c_\beta c_\gamma - s_\alpha s_\beta s_\gamma & -c_\beta s_\gamma - s_\alpha s_\beta c_\gamma & -c_\alpha s_\beta \\ c_\alpha s_\gamma & c_\alpha c_\gamma & -s_\alpha \\ s_\beta c_\gamma + s_\alpha c_\beta s_\gamma & s_\alpha c_\beta c_\gamma - s_\beta s_\gamma & c_\alpha c_\beta \end{bmatrix}, \tag{22}$$

where we use the abbreviations $c_\alpha = \cos\alpha, s_\alpha = \sin\alpha$. Because the trigonometric functions are cyclic, we require additional restrictions to find a unique solution. A possible restriction is to assume that $-\pi/2 < \alpha < \pi/2$. This is a sensible assumption for a normal camera, since it limits the up-down rotation to $\pm 90°$. Now, α can be obtained directly from $r_{12} = -\sin\alpha$. Further, we get from the last column of $\mathbf{R}$ that

$$\frac{r_{02}}{r_{22}} = \frac{-c_\alpha \cdot s_\beta}{c_\alpha \cdot c_\beta} \text{ , and consequently, } \quad \tan\beta = \frac{s_\beta}{c_\beta} = \frac{-r_{02}}{r_{22}}. \tag{23}$$

The correct quadrant of β can be determined from the signs of nominator and denominator. Thus, it is important that we write the minus sign at the r_{02} term and not at r_{22}. Since we assumed that $|\alpha| < \pi/2$, we know that $c_\alpha > 0$. This makes it easier to obtain the correct quadrant for β in Eq. (23). The angle γ can be obtained similarly from r_{10} and r_{11}.

7. RESULTS

We applied our algorithm to a number of sequences with varying complexity of camera motion. For each of the sequences, the camera rotations and the focal lengths have been extracted. The rotations are further factorized into the Euler-angle sequence of Eq. (21).

Figure 10(a) shows the extracted rotation angles for a sequence containing a simple horizontal pan. Since the rotation angle is limited, no multi-sprite partitioning was required. The figure includes the results after the linear calibration step and also after the non-linear refinement. The estimated focal-length was almost constant throughout the sequence, with a measured focal-length of 593 pixels for the linear algorithm and 617 pixels for the non-linear optimization. The rotation angles are very close for both estimation algorithms. The difference is mainly a scaling factor that is due to the different focal-length estimate.

In Figure 9, we repeated the same experiment with the well-known *stefan* test-sequence. Since the *stefan* sequence comprises a wide field of view, a multi-sprite partitioning had to be used. If we compare the linear estimation with the non-linear estimation, we can observe the same behaviour as for the first sequence. The estimates of the rotation angles are similar except for a scaling factor, which again is a result of the differing focal-length estimates.

Experiments with several other sequences reveal that the accuracy of the focal-length estimate is numerically more sensitive than the rotation angles. The linear algorithm alone already succeeds in computing a good estimate of the angles. However, since the amplitude of the angles is connected to the focal-length, the amplitude of the angles can slightly differ in the result of the linear algorithm.

Finally, we applied the calibration algorithm to a sequence with very complex camera motion. To provide an impression of the sequence, we depicted the three background sprites that were created in the motion estimation step (Fig. 11). The corresponding estimated camera rotation parameters are depicted in Fig. 10(b). The provided results do correspond smothly with the camera motion in the scene.

To evaluate the robustness of convergence for the non-linear optimization algorithm, we made the experiment to omit the initialization of the parameters based on the result of the linear estimation algorithm. Instead, we started with zero rotation angles and the principal points at the image centers. The focal length has been set to the length of the image diagonal. Even with this simple initialization, the non-linear optimization converged reliably to the same solution for all our test sequences.

So far, the accuracy of the obtained parameters could only be evaluated by visual inspection. As experiment, we placed the input images into a virtual 3-D space at the positions defined by the calibration parameters (Fig. 8). Viewed from the camera position, all images fit smoothly and no estimation errors could be observed. Evaluation of the absolute estimation accuracy requires to know the ground-truth value for the camera pose, which we do not yet have available.

8. CONCLUSIONS

This paper described a new algorithm the estimate camera calibration parameters from a video sequence. The parameters are expressed in the physically meaningful units of the camera rotation angles and the varying focal-length. The combination of the calibration algorithm with the multi-sprite motion-estimation approach made it possible to carry out the estimation for general camera motion without limitation on the observable field of view.

The algorithm consists of two steps, where the first part determines a first estimate of the camera parameters. The accuracy of this estimate is improved in an optional second step. We have observed that especially the camera rotation angles obtained in the first step are already closely following the camera motion. The relative change of focal-length is also reproduced accurately, but the absolute value can differ from the results obtained with the refinement step.

Hence, for applications that do not require absolute values for the camera parameters (like video content analysis), applying only the first step is already sufficient. The second step is only required if the application depends on an accurate estimate of the focal-length, such as, e.g., augmented reality applications.

REFERENCES

1. S. Coorg and S. Teller, "Spherical mosaics with quaternions and dense correlation," *Int. J. Comput. Vision* **37**(3), pp. 259–273, 2000.
2. D. Farin, P. H. N. de With, and W. Effelsberg, "Minimizing MPEG-4 sprite coding-cost using multi-sprites," in *SPIE Visual Communications and Image Processing (VCIP)*, **5308**, pp. 234–245, Jan. 2004.
3. D. Farin, P. H. N. de With, and W. Effelsberg, "Video-object segmentation using multi-sprite background subtraction," in *Proc. IEEE International Conference on Multimedia and Expo (ICME)*, 2004.
4. R. I. Hartley, "Self-calibration of stationary cameras," *International Journal of Computer Vision* **22**(1), pp. 5–23, 1997.
5. R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge University Press, 2000.
6. B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon, "Bundle adjustment – A modern synthesis," in *Vision Algorithms: Theory and Practice*, W. Triggs, A. Zisserman, and R. Szeliski, eds., *LNCS*, pp. 298–375, Springer Verlag, 2000.
7. J. Schmidt and H. Niemann, "Using Quaternions for Parametrizing 3–D Rotations in Unconstrained Nonlinear Optimization," in *Vision, Modeling, and Visualization 2001*, pp. 399–406, November 2001.
8. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical recipes in C*, Cambridge Univ. Press, 1988.
9. M. I. A. Lourakis and A. A. Argyros, "The design and implementation of a generic sparse bundle adjustment software package based on the levenberg-marquardt algorithm," tech. rep., Institute of Computer Science of the Foundation for Research and Technology - Hellas FORTH, Aug. 2004.
10. D. Farin and P. H. N. de With, "Estimating physical camera parameters for 3DAV video coding," in *Proc.* 25^{th} *Symposium on Information Theory in the Benelux*, pp. 201–208, June 2004.
11. L. Agapito, E. Hayman, and I. Reid, "Self-calibration of rotating and zooming cameras," *International Journal of Computer Vision* **45**(2), pp. 107–127, 2001.
12. R. I. Hartley, "Self-calibration from multiple views with a rotating camera," in *ECCV '94: Proceedings of the third European conference on Computer vision (vol. 1)*, pp. 471–478, Springer-Verlag New York, Inc., 1994.

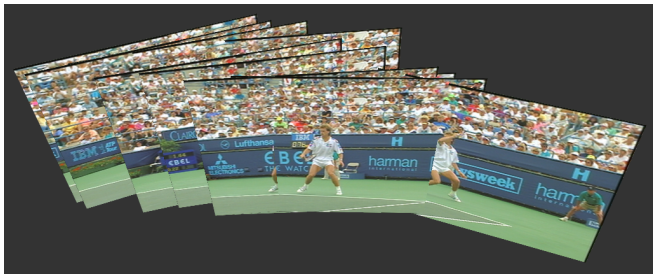
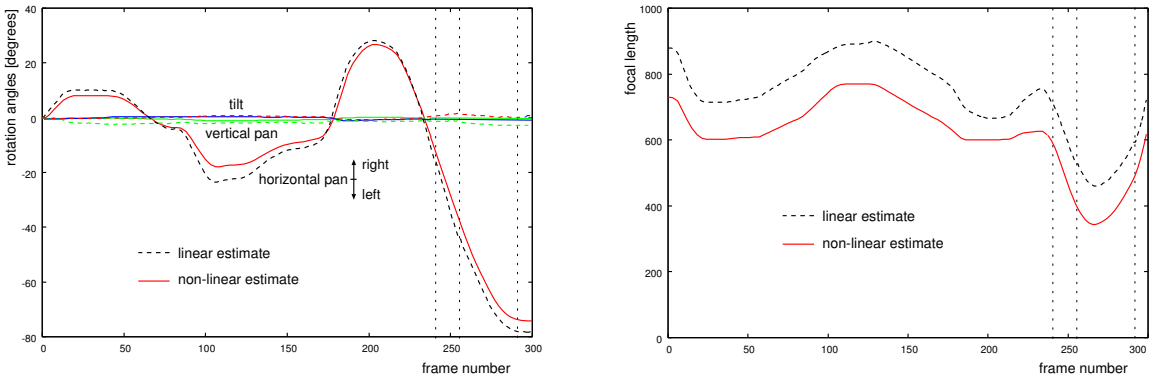
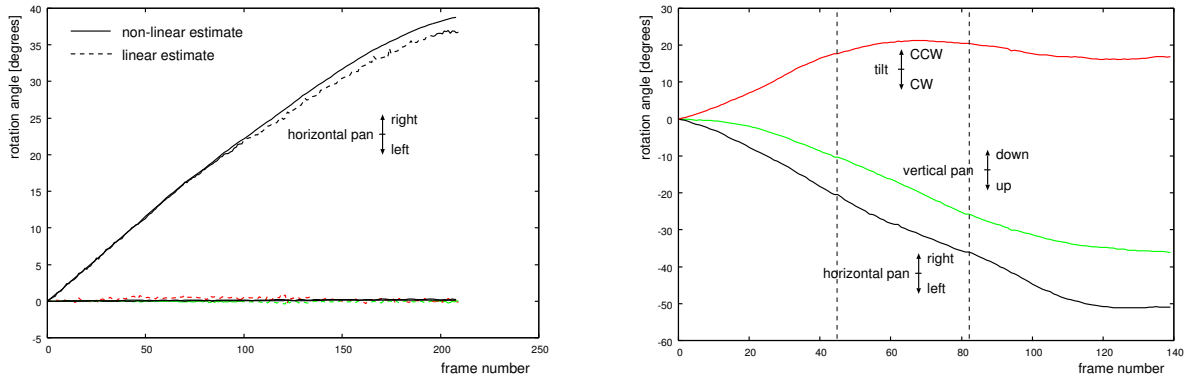


Figure 8. Image planes of the *Stefan*-sequence placed in their virtual position in 3-D space (see also[10]).



(a) Rotation angles. (b) Focal lengths.

Figure 9. Camera calibration results for the *stefan* sequence. The dotted vertical lines denote the multi-sprite boundaries.



(a) Pure horizontal pan. (b) Complicated camera motion *rail sequence*.

Figure 10. Camera calibration results for a pure horizontal pan and a complicated camera motion. The sequence used for graph (b) is depicted in Figure 11.

(a) frames 1–45 (b) frames 46–82 (c) frames 83–140

Figure 11. Multi-sprite for the *Rail* sequence. The sequence shows a camera rotation around all axes at the same time. At the beginning of the sequence, the camera is looking down. It turns left and around its optical axis until it looks left in the last frame.