# Part III

# From Camera Motion to 3-D Models

*The biggest difference between time and space*
*is that you can't reuse time.*
*(Merrick Furst)*

Chapter 12

# Estimation of Physical Camera Parameters

*The earlier presented camera-motion estimator is employing projective transforms to describe the geometric mapping between input frames. It was shown that these transforms are general enough to include all kinds of motion that occur for a rotational camera with varying focal length. In fact, when taking a sequence of elementary camera operations, it is easy to get the corresponding transform between two input images. However, the inverse problem of calculating the* physical *transform parameters from the frame-to-frame homographies is far more complicated. Despite the problems involved, it is valuable to carry out the estimation of the physical camera parameters, since it enables a whole new area of applications. Examples are video content analysis, where camera motion represents a valuable feature for sequence classification, or augmented-reality applications that insert computer-generated 3-D objects into a natural scene. This chapter presents two algorithms for factorizing the projective transforms of the camera motion into the physically meaningful camera rotation angles and the camera focal length. The first algorithm applies a linear optimization approach that is fast, but which has only limited accuracy. The second algorithm uses a non-linear bundle-adjustment algorithm that provides a high accuracy. Both algorithms have been combined with our multi-sprite technique of Chapter 6 to support unrestricted camera motion.*

## 12.1 Introduction

In current video coding and video-object segmentation systems, camera-motion is usually described with the projective motion model with eight parameters. A prominent example is the sprite-coding tool of the MPEG-4 standard, which uses the projective motion model to align background frames into a common background sprite image. In our video-object segmentation algorithm, we applied the same motion model, because of its ease of use and to enable the easy integration of the segmentation algorithm into MPEG-4 encoders.

In Chapter 2, it was derived that this model can describe any image motion that results from a rotating camera with varying focal length. Furthermore, the projective motion model also allows the alignment of the input images to a larger background-sprite image that can be used to reconstruct any arbitrary camera view from this background image. Later, Chapter 6 clarified that in practice, the principle of background sprites should be generalized to prevent degenerated transforms. However, with this multi-sprite generalization, the projective motion model can be applied to describe any motion of rotatorial cameras.

The parameters of the projective motion model describe the camera motion in an abstract way, without a direct correspondence to physically meaningful operations. However, when we derived the projective motion model for rotational cameras, it was indicated that the model can be considered as a concatenation of elementary physical operations, like rotating the cameras, and perspectively projecting the 3-D object onto the image plane. Unfortunately, it is not obvious how the parameters of these elementary operations can be recovered from the combined transformation matrix. In fact, it is not even possible to factorize every projective transformation into camera rotations and changes of camera zoom, since the general projective transformation also includes physically impossible transformations like anisotropic scaling or image skewing.

Nevertheless, it is advantageous or even required for some applications 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.

Whereas a qualitative analysis of camera motion can be sufficient for content analysis, there are other applications that require parameters with a high accuracy. An example is an augmented-reality application, where virtual 3-D objects are added to a recorded video such that they are seam-

lessly integrated. This can only be achieved if the view onto the synthetic 3-D objects is synchronized 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 [30] from video sequences, where especially the focal length of the camera is required. These parameters are usually not known and have to be estimated from the video sequence itself.

In this chapter, we propose two algorithms that factorize projective motion parameters into a sequence of elementary, physically meaningful transformations. 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. The parameter estimation is carried out in two steps, where 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. Both algorithms incorporate the multi-sprite partitioning of a sequence and consequently, they have no limitations about the range of parameters.

The chapter is organized as follows. We begin with a brief repetition of the global-motion estimation to clarify the notation used in this chapter. Furthermore, we give a short survey of previous approaches and discuss their performance. Section 12.3 describes the linear camera-calibration algorithm that was proposed by Hartley *et al.* in [85] and we present our extension of this algorithm to support the generalized multi-sprite approach. A non-linear calibration is described in Section 12.4, where we again consider in particular the integration of multi-sprites. Finally, we present visualizations of results for several sequences in Section 12.5.

### 12.1.1 Geometry of background image generation

This section gives a brief review of the geometry of rotational cameras to introduce the notation that we are going to use in this chapter.

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 $o_x, o_y$. The focal length $f_i$ and principal point are collected in an intrinsic camera parameters matrix

$$\mathbf{K}_i = \begin{bmatrix} f_i & \tau & o_x \\ 0 & \eta f_i & o_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{12.1}$$

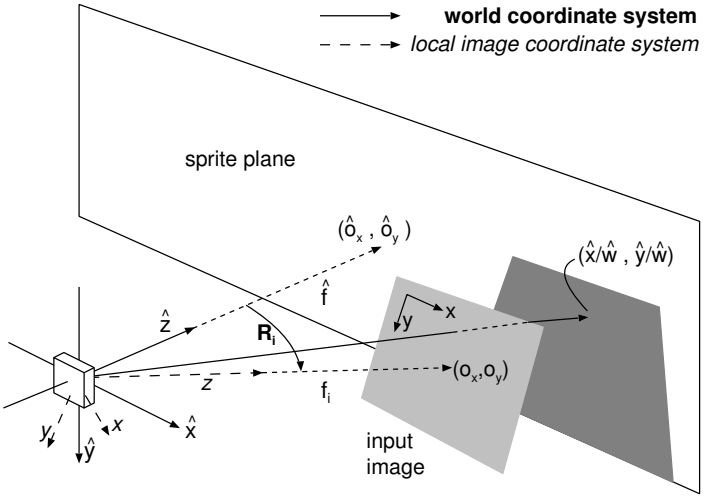



**Figure 12.1:** *The camera is located at the origin of the world coordinate system. The sprite plane is assumed to be orthogonal to the z-axis. Input images are at a distance to the origin that is equal to the focal length when the image was captured. A point $(x/w, y/w)$ on the image is projected onto the sprite position $(\hat{x}/\hat{w}, \hat{y}/\hat{w})$.*

Additionally, the parameter $\eta$ denotes the pixel aspect ratio and $\tau$ is the image skew. However, for typical CCD cameras, we can assume zero skew $\tau = 0$ and square pixels $\eta = 1$, so that we obtain the simpler intrinsic parameters matrix

$$\mathbf{K}_i = \begin{bmatrix} f_i & 0 & o_x \\ 0 & f_i & o_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{12.2}$$

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 matrix $\mathbf{R}_i$ gives the transformation between two images or, similarly, between a background sprite image and an input frame, as

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = \underbrace{\begin{pmatrix} f_i & 0 & o_x \\ 0 & f_i & o_y \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{K}_i} \mathbf{R}_i \underbrace{\begin{pmatrix} 1/\hat{f}_s & 0 & -(\hat{o}_x)_s/\hat{f}_s \\ 0 & 1/\hat{f}_s & -(\hat{o}_y)_s/\hat{f}_s \\ 0 & 0 & 1 \end{pmatrix}}_{\hat{\mathbf{K}}_s^{-1}} \begin{pmatrix} \hat{x} \\ \hat{y} \\ \hat{w} \end{pmatrix} = \mathbf{H}_i \begin{pmatrix} \hat{x} \\ \hat{y} \\ \hat{w} \end{pmatrix}. \tag{12.3}$$

This set-up is also visualized in Figure 12.1. Note that we denote all coordinates and parameters related to sprite coordinates with a hat on the variable name. Moreover, we use indices $i$ for variables relating to input images and $s$ for variables relating to sprites. Consequently, the transformation from a sprite $s$ to an image $i$ is denoted as $\mathbf{H}_i^{(s)}$. If the superscript is omitted, we simply mean the sprite to which the image $i$ was assigned by the multi-sprite partitioning.

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 inhomogeneous formulation of the projective motion model

$$x = \frac{h_{00}\hat{x} + h_{01}\hat{y} + h_{02}}{h_{20}\hat{x} + h_{21}\hat{y} + 1}, \quad y = \frac{h_{10}\hat{x} + h_{11}\hat{y} + h_{12}}{h_{20}\hat{x} + h_{21}\hat{y} + 1}. \tag{12.4}$$

Taking the sprite-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 $\mathbf{H}_{i;k} = \mathbf{H}_i\mathbf{H}_k^{-1}$.

### 12.1.2 Global motion estimation

The input for the camera-calibration algorithm is obtained from our global-motion estimation, where we use the more accurate parameters after the direct estimation step (Chapter 5). From this motion estimator, we obtain the parameters $h_{00}, \ldots, h_{21}$, corresponding to the multiplied matrices of Eq. (12.3).

For the case that the rotation between two frames is large, Chapter 6 has shown that they cannot be projected onto the same sprite. Following our multi-sprite technique, those frames are assigned to separate sprites. In the multi-sprite algorithm, the intention was to minimize the size of the generated sprite images for improved coding efficiency, but we can also use the same algorithm without modification 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 $M$ ranges

$$P = \{(1, p_2 - 1)\ ,\ (p_2, p_3 - 1)\ ,\ (p_3, p_4 - 1)\ ,\ \ldots\ ,\ (p_M, N)\}\,, \tag{12.5}$$

where $p_s$ denotes the first frame used in sprite $s$ ($s \in [1; M]$). Now, 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.

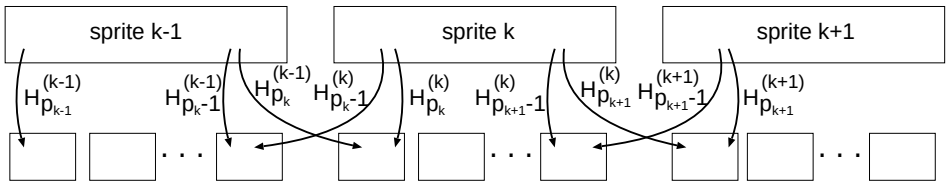



**Figure 12.2:** *Transformations between sprites and input frames. Every image $i$ is connected to its assigned sprite $k$ by the transform $\mathbf{H}_i^{(k)}$. The first and last images from each partition are also connected to the previous or next sprite, respectively.*

The multi-sprite partitioning assigned the images to separate sprites, but since we also need to know the geometric relation between images that are far apart, we also have to *connect* all the sprites. To achieve this, we compute for the first frame $p_s$ that has been assigned to a sprite $s$ not only the transform $\mathbf{H}_{p_s}^{(s)}$, but also the transform to the previous sprite $s-1$, which we denote by $\mathbf{H}_{p_s-1}^{(s-1)}$. Similarly, we also compute the transform between the last frame $p_{s+1}-1$ of the sprite to the next sprite $s+1$ (see Fig. 12.2). With these connecting transforms, we can compute the transformation between any two frames (e.g., frame $i$, assigned to sprite $s$, and frame $k$, assigned to sprite $s-1$) by

$$\mathbf{H}_{i;k} = \left(\mathbf{H}_i^{(s-1)}\right) \cdot \left(\mathbf{H}_{p_s}^{(s-1)}\right)^{-1} \cdot \left(\mathbf{H}_{p_s}^{(s)}\right) \cdot \left(\mathbf{H}_k^{(s)}\right)^{-1}. \tag{12.6}$$

For images that are several sprites apart from each other, more transforms have to be concatenated.

When chaining the inter-sprite transformations, we should 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 introduces problems if the rotation angle becomes larger. The problem is that at 90 degrees, we obtain $h_{22} = 0$ after the concatenation of the transforms, which cannot be normalized, and for larger angles, $h_{22}$ becomes negative. This will result in a change of orientation that makes it impossible to recover the correct rotation angles. Consequently, we cannot normalize the transforms to $h_{22} = 1$ when they are chained together. Note that it is still possible to carry out the inter-image motion estimation using normalized parameters, since the multi-sprite partitioning ensures that the transforms do not degenerate.

## 12.2 Previous work

Camera calibration is an active topic of research and several algorithms have been proposed to solve the problem. The approaches can be classified according to the applied camera models. Their main differences are in the parameters that are assumed to stay fixed, such as, e.g., the focal length or the principal point.

### 12.2.1 Estimation of focal length

Once the principal point and the focal lengths are known, it is straightforward to compute the camera rotation $\mathbf{R}$ by pre- and post-multiplying the intrinsic parameter matrices to the transform $\mathbf{H}_{k;i}$, this leading to $\mathbf{R} = \mathbf{K}_k^{-1}\mathbf{H}_{k;i}\mathbf{K}_i$. Since the principal point is usually near the image center, the problem reduces to find an estimate for the focal lengths.

In [180], Szeliski *et al.* described a simple approach for estimating the focal length from two images that were captured with a rotating camera. As shown previously, the transformation from frame $i$ to $k$ is then composed of the elementary transformations $\mathbf{H}_{k;i} = \mathbf{K}_k\mathbf{R}\mathbf{K}_i^{-1}$. Szeliski's algorithm assumes that the origin of the image coordinate system is at the principal point ($o_x = o_y = 0$). If this is not the case, the coordinate system can be shifted easily by factorizing $\mathbf{K}$ into $\mathbf{K} = \mathbf{T}\mathbf{K}'$, with

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & o_x \\ 0 & 1 & o_y \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{K}' = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{12.7}$$

so that we obtain

$$\mathbf{H}_{k;i} = \mathbf{T}\mathbf{K'}_k\mathbf{R}\mathbf{K'}_i^{-1}\mathbf{T}^{-1}, \tag{12.8}$$

or alternatively,

$$\mathbf{H'}_{k;i} = \mathbf{T}^{-1}\mathbf{H}_{k;i}\mathbf{T} = \mathbf{K'}_k\mathbf{R}\mathbf{K'}_i^{-1}. \tag{12.9}$$

Multiplying the rotation matrix $\mathbf{R} = \{r_{mn}\}$ with the intrinsic parameter matrices gives

$$\mathbf{H'}_{k;i} = \begin{pmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{pmatrix} \sim \begin{pmatrix} r_{00} & r_{01} & r_{02}f_i \\ r_{10} & r_{11} & r_{12}f_i \\ r_{20}/f_k & r_{21}/f_k & r_{22}f_i/f_k \end{pmatrix}. \tag{12.10}$$

Since we know that the rows and columns of the rotation matrix $\{r_i\}$ are orthogonal, we can derive

$$h_{00}h_{10} + h_{01}h_{11} + h_{02}h_{12}/{f_i}^2 = 0 \tag{12.11}$$

and

$$h_{00}h_{01} + h_{10}h_{11} + h_{20}h_{21} \cdot {f_k}^2 = 0. \tag{12.12}$$

From this, we can determine the focal lengths as

$$f_i^2 = \frac{-h_{02}h_{12}}{h_{00}h_{10} + h_{01}h_{11}} \quad \text{and} \quad f_k^2 = \frac{h_{00}h_{01} + h_{10}h_{11}}{-h_{20}h_{21}}. \tag{12.13}$$

Alternatively, we also can exploit the fact that the rows and columns of rotation matrices have unit norm. Since the transformation matrix can be scaled, we cannot test directly for unit norm, but we can still test for equal norm by

$$h_{00}^2 + h_{01}^2 + h_{02}^2/f_i^2 = h_{10}^2 + h_{11}^2 + h_{12}^2/f_i^2 \tag{12.14}$$

or

$$h_{00}^2 + h_{10}^2 + h_{20}^2 \cdot f_k^2 = h_{01}^2 + h_{11}^2 + h_{21}^2 \cdot f_k^2. \tag{12.15}$$

Again, we can derive equations[1] for the focal length as

$$f_i^2 = \frac{h_{02}^2 - h_{12}^2}{h_{11}^2 - h_{00}^2 + h_{10}^2 - h_{01}^2} \quad \text{and} \quad f_k^2 = \frac{h_{11}^2 - h_{00}^2 + h_{01}^2 - h_{10}^2}{h_{20}^2 - h_{21}^2}. \tag{12.16}$$

In total, we obtained four different equations to compute the focal lengths. Each of the focal lengths $f_i$, $f_k$ can be computed with two approaches, which we will further denote as the *orthogonal approach* and the *equal-norm approach*. If it is known that the focal length is fixed, the authors propose to take the geometric mean of the estimates for $f_i$, $f_k$.

### Degenerated cases

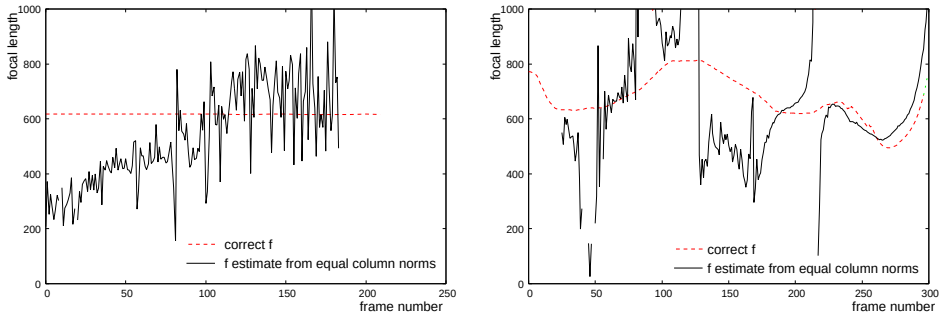
The described algorithm seems attractive for computing the focal length since it is easy to implement. However, it turns out to be very numerically unstable in practice. To get more insight into the behaviour, let us consider some special (but common) cases.

If the rotation angle between two frames is small, we have $h_{jj} \approx 1$, while for $i \neq j$, $h_{ij} \approx 0$. Consequently, all the denominators in Equations (12.13) and (12.16) approach zero, which makes the estimation unstable, especially at the presence of noise.

Let us observe the behaviour for rotations around the three coordinate axes. For the $x$-axis, it holds that

$$\mathbf{H}' \sim \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & f_i \sin\alpha \\ 0 & -(1/f_k)\sin\alpha & (f_i/f_k)\cos\alpha \end{pmatrix}, \tag{12.17}$$

[1] Note that the original article [180] contains several typing errors in the equations.




(a) *Roma* sequence. (b) *Stefan* sequence.

**Figure 12.3:** *Estimation of focal length as proposed by Szeliski* et al. *in [180]. Even though the considered images were 25 frames apart, the stability of the obtained estimate is very low.*

from which we can derive the constraint of orthogonal columns: $1 \cdot 0 + 0 \cdot \cos\alpha + 0 \cdot (-1/f_k)\sin\alpha = 0$. Obviously, this is not sufficient to solve for $f_k$. Similarly, the constraint for orthogonal rows does not provide $f_i$. However, using the approach of equal column norms works, since we obtain $f_i^2 = ((\cos\alpha)^2 - 1)/(-(\sin\alpha)^2/f_k^2) = f_k^2$. For rotation around the $y$-axis, similar results are obtained. When there is a pure rotation around the $z$-axis, all approaches fail to determine the focal lengths. There are also rotations around two axes that cause problems for the approach. Consider a rotation around the $z$-axis, followed by a rotation around the $y$-axis. In this case, the orthogonality approach for determining $f_i$ fails, because the transformation matrix is

$$\mathbf{H}' \sim \begin{pmatrix} \cos\beta\cos\gamma & \cos\beta\sin\gamma & f_i\sin\beta \\ -\sin\gamma & \cos\gamma & 0 \\ -1/f_k\sin\beta\cos\gamma & -1/f_k\sin\beta\sin\gamma & f_i/f_k cos\beta \end{pmatrix} \tag{12.18}$$

and from the orthogonality of rows, it follows that

$$-\sin\gamma\cos\beta\cos\gamma + \sin\gamma\cos\beta\cos\gamma + 0 = 0. \tag{12.19}$$

Again, this provides no information about $f_i$.

### Evaluation

We used two real-world sequences to evaluate the estimation of focal length based on Eq. (12.13). The *roma* sequence shows a pure horizontal pan with fixed focal length, while *stefan* has a more complicated camera motion with varying focal length. The transformations $\mathbf{H}$ were estimated with our

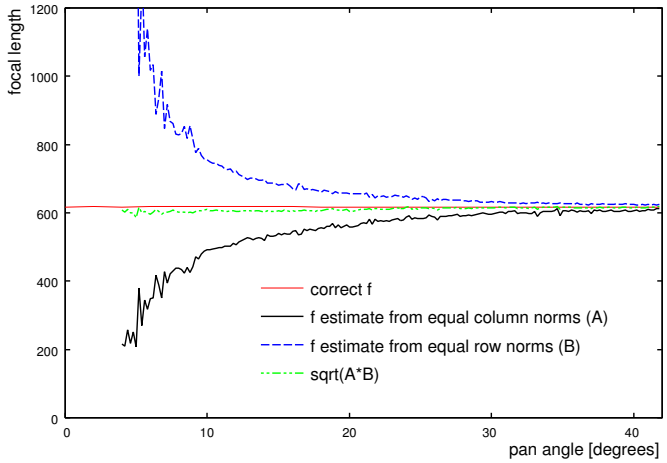



**Figure 12.4:** *Estimation of focal length for different rotation angles between the frames for the* roma *sequence. Larger rotation angles give a better accuracy. The geometric mean of the two approaches of row and column norm gives good results, but only works if the focal length is fixed.*

global-motion estimator, and the principal point $(o_x, o_y)$ was assumed to be in the image center.

To estimate the focal length, we fixed the distance between pairs of pictures to 25 frames, so that the rotation between them is large. Figure 12.3 depicts results for two example input sequences. It is well visible that the accuracy of the estimation is low and often it degenerates into cases where the right-hand-side of the equations is negative, which leads to no solution, since the square-root of negative values is undefined. We only depicted the results for the column-norm approach. Results for equal row norms are not very different, and results for the orthogonality approaches are much worse.

In a second experiment, we computed the estimated focal length depending on the rotation angle between the two frames. This experiment was carried out only with the *roma* sequence, since this sequence shows a smooth horizontal pan. The estimation was carried out between the first frame of the sequence and a later frame. The results (Fig. 12.4) show that the estimation accuracy increases with a larger rotation angle. Interestingly, the estimation error for the row-norm approach seems to be just the opposite of the estimation error for the column-norm approach. Hence, we can get a good estimate even for small rotation angles, when we take the geometric mean between the two estimates. Obviously, this is only possible if the focal length is fixed.

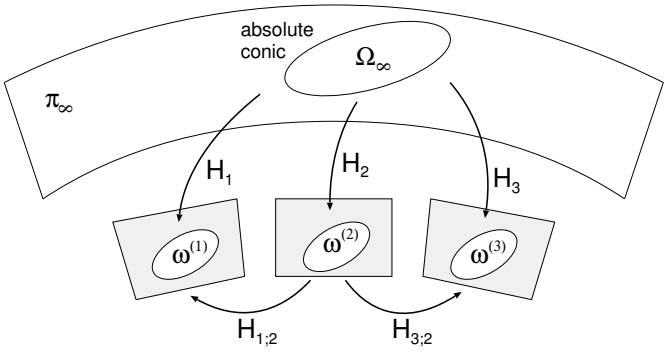



**Figure 12.5:** *The absolute conic $\Omega_\infty$ is transformed to the input frames and shows there as the IAC $\boldsymbol{\omega}^{(i)}$.*

To summarize, we can conclude that Eq. (12.13) can be applied to cases where the focal length is fixed and the rotation angle between the images is large. One such application can be image mosaicing from images. However, for video applications, the algorithm has a number of disadvantages: the principal point of the image should be known and the estimation accuracy is low or it even degenerates when the rotation angles are small.

## 12.3 Linear camera calibration

In the remainder of this chapter, we describe two algorithms for the estimation of physical camera parameters. The first algorithm is described in this section. It is a linear calibration algorithm based on proposal by Hartley *et al.* in [85], and which is extended to support multi-sprite motion estimation.

The next section will present a non-linear calibration algorithm that yields a higher accuracy. The second algorithm is an iterative optimization algorithm that can be implemented independently, or it can be initialized with the result of the first algorithm. The latter approach will provide a faster convergence of the solution.

### 12.3.1 Calibration using the image of the absolute conic

The linear calibration algorithm uses the method based on the transformation of the absolute conic [87, 85]. The absolute conic $\Omega_\infty$ is defined as the points $(x, y, z, w)^\top$ satisfying

$$(x, y, z)\mathbf{I}(x, y, z)^\top = 0 \quad \text{and} \quad w = 0, \tag{12.20}$$

with $\mathbf{I}$ being the $3 \times 3$ identity matrix. Hence, because of $w = 0$, the absolute conic lies in the plane at infinity $\boldsymbol{\pi}_\infty$ (see Fig. 12.5). Since the trivial solution $x = y = z = w = 0$ is not a valid point in projective space, the conic only consists of imaginary points. 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

$$\boldsymbol{\omega}^{(i)} = \mathbf{H_i}^{-\top}\mathbf{I}\mathbf{H_i}^{-1} = \mathbf{K_i}^{-\top}\mathbf{R_i}^{-\top}\mathbf{R_i}^{-1}\mathbf{K_i}^{-1} = \mathbf{K_i}^{-\top}\mathbf{K_i}^{-1}, \tag{12.21}$$

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

$$\boldsymbol{\omega}^{(i)} = \begin{bmatrix} 1/f_i^2 & 0 & -o_x/f_i^2 \\ 0 & 1/f_i^2 & -o_y/f_i^2 \\ -o_x/f_i^2 & -o_y/f_i^2 & o_x^2/f_i^2 + o_y^2/f_i^2 + 1 \end{bmatrix}. \tag{12.22}$$

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

Now, let us consider several views. The constraints that we derived previously are true for each of the views (each with its own $f_i$). Let us select one of the views as the reference view $r$. With the transformation $\mathbf{H}_{i;r}$, we can convert the coordinates of the reference view $r$ to coordinates of view $i$. Moreover, we can transform the conic $\boldsymbol{\omega}^{(r)}$ from the reference view to other views according to

$$\boldsymbol{\omega}^{(i)} = \mathbf{H}_{i;r}^{-\top}\boldsymbol{\omega}^{(r)}\mathbf{H}_{i;r}^{-1}. \tag{12.23}$$

The idea is now to use this transformation of conics to express the conics in all views $i$ using the conic parameters of the reference view $r$. Since the constraints for zero skew and square pixels must be fulfilled for all IACs $\boldsymbol{\omega}^{(i)}$, we can formulate constraints for all views and express these constraints in the parameters of the reference view. These equations can then be stacked into an equation system, from which we subsequently estimate the parameters of $\boldsymbol{\omega}^{(r)}$. Because the constraints of all views are expressed in the coordinate system of view $r$, we have enough equations to solve for the parameters of $\boldsymbol{\omega}^{(r)}$. When $\boldsymbol{\omega}^{(r)}$ is known, it is easy to obtain the intrinsic parameters using a Cholesky decomposition.

To exploit the constraints from all the views, the parameters of $\boldsymbol{\omega}^{(r)}$ are collected in a vector

$$\mathbf{c} = (c_1\ ,\ \ldots\ ,\ c_6)^\top = (\omega_{00}, \omega_{01}, \omega_{02}, \omega_{11}, \omega_{12}, \omega_{22})^\top. \tag{12.24}$$

Note that $\boldsymbol{\omega}^{(r)}$ is symmetric so that $\mathbf{c}$ only holds six parameters. Based on Equation (12.23), we derive a linear equation, which expresses every component of $\boldsymbol{\omega}^{(i)}$ as a linear combination of the parameters of $\mathbf{c}$:

$$\omega_{jk}^{(i)} = \sum_n \phi_{jk}^n \cdot c_n, \tag{12.25}$$

where $\phi_{jk}^n$ depend on the transformation matrices $\mathbf{H}_{i;r}^{-1}$.

The zero-skew assumption $\omega_{01}^{(i)} = 0$ of view $i$ can now be written in parameters of the reference view as

$$(\phi_{01}^1 \ , \ \dots \ , \ \phi_{01}^6) \cdot \mathbf{c} = 0. \tag{12.26}$$

Similarly, the square-pixel assumption $\omega_{00}^{(i)} - \omega_{11}^{(i)} = 0$ can be expressed as

$$(\phi_{00}^1 - \phi_{11}^1 \ , \ \dots \ , \ \phi_{00}^6 - \phi_{11}^6) \cdot \mathbf{c} = 0. \tag{12.27}$$

This process can be carried out for all views, generating two equations over $\boldsymbol{\omega}^{(r)}$ for each view. Stacking all the equations into a matrix $\mathbf{A}$, we obtain 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 derive $\mathbf{K}_i$ from $\boldsymbol{\omega}^{(i)}$, we carry out a Cholesky decomposition.[2]

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

$$\mathbf{H}_{k;i} = \mathbf{H}_k\mathbf{H}_i^{-1} = \mathbf{K}_k\mathbf{R}\mathbf{K}_i^{-1}, \tag{12.28}$$

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 recover the rotation matrix as

$$\mathbf{R} = \mathbf{K}_k^{-1}\mathbf{H}_k\mathbf{H}_i^{-1}\mathbf{K}_i. \tag{12.29}$$

[2]The Cholesky decomposition $\boldsymbol{\omega} = \mathbf{A}^\top\mathbf{A}$ is often implemented to give lower triangular matrices $\mathbf{A}$. We use the trick of transposing the matrix $\omega$ along the upward diagonal ($\omega_{00} \leftrightarrow \omega_{22}, \omega_{01} \leftrightarrow \omega_{12}, \omega_{10} \leftrightarrow \omega_{21}$) before and after the decomposition to find a decomposition into upper triangular matrices.

### 12.3.2 Integration of multi-sprite motion estimation

The algorithm presented so far assumed that accurate inter-image transforms between all pairs of views are available. However, the chaining of inter-image transforms to obtain transformations between frames that are further apart leads to an accumulation of errors, which subsequently also results in an inaccurate parameter estimation. This motivates why we added an additional motion-estimation step to our feature-based motion estimator. The solution is that instead of chaining inter-image transforms, we determine 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 usable 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 use the approach described in Section 12.1.2. While the original multi-sprite algorithm computed motion parameters only between the current view and the sprite to which they were assigned to, we extend this concept by computing also the transforms to the previous or successive sprites for the first and last frame of a sub-sequence, respectively. These transforms allow to chain the transformations between sprites to allow for large rotation angles. 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 are obtained with the long-term motion estimator, such that the parameters have *a-priori* a higher accuracy.

To adapt our calibration algorithm to the more general multi-sprite approach, we explore how Eq. (12.23) is implemented in practice. Because our long-term motion estimator gives sprite-to-image transforms $\mathbf{H}_i$, we

write Eq. (12.23) as

$$\boldsymbol{\omega}^{(i)} = \left(\mathbf{H}_i\mathbf{H}_r^{-1}\right)^{-\top} \boldsymbol{\omega}^{(r)} \left(\mathbf{H}_i\mathbf{H}_r^{-1}\right)^{-1} = \mathbf{H}_i^{-\top}\mathbf{H}_r^{\top}\boldsymbol{\omega}^{(r)}\mathbf{H}_r\mathbf{H}_i^{-1}, \qquad (12.30)$$

provided that both images $i$ and $r$ are assigned to the same sprite. If they are not on the same sprite, the inter-sprite transforms have to be used as demonstrated in Eq. (12.6) to span several sprites.

## 12.4 Non-linear camera calibration

In this section, we apply a bundle-adjustment algorithm [187] to increase the accuracy of the camera parameters that we obtained with the linear calibration algorithm. In the last section, we obtained the calibration in the linear case with an overdetermined equation system comprising constraints on the intrinsic parameters matrix. This equation system was solved in the least-squares sense. Hence, the quantity that is minimized is a physically meaningless algebraic error. This algebraic error is view-dependent and biased geometrically. The accuracy of the camera parameters can be increased by replacing this semantical weak error definition with a physically meaningful measure like Euclidean distance. The non-linear estimation algorithm resulting from it has the advantage that it allows to integrate additional constraints, which are difficult to express in the linear parameter estimation. For example, if the principal point is known, it is easy to integrate this constraint in the optimization.

The non-linear calibration uses also 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

$$\mathbf{H}_i = \mathbf{K}_i \cdot \mathbf{R}_i \cdot \hat{\mathbf{K}}^{-1}. \qquad (12.31)$$

If the sprite is constructed based on an input view $r$ as reference coordinate system, then $\hat{\mathbf{K}}$ 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 Section 6.5.4), 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. In our algorithm, we use

the backprojection error of the four image corners on the sprite plane. To compute this, we project the image corners onto the sprite plane using the estimated transform parameters of the global motion estimation, and also using the current estimate of the physically camera parameters. The Euclidean distance between both defines the backprojection error.

More precisely, a pixel $\mathbf{p}$ on view $i$ is projected to the sprite position with the measured motion parameters $\mathbf{H}_i$ according to

$$\breve{\mathbf{p}} = \mathbf{H}_i^{-1}\mathbf{p}. \tag{12.32}$$

Similarly, we project the same pixel onto the sprite position using the current estimate of the physical camera parameters,

$$\mathbf{p}' = \hat{\mathbf{K}}\mathbf{R}_i\mathbf{K}_i^{-1}\mathbf{p}. \tag{12.33}$$

With $d(\cdot,\cdot)$ denoting Euclidean distance, we define the error of position $\mathbf{p}$ in view $i$ as the squared distance between both parameterizations (Fig. 12.6),

$$e_i(\mathbf{p}) = d(\breve{\mathbf{p}}, \mathbf{p}')^2. \tag{12.34}$$

The total error $E$ for all parameters is then simply the sum over all views $i$ and corner positions $n$, thus

$$E = \sum_i \sum_{n=1}^{4} d(\mathbf{H}_i^{-1}\mathbf{p}^{(n)} \ , \ \hat{\mathbf{K}}\mathbf{R}_i\mathbf{K}_i^{-1}\mathbf{p}^{(n)}) = d(\breve{\mathbf{p}}_i^{(n)}, \mathbf{p'}_i^{(n)}), \tag{12.35}$$

where $\mathbf{p}^{(n)}$ denotes the $n$-th corner position of an input image.

### 12.4.1 Parameterization

The parameters to be estimated 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[3]. Additionally, we include parameters for the projection into the sprite coordinate system, comprising the focal length $\hat{f}$ and the principal point $\hat{o}_x, \hat{o}_y$ in the sprite.

For the subsequent optimization, we need a suitable parameterization of the camera rotation. A camera rotation has three free parameters, but a rotation matrix as used previously has nine parameters. Compared to the three free parameters, this is an unnecessary over-parameterization. An

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

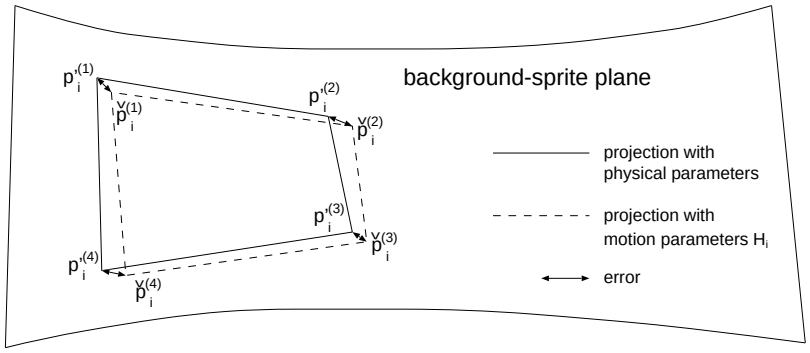



**Figure 12.6:** *The corners of an input image are projected into the sprite coordinate system, once with the global motion parameters and once with the physical camera parameters. The Euclidean distance between the respective corner positions define the error function for the physical camera parameters.*

alternative parameterization with Euler angles has only three parameters, but it shows singularities (see Section 2.4.2) that lead to problems in the optimization [91]. It has been shown [164] that better numerical stability (see Section 2.4.2) can be obtained with a parameterization using quaternions. A quaternion $\mathbf{q}$ can be associated with a rotation, when $||\mathbf{q}|| = 1$ (see Section 2.4.2). We parameterize rotations using the four components of the quaternion, without imposing the unit norm constraint. For all transform calculations, the quaternion parameters are converted to a rotation matrix. In this conversion, the quaternion parameters are normalized to unit norm. Effectively, this means that we have one additional degree of freedom in the optimization that has no effect on the error.

All camera parameters are stacked into a parameter vector $\mathbf{x}$. In particular, each image has a quaternion for the rotation and the focal length parameter. Since the sprite plane serves as the reference, no rotation parameters are estimated for it. Hence, we only estimate the focal length and the principal point of the sprite plane. The parameter vector $\mathbf{x}$ becomes

$$\mathbf{x} = (\underbrace{\mathbf{q}_1, f_1,\ \mathbf{q}_2, f_2,\ \ldots,\ \mathbf{q}_N, f_N,\ o_x, o_y}_{\text{image view parameters}}\ ,\ \underbrace{\hat{f},\ \hat{o}_x, \hat{o}_y}_{\text{sprite param.}}\ )^\top. \tag{12.36}$$

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

$$\mathbf{y} = (\underbrace{\underline{\mathbf{p}}'^{(1)}_1, \underline{\mathbf{p}}'^{(2)}_1, \underline{\mathbf{p}}'^{(3)}_1, \underline{\mathbf{p}}'^{(4)}_1}_{\text{first view}}\ ,\ldots,\underbrace{\underline{\mathbf{p}}'^{(1)}_N, \underline{\mathbf{p}}'^{(2)}_N, \underline{\mathbf{p}}'^{(3)}_N, \underline{\mathbf{p}}'^{(4)}_N}_{\text{view } N})^\top, \tag{12.37}$$

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

$$\breve{\mathbf{y}} = (\underbrace{\underline{\breve{\mathbf{p}}}_1^{(1)}, \underline{\breve{\mathbf{p}}}_1^{(2)}, \underline{\breve{\mathbf{p}}}_1^{(3)}, \underline{\breve{\mathbf{p}}}_1^{(4)}}_{\text{first view}}, \ldots, \underbrace{\underline{\breve{\mathbf{p}}}_N^{(1)}, \underline{\breve{\mathbf{p}}}_N^{(2)}, \underline{\breve{\mathbf{p}}}_N^{(3)}, \underline{\breve{\mathbf{p}}}_N^{(4)}}_{\text{view } N})^\top, \tag{12.38}$$

These three vectors (parameters $\mathbf{x}$, measured point positions $\mathbf{y}$, and point positions $\breve{\mathbf{y}}$ for the parameter estimate) 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 is minimized. Hence, computing

$$\min_{\mathbf{x}} \sum_{i=1}^{N} \sum_{n=1}^{4} d(\mathbf{p}'^{(n)}_i, \breve{\mathbf{p}}_i^{(n)})^2 = \min_{\mathbf{x}} ||\breve{\mathbf{y}} - \mathbf{y}||^2 \tag{12.39}$$

provides the solution vector $\mathbf{x}$ with the camera-calibration parameters.

### 12.4.2 Generalizing to multi-sprites

For a possible extension to arbitrary camera rotations, like in the linear calibration algorithm, we have to generalize the algorithm to multiple sprites. At first glance, it seems that a solution could be to process each sprite individually and then concatenate the estimated transforms. However, this approach cannot work in cases where some sprites are generated only from a camera-zoom operation (for example, see Figure 6.17 or 6.22(d)). As we have already seen in the beginning of this chapter, the focal length can only be estimated if there is some camera rotation present in the sequence. Although it is not possible to estimate the focal lengths for the images of a sprite that only shows camera zoom, we can solve it when we use the information about the focal lengths from a neighboring sprite. Since the last and first images of every sprite are also projected to the neighboring sprite, and because the focal length of this image is obviously the same in both projections, we can use those images as the connection that helps in the parameter estimation for the sprite showing only the zoom operation. Similar cases also occur if the estimation in some sprites are difficult to obtain because the rotation angle is small, or when there are only a small number of frames in the sprite.

Hence, to provide an approach that works in the general case, we have to jointly estimate the parameters of all sprites simultaneously. To achieve this, we build a parameter vector with the following conventions.

- *The image rotations are defined relative to the sprite to which they are assigned to.* This makes the estimation for these images independent from the location of the other sprite planes.

- *Rotations between sprites are parameterized as the rotation between two consecutive sprites and not to a global reference.* This reduces the interdependency between parameters and leads to a faster convergence in the optimization.

- *Backprojection costs are calculated between images and the sprite to which they are assigned*, but additionally, *the first and last frame of each frame is also projected onto the previous and successive sprites, respectively.*

The parameters and costs that are involved in the computations are illustrated in Figures 12.7 and 12.8. All the image and sprite-calibration parameters are again stacked into an extended parameter vector $\mathbf{x}$ such that we construct a vector of the following layout:

$$\begin{aligned}\mathbf{x} =(&\underbrace{\mathbf{q}_1, f_1,\ \mathbf{q}_2, f_2,\ \ldots,\ \mathbf{q}_N, f_N,\ o_x, o_y}_{\text{image-view parameters}}\ ,\\ &\underbrace{\hat{f}_1, \hat{o}_{x;1}, \hat{o}_{y;1},\ \hat{\mathbf{q}}_2, \hat{f}_2, \hat{o}_{x;2}, \hat{o}_{y;2}, \ldots,\ \hat{\mathbf{q}}_M, \hat{f}_M, \hat{o}_{x;M}, \hat{o}_{y;M}}_{\text{sprite-view parameters}})^\top.\end{aligned} \tag{12.40}$$

Note that every sprite has its own estimate of the principal point, but that all input images are assumed to have to same principal point.

The measurement vectors $\mathbf{y}$ and $\breve{\mathbf{y}}$ are constructed similarly as before, except that the corners of the first and last image of every sprite appear twice in these vectors, since they are projected onto two different sprites (Fig. 12.8). Furthermore, because the points are projected onto different sprites, the coordinates in these vectors are not expressed in the same coordinate system, but each position in the vector is relative to the corresponding sprite coordinate system. This does not change the error definition, which remains $E = ||\mathbf{y} - \breve{\mathbf{y}}||^2$.

### 12.4.3 Optimization algorithm

For the optimization, we apply the Levenberg-Marquardt (LM) algorithm [151]. 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 $\breve{\mathbf{y}}$ in terms of minimizing $(\breve{\mathbf{y}} - \mathbf{y})^\top(\breve{\mathbf{y}} - \mathbf{y})$. Starting with a

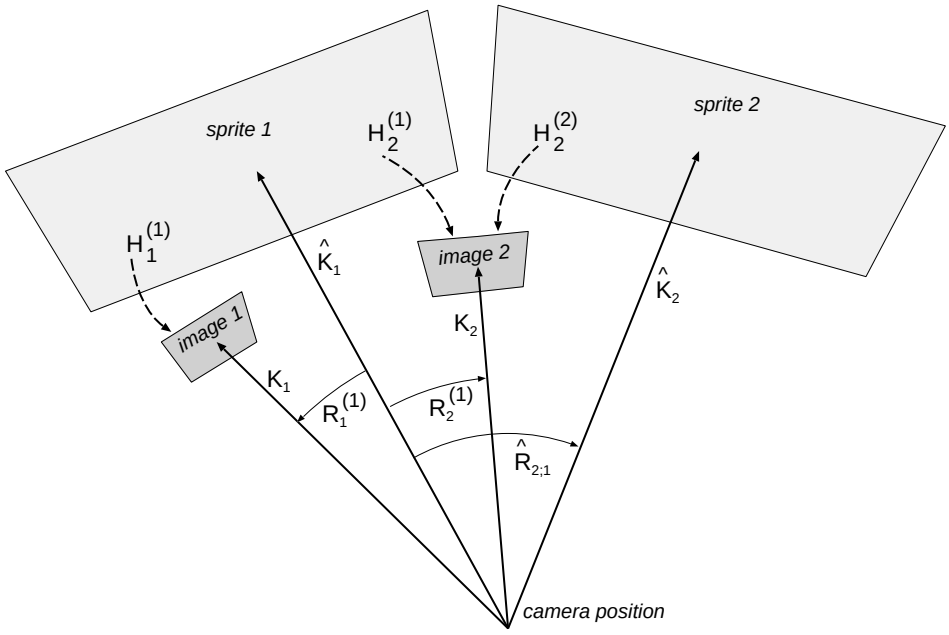



**Figure 12.7:** *The transformations involved in the non-linear camera calibration algorithm for multi-sprite sequences. In this example, both images are assigned to sprite 1.*

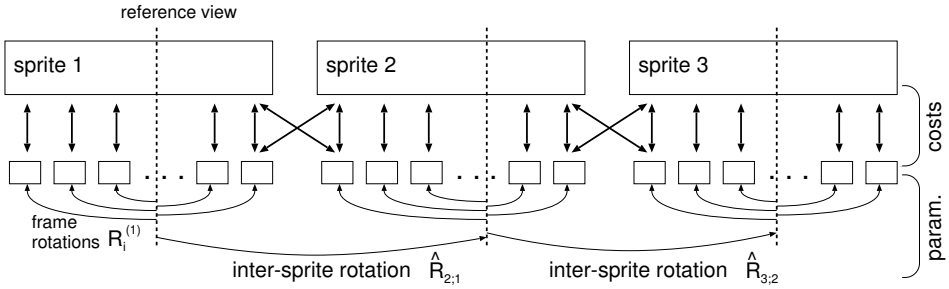



**Figure 12.8:** *The transformations and backprojection costs for a multi-sprite case. The thin arrows in the lower half indicate the transformations that are used in the optimization. Note that most image transforms are independent from each other. The sprites are connected with transforms between consecutive sprites. The upper half of the figure indicates the backprojections that are included in the error calculation. Note that the first and last image of each sprite is projected onto two sprites to provide a connection between these sprites.*

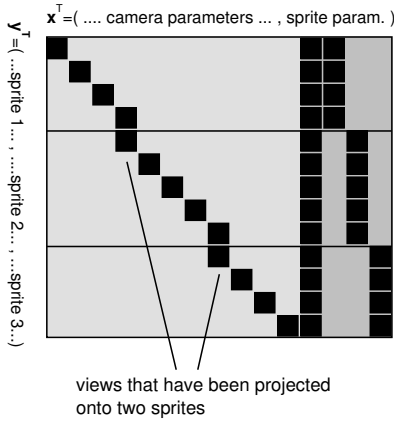



**Figure 12.9:** *Non-zero entries of the Jacobian matrix* $\mathbf{J}$.

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

$$(\mathbf{J}^\top \mathbf{J} + \lambda \mathbf{I})\boldsymbol{\delta}_\mathbf{x} = -\mathbf{J}^\top (\check{\mathbf{y}} - \mathbf{y}) \tag{12.41}$$

where $\mathbf{J}$ is the Jacobian matrix $\partial \mathbf{y}/\partial \mathbf{x}$. The parameter $\lambda$ is controlled by the LM algorithm to switch between steepest-descent (large $\lambda$) and Gauss-Newton ($\lambda = 0$) behaviour.

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. 12.9). Therefore, an optimized implementation [187, 117] exploiting this spare matrix structure can be used to solve Eq. (12.41).

The non-linear optimization algorithm is initialized with the result of the linear estimation. This leads to a fast convergence in only a few iterations. We have observed that the non-linear optimization also converges reliably when it is initialized with zero rotations and a common value for the focal lengths and principal points. However, this requires more iterations and leads to a higher total computational complexity.

### 12.4.4 Recovering rotation angles

For the final algorithm output, we desire to express the output in angles of an Euler rotation sequence. This requires that we first express all rotations

relative to a common reference view. Second, we have to convert the internal quaternion representaton of rotations to the Euler-angle representation.

After selecting a reference view, the rotations to every other view can be determined easily by a concatenation of rotations. This can be done directly with the quaternion representation since the multiplicaton of two quaternions $\mathbf{q}_b\mathbf{q}_a$ gives the joint rotation, composed of first $\mathbf{q}_a$ and then $\mathbf{q}_b$.

Using the technique described in Section 2.4.2, it is possible to obtain Euler rotation angles from a rotation matrix. Instead of first converting the representation to a rotation matrix using Eq. (2.22) and then extracting the angles, the steps can be combined and we compute the angles directly from the quaternion representation as

$$\sin\alpha = 2q_wq_x - 2q_yq_z \ , \tag{12.42}$$
$$\tan\beta = \frac{-2q_xq_z - 2q_wq_y}{1-2q_x^2-2q_y^2} \ , \quad \tan\gamma = \frac{2q_xq_y + 2q_wq_z}{1-2q_x^2-2q_y^2}.$$

Note that these angles are for the rotation sequence $\mathbf{R_y}(\beta)\mathbf{R_x}(\alpha)\mathbf{R_z}(\gamma)$. The angles are different for other rotation sequences.

## 12.5 Experimental results

We have 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 to the Euler angles.

Let us first consider the *roma* sequence, which consists of a pure horizontal camera pan. Figure 12.10 shows the extracted rotation angles. Since the rotation angle is limited, no multi-sprite partitioning was required. The figure depicts the results of both the linear calibration step and 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.

For visualization of the results, we have designed an OpenGL-based program that places the input frames of the sequence at their virtual 3-D position, using the estimated camera calibration parameters. Figure 12.11 depicts two views onto the 3-D positioned images of the *roma* sequence. It is visible that the input frames align nicely into a panoramic view, which shows that the estimation accuracy is good.

In Figures 12.12 and 12.13, we have repeated the same experiment for the well-known *stefan* test-sequence. Since the *stefan* sequence comprises

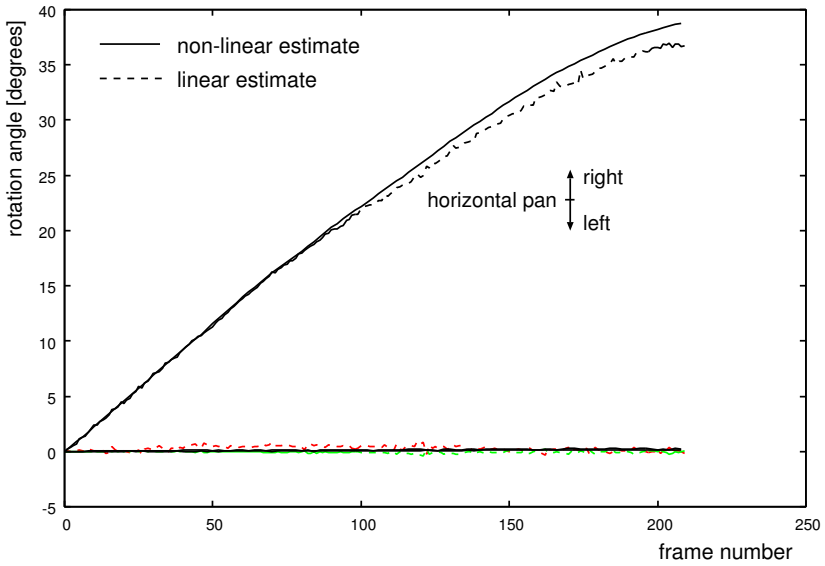


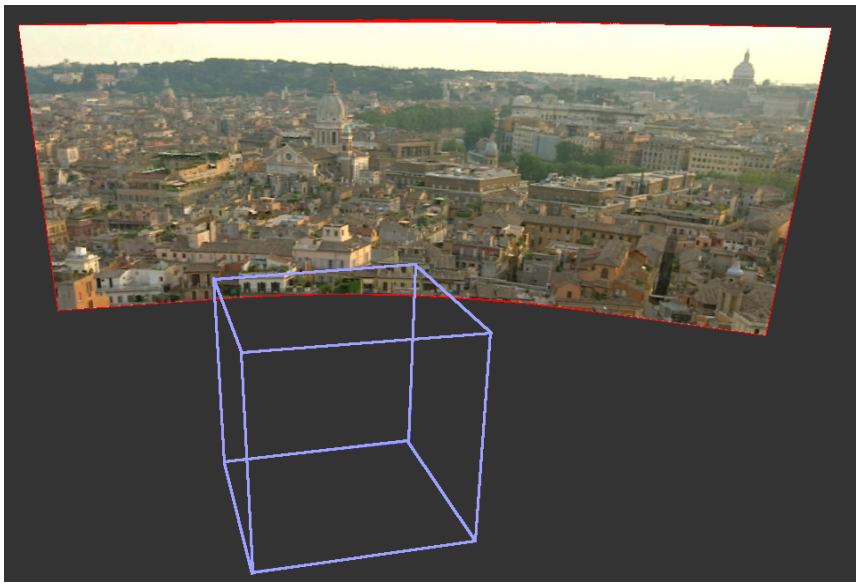

**Figure 12.10:** *Calibration result for the* roma *sequence, which is a pure horizontal pan.*

a wide field of view, a multi-sprite partitioning is required. If we compare the linear estimation with the non-linear estimation, we can observe the same behaviour as for the *roma* 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.

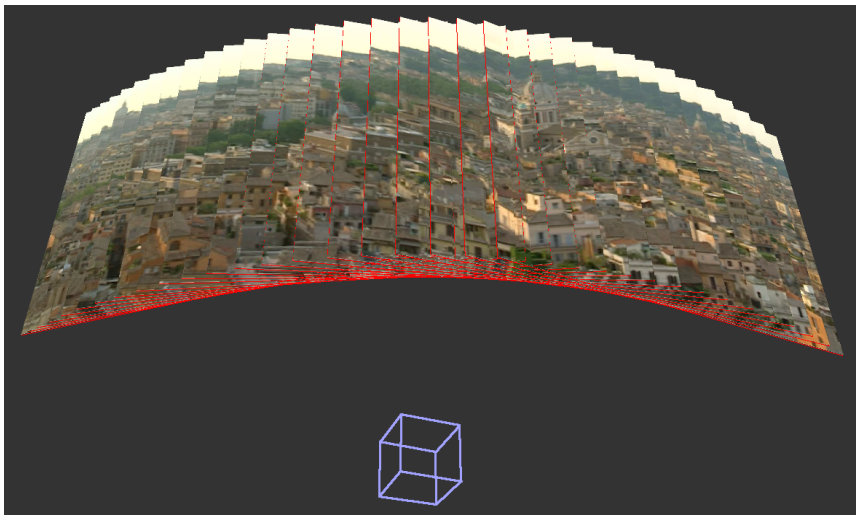
Finally, we have applied the calibration algorithm to sequences with very complex camera motion. We show here the results for the *rail* and the *nature-2* sequence, comprising three and seven sprites, respectively. The corresponding estimated camera-rotation parameters are depicted in Figure 12.16 and Figure 12.14. The provided results do correspond smoothly with the camera motion in the scene.

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.

To evaluate the robustness of convergence for the non-linear optimization algorithm, we have 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



(a) Front side.



(b) Back side.

**Figure 12.11:** *Images from the* roma *sequence, placed at their virtual image planes. The cube indicates the camera position. It is aligned to the camera viewing direction of the first frame, which is at the left side of the pan.*

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. We have used our 3-D visualization to display a virtual view from the camera position. At this position, all input images should fit together without seams between the images. Since this is achieved, we conjecture that the estimated parameters have a high accuracy. Evaluation of the absolute estimation accuracy requires the ground-truth value for the camera pose, which we do not have available yet.

## 12.6 Conclusions

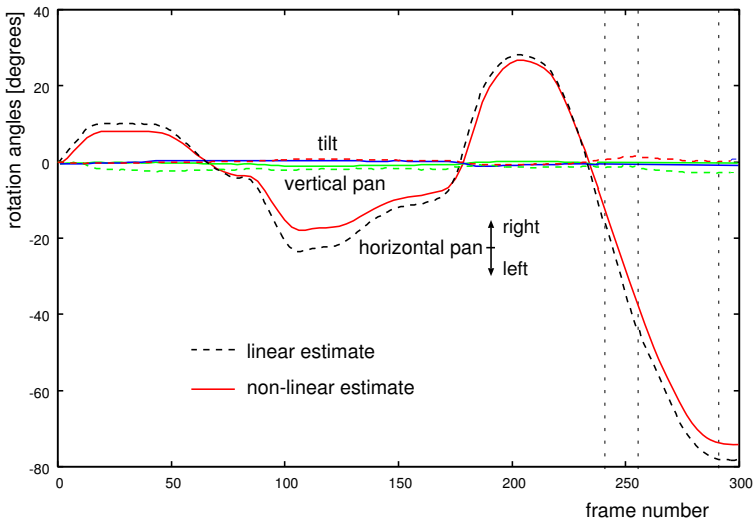
This chapter has described a new algorithm to estimate camera calibration parameters from a video sequence. The output parameters are expressed in the physically meaningful units of camera rotation-angles and the varying focal length.

The difference to the previous algorithm is that it is not required to have the input sequence itself as input, since our algorithm operats directly on the projective motion parameters as used in the MPEG-4 sprite coding or in the MPEG-7 parametric-motion descriptor. Hence, the algorithm can be used e.g. to transform the parametric-motion descriptors of MPEG-7 to the physical camera-motion descriptors. Moreover, as a potential application, the algorithm can be used in an MPEG-4 decoder to derive the camera motion from the camera-motion parameters and use these e.g. to add virtual 3-D objects into the scene.

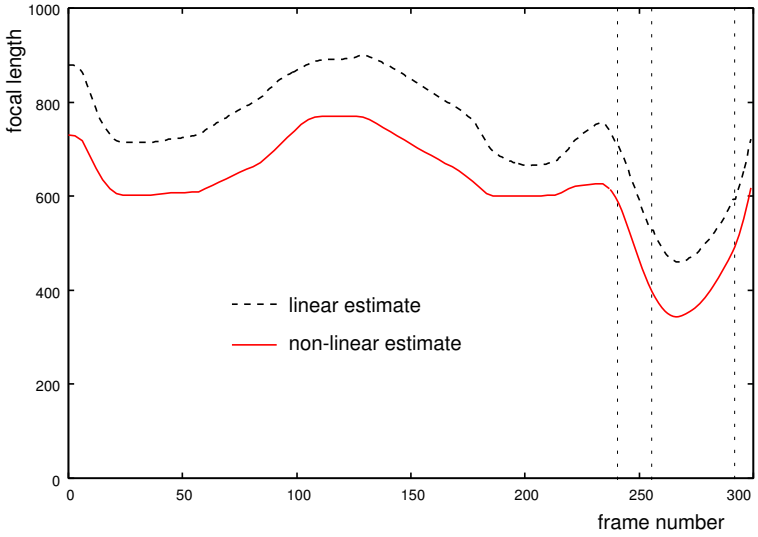
Our algorithm consists of two steps, where the first part determines a first estimate of the camera parameters. The accuracy of this estimate is improved in an optional second step. Because the calibration algorithm is combined with the multi-sprite motion-estimation approach, it is possible to carry out the estimation for general camera motion without limitation on the observable field-of-view.

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.

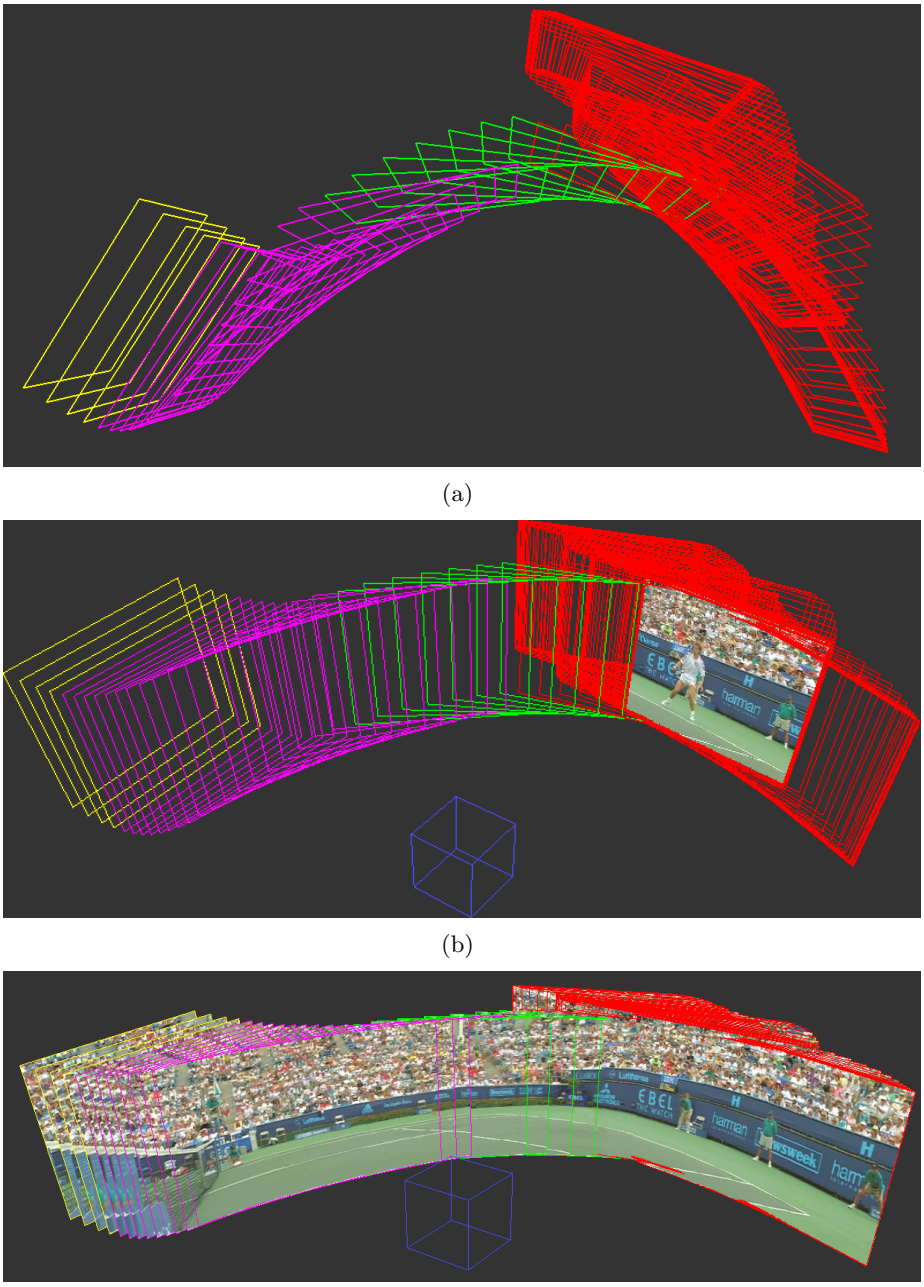



(a) Rotation angles.




(b) Focal lengths.

**Figure 12.12:** *Camera-calibration results for the* stefan *sequence. The dotted vertical lines denote the multi-sprite boundaries. Note that the last sprite contains only a few frames and shows no camera rotation. Even in this case, the focal length of all frames could be estimated successfully.*



(a)

(b)

(c)

**Figure 12.13:** *Illustration of the virtual image planes for the* stefan *sequence (every $2^{nd}$ frame is shown). Different colors of the frames borders indicate the sprites to which the frames were assigned.*

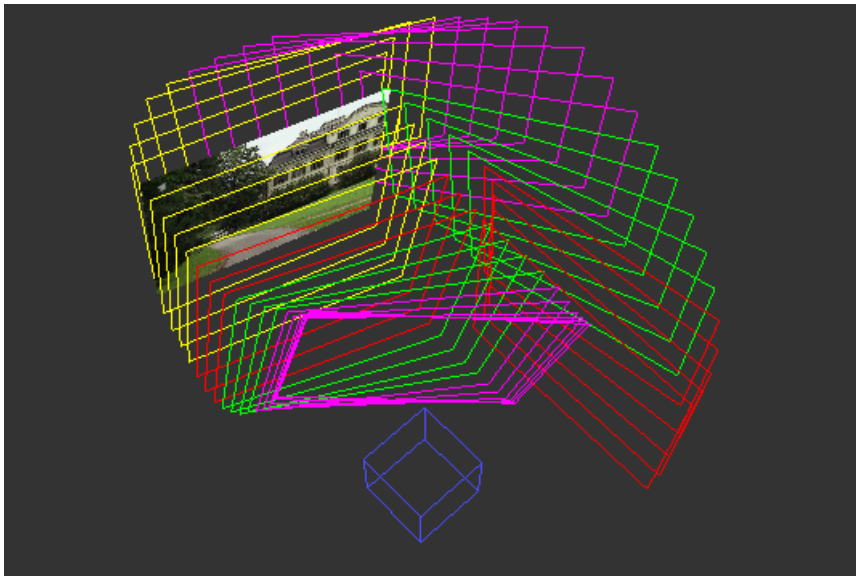


**Figure 12.14:** *3-D position of image planes for the* nature-2 *sequence (every* $10^{th}$ *frame is shown).*

(a) *Sprite 3.*

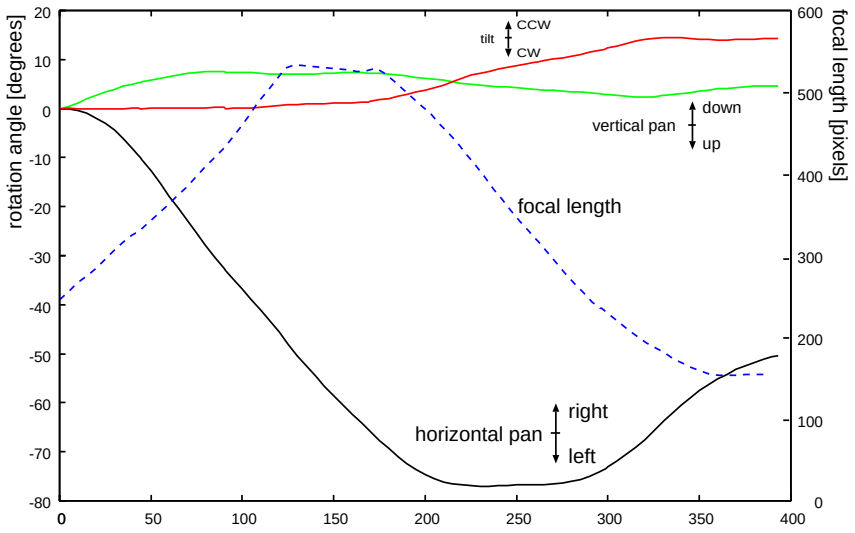
(b) *Sprite 2.*

(c) *Sprite 4.*

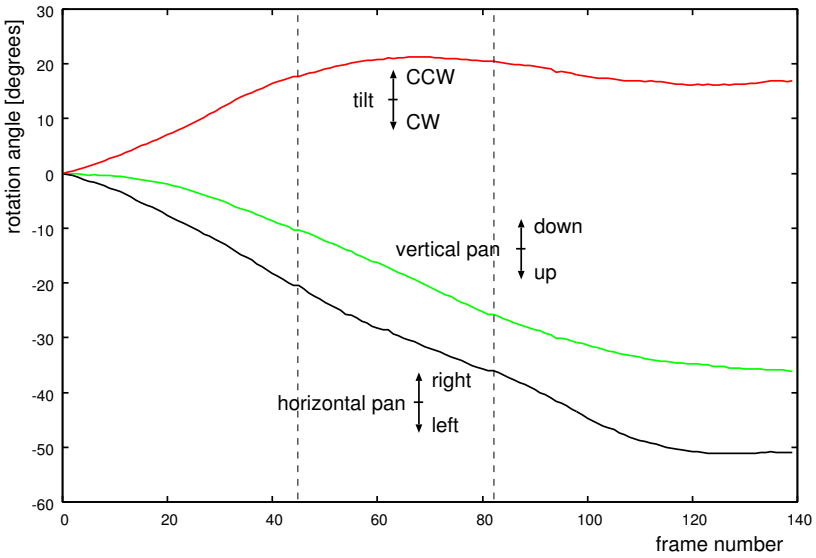
(d) *Sprite 7 (last).*

(e) *Sprite 1 (first).*

**Figure 12.15:** *Background sprites for* nature-2 *sequence. Sprites 5 and 6 are not shown*




(a) *Nature-2* sequence.




(b) *Rail* sequence.

**Figure 12.16:** *Camera calibration for two sequences with complicated camera motion.*