It's not an optical illusion, it just looks like one.
(Phil White)

Chapter 5

Background Reconstruction

If a video sequence is recorded with a rotational camera, it is possible to reconstruct an image of the scene background. In the context of our video-object segmentation system, this synthetic background image can be used for two purposes. First, the background images enable an easy segmentation algorithm that detects the foreground objects based on the differences to this background image. Second, the background image can be used together with the MPEG-4 sprite coding tools to transmit the background content of a video scene with a very low bit-rate. This section describes the background reconstruction process, involving the following main tasks. At the start, the accuracy of the camera-motion parameters is increased such that long-term consistency is achieved. After that, the input frames are combined into the background image such that moving foreground objects are removed. We briefly review some of the existing approaches and present a new algorithm for background estimation which is designed for difficult reconstruction cases in sequences where the background is only visible for a short time period.

5.1 Introduction

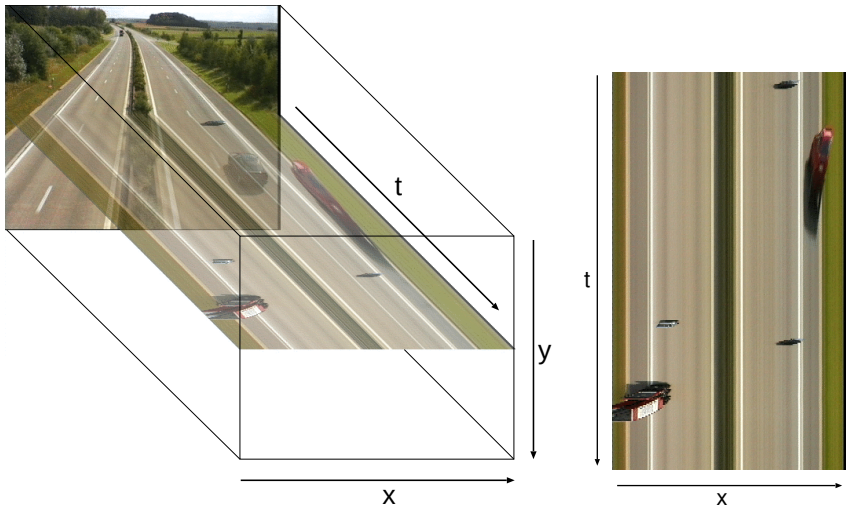
The segmentation algorithm that will be described in Chapter 7 requires a pure background image of the captured scene. When the scene background is available, the segmentation can be obtained by detecting changes between the background image and the input images. For some applications, the background image can be captured manually. One example are indoor surveillance systems that observe a non changing scene. However, in other cases, the background image should be synthesized automatically for various reasons. First, it can be impractical to wait for a moment when there is no foreground object visible, like when observing a busy highway. Second, the background may change slowly during the day, e.g., because of changing light conditions. Finally, automatic video segmentation for content analysis may operate on many small sequences for which it is tedious or impossible to generate background images by hand. Because of these reasons, algorithms that synthesize backgrounds automatically become important.

The basic principle of these algorithms can be explained using Figure 5.1. The figure shows a slice through the 3-D video volume, comprising the spatial x,y axes and the time t. Since the sequence was recorded with a static camera, background pixels do not change over time. However, during some time periods, the background image content can be occluded by foreground objects. Under the assumption that the background is visible for a longer time than the foreground, we can apply some averaging method to filter out the time periods showing foreground content and identify the background content.

In practice, it is not always true that the background content is visible during most of the time like in Figure 5.1(b). A much more difficult scene is shown in Figure 5.2, in which some parts of the background are only visible during a very short time, or they are not visible at all. The second part of this chapter will present a new algorithm that is more robust than previous algorithms in such difficult cases.

In the first part of this chapter, we study camera-motion compensation. Whereas background estimation for static cameras can simply observe pixels at constant position during time, this is not possible for moving cameras. As is visualized in Figure 5.3, a moving camera leads to trajectories through x-y-t space that should be followed for each background pixel. This is achieved by aligning all input frames to a common reference frame, which is at the same time the reference for the background image to be generated.

In the previous chapters, we described the estimation of camera motion, but unfornately, the obtained motion parameters cannot be used directly for the camera compensation because of two reasons. First, the motion



(a) Cut through a 3-D video volume for a sequence captured with a static camera.

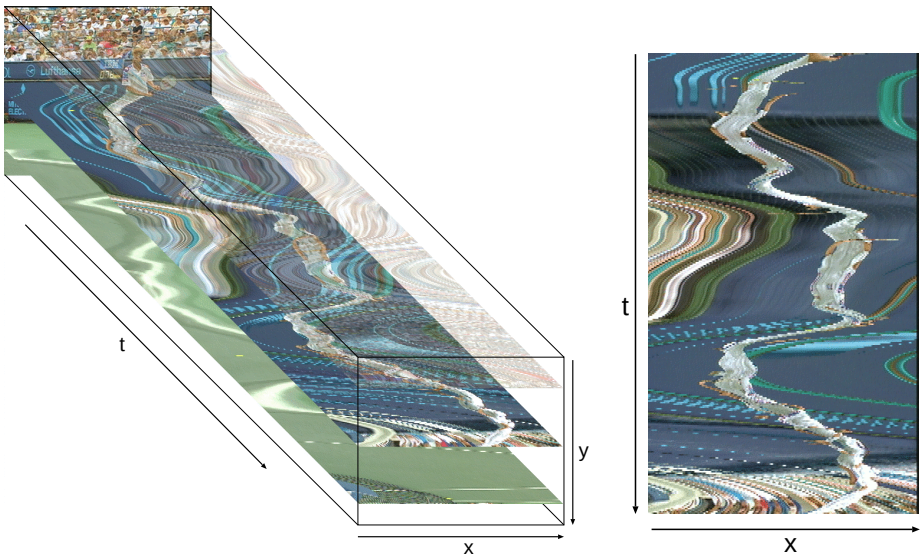
(b) An x-t slice of the 3-D video volume.

Figure 5.1: *For a static camera, the background does not change along the t-axis. However, foreground objects appear for some time and occlude the background.*

(a) Slice at row 140.

(b) Slice at row 180.

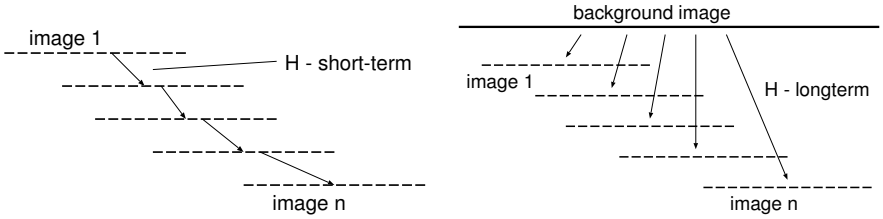
Figure 5.2: *Two x-t slices for an especially difficult sequence with many foreground objects. See Figure 5.21(a) for an example input frame.*



(a) Cut through a 3-D video volume for a sequence captured with a moving camera.

(b) An x-t slice of the 3-D video volume.

Figure 5.3: *Same visualization as in Fig. 5.1, but now for a moving camera. With a moving camera, the background pixels are not aligned in temporal direction.*



(a) short-term motion parameters

(b) long-term motion parameters

Figure 5.4: *The feature-based motion estimator described in the previous chapters gives us short-term motion parameters between successive frames. To combine all the images into a single background frame, long-term motion parameters are required.*

parameters describe the motion between of successive frames (short-term motion estimation) instead of being relative to a common background reference frame (long-term motion estimation). The difference is shown in Figure 5.4. Second, a simple chaining of short-term transforms to obtain long-term motion parameters would lead to an accumulation of estimation errors. Hence, we carry out a parameter refinement step that computes accurate long-term motion parameters.

5.2 Frame alignment

The first step in synthesizing a background image is to determine the accurate placement of the input frames in the background image. With the feature-based motion estimator described in the last chapter, we computed the transforms between successive input frames. These *inter-image* transforms, we have to derive the transformation from each of the input images to a common, virtual background image. Even though we can obtain the motion parameters with a very high accuracy of about 0.15 pixels (Table 4.1), small errors in these parameters will accumulate when we combine these transforms. These errors will show as blurring when several images are averaged. In the worst case, the alignment errors can lead to discontinuities along the boundaries of images that are combined together. Consequently, it is required to further refine the motion parameters by aligning each input image to the constructed background image with high accuracy. We commence with applying a *direct estimation method* [92] to obtain the motion parameters. This technique does not rely on detected feature-points and hence, in combination with the feature-based estimator, it can increase the robustness in those cases where the feature-based method yields only low accuracy because the number of features is low. The direct method requires a good initialization of the parameters, which can be obtained from the previous feature-based estimator. This combination of both estimation algorithms brings together the advantages of both. While the feature-based estimator supports fast camera motion, the direct estimation method preserves long-term consistency due to its higher accuracy.

5.2.1 Motion models for sprite generation

The MPEG-4 sprite coding tools allow to choose between four different motion models of which the projective model is the most general. In MPEG-4, a slightly different (but equivalent) parameterization of the projective model with four motion-vectors at the image corners is used. The other models are affine (where only three motion-vectors are transmitted), a rota-

tion/scale/translation model, which only requires two motion-vectors, and a translatorial model using only one vector. Each of these models is actually a superset of those with less motion-vectors.

As discussed in Section 2.5.4, all images recorded with a rotating camera can be aligned into a common reference image by applying the projective motion model with eight parameters. The parameter estimation of the projective motion model is often considered difficult since it is non-linear. For this reason, various authors, e.g., [97, 24] propose to approximate it with an affine model comprising six parameters. However, the affine model is only valid for an orthographic projection, i.e., for the case of very large focal length (see Section 2.4.3). In [172, 122], the bilinear model (eight parameters) or the biquadratic model (twelve parameters) have been proposed. But neither of these models is capable to describe a rotational camera motion, so that they are only applicable as an approximation to camera motion when the rotation angle is very small.

5.2.2 Geometry of background image generation

Let us recall the physical image-formation process for a rotation-only camera. The restriction to a rotating and zooming camera is necessary because with translatorial camera motion, the parallax effect leads to different speeds for objects at different depths. This would make it impossible to align the background images into a seamless mosaic[1].

Let us define the 3-D world coordinate system such that the camera is located at its origin (Figure 5.5). The camera captures a number of images I_i with different rotations $\mathbf{R}_i$ and focal lengths f_i. In a rotated local image coordinate-system, where the frontal viewing direction is along the positive z-axis, the corresponding 3-D position of each image pixel $(\hat{x}, \hat{y})$ in the focal plane is $(\hat{x}, \hat{y}, f_i)^\top$. For simplicity of notation, we assume that the origin of image coordinates is at the principal point, which can usually be assumed to be at the center of the image. Now let the virtual sprite-plane be placed orthogonal to the z-axis of the world coordinate system at a distance f_s.

[1] In fact, it is also possible to generate sprites for arbitrary camera motion including translation provided that the background is planar.

The projection of the image point $(\hat{x}, \hat{y})$ can then be determined by

$$\begin{aligned}\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} &= \underbrace{\begin{pmatrix} f_s & 0 & o_x \\ 0 & f_s & o_y \\ 0 & 0 & 1 \end{pmatrix}}_{\substack{\text{intrinsic camera parameters:} \\ \mathbf{K}_s}} \underbrace{\begin{pmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{pmatrix}}_{\substack{\text{extrinsic parameters (camera} \\ \text{rotation): } \mathbf{R}_i}} \begin{pmatrix} \hat{x} \\ \hat{y} \\ f_i \end{pmatrix} \\ &= \mathbf{K}_s \cdot \mathbf{R}_i \cdot \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & f_i \end{pmatrix}}_{\substack{\text{shift onto focal} \\ \text{plane: } \mathbf{K}_i^{-1}}} \begin{pmatrix} \hat{x} \\ \hat{y} \\ 1 \end{pmatrix} = \mathbf{H}_i \begin{pmatrix} \hat{x} \\ \hat{y} \\ 1 \end{pmatrix},\end{aligned} \tag{5.1}$$

where (o_x, o_y) is the position of the principal point on the sprite plane, and the resulting sprite coordinates (x', y', w') are given in homogeneous coordinates. Multiplying the intrinsic and extrinsic transformation matrices together, we obtain the combined matrix $\mathbf{H}_i$, describing the projection of the image coordinates from frame i onto the background plane. Taking the image-to-background 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 $\mathbf{H}_{i;k} = \mathbf{H}_i^{-1}\mathbf{H}_k$. The motion $\mathbf{H}_{i+1,i}$ between successive frames i and $i+1$ is known, since this is the output of our feature-based motion estimator. However, it is not yet possible to obtain the image-to-background transform from the inter-image transforms, because the location of the background plane has not been fixed. Basically, the background plane can be placed at an arbitrary position (see Fig. 5.6), but we will see in Chapter 6 that the placement of the background plane still has some important practical consequences. However, in this section, we will simply choose one arbitrary input frame as the reference frame r, and use it as the background image reference by setting the transformation $\mathbf{H}_r$ to the identity matrix: $\mathbf{H}_r = \mathbf{I}$. Since this fixes the relationship between the input frames and the background frame, we can obtain the remaining background transforms $\mathbf{H}_i$ by a suitable concatenation of known transforms. Let, for example, frame 4 be the reference frame, then we can set $\mathbf{H}_4 = \mathbf{I}$ and thus get $\mathbf{H}_1$ as

$$\mathbf{H}_4\mathbf{H}_{4,3}\mathbf{H}_{3,2}\mathbf{H}_{2,1} = \mathbf{H}_{4,3}\mathbf{H}_{3,2}\mathbf{H}_{2,1} = \mathbf{H}_1, \tag{5.2}$$

while we have to take inverse transforms for frames $i > r$. For, e.g., frame 7, we get

$$\mathbf{H}_7 = \mathbf{H}_4\mathbf{H}_{5,4}^{-1}\mathbf{H}_{6,5}^{-1}\mathbf{H}_{7,6}^{-1}. \tag{5.3}$$

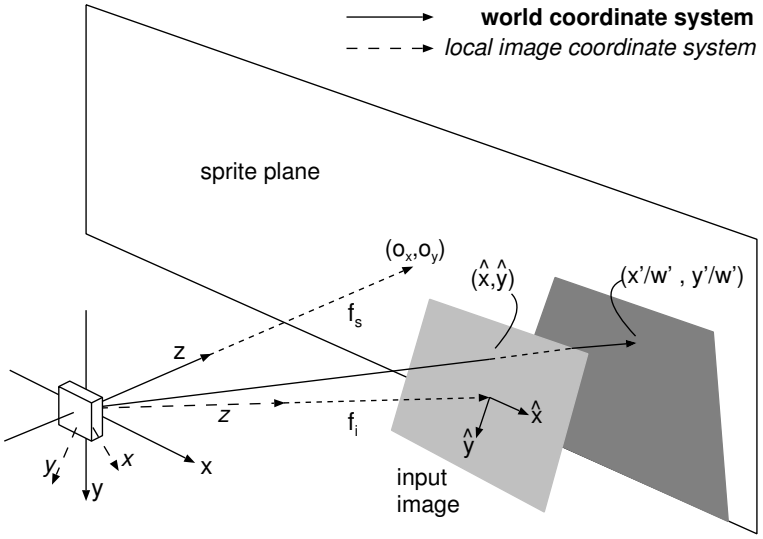


Figure 5.5: *The rotating camera is located at the origin of the world coordinate system. The sprite plane is assumed to be orthogonal to the z-axis. Input images are at a distance to the origin that is equal to the focal length when the image was taken. A point $(\hat{x}, \hat{y})$ on the image is projected onto the sprite position $(x'/w', y'/w')$.*

One example result of aligning several frames from the *stefan* sequence into a reference frame is presented in Figure 5.7.

5.2.3 Long-term motion estimation

The computation of the image-to-mosaic transforms by concatenating inter-image transforms is subject to a practical problem. The parameters that we obtain from the feature-based motion estimator have small inaccuracies that are negligible when only pairs of images are considered, but which can accumulate when many transforms are concatenated. As a consequence, images that are temporally far apart will not fit together in a composed background image. For a straight camera pan, this effect is almost invisible, since only successive images overlap. However, if the camera motion returns to a previous position after some time, the accumulated errors might be well visible (see Figure 5.8(a)).

To prevent this accumulation of parameter errors, we refine the image-to-mosaic transforms with a high-accuracy motion estimation algorithm, where the concatenation of inter-image transforms is used to initialize the

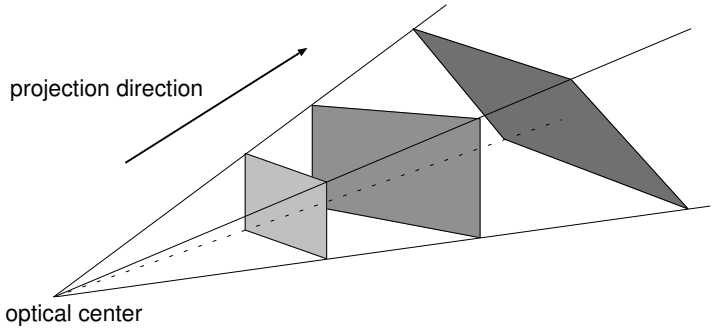


Figure 5.6: *The placement of the background plane can be chosen arbitrarily, because an image on each of these planes will look identical, seen from the optical center.*

optimization.

Accurate alignment with direct estimation methods

In contrast to the feature-based motion estimation algorithms that estimate the parameters from a set of corresponding point features, *direct methods* estimate these parameters directly by minimizing the motion-compensated residual image. This makes use of the *brightness-constancy constraint*, which states that the brightness of pixels does not change during motion. In other words, this means that if $\bar{\mathbf{H}}_{t+1,t}$ is the true image motion, then $|I_{t+1}(\bar{\mathbf{H}}_i\mathbf{p}) - I_t(\mathbf{p})| = 0$.

Let us denote the background image as $I_B(x, y)$ and the current frame t as $I_t(x, y)$. In the direct motion estimation algorithm, the refined motion parameters $\breve{\mathbf{H}}_t$ are computed by minimizing the motion-compensated image difference

$$\breve{\mathbf{H}}_t = \arg\min_{\mathbf{H}_t} \sum_{\mathbf{p}\in\mathcal{A}} |I_B(\mathbf{H}_t\mathbf{p}) - I_t(\mathbf{p})|^2, \tag{5.4}$$

where $\mathcal{A}$ denotes the complete image area of I_t. Since the number of parameters is large and the function to be minimized is non-linear, an iterative gradient-descent algorithm has to be applied to find a solution. A gradient-descent algorithm does not guarantee to find the global optimum, since it usually converges to a local optimum near the initialization. Hence, it is important to start with a good initialization, which we can in our case get from the motion parameters of the feature-based motion estimator.

Practically, we cannot use Eq. (5.4) in the presented form, because of three reasons. First, during the construction process, the complete background image is not available yet, which means that some of the trans-

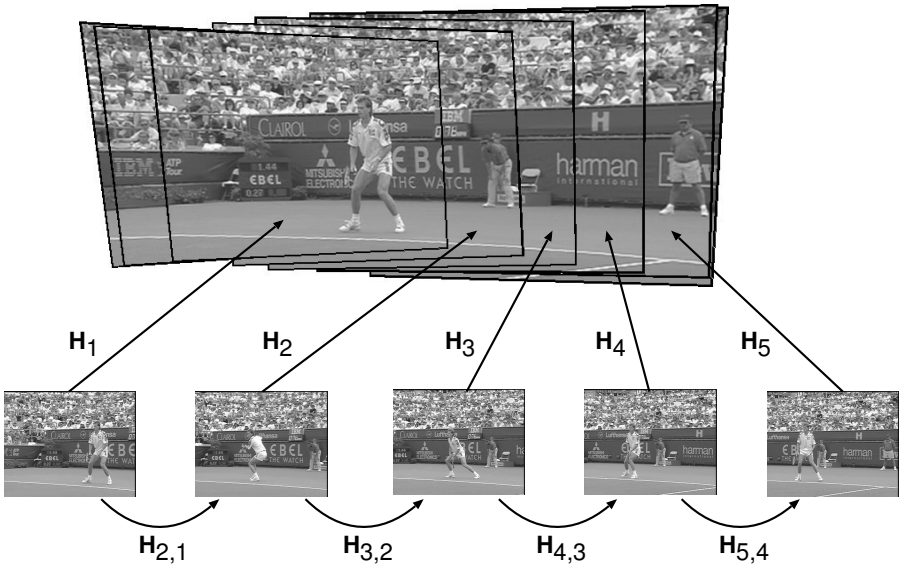
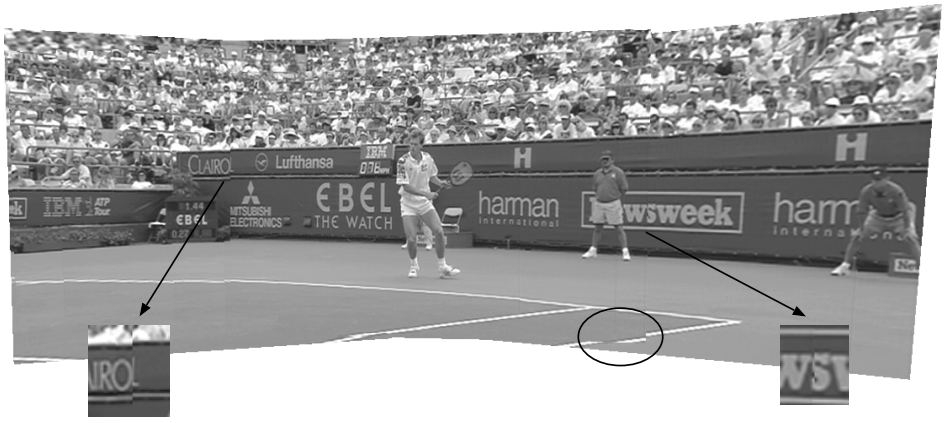


Figure 5.7: *Alignment of input frames into a common reference frame. The inter-image transforms* $\mathbf{H}_{i+1,i}$ *are obtained from the feature-based motion estimator. The input-to-mosaic transforms* $\mathbf{H}_i$ *can be obtained by a concatenation of inter-image transforms as soon as one reference transform* $\mathbf{H}_r$ *has been defined.*

formed pixels $\mathbf{H}_t\mathbf{p}$ will fall onto pixels which are still undefined. It is not sufficient to set all undefined pixels in I_B to an arbitrary color value, since this can cause large matching errors if the input image I_t has a different color. As a consequence, the optimization would try to squeeze I_t onto the already defined area of I_B as much as possible. A solution is to introduce a special background pixel value that denotes *transparent* pixels and set the matching error between any value and *transparent* pixels to zero. Of course, this also means that we get the minimum matching error if both images do not overlap at all, but since the optimization algorithm locks to the nearest local minimum, this unfavourable solution is not reached.[2]

A second difficulty is that our input sequences usually do not show pure

[2]At first view, both approaches seem to be symmetric. If the cost of non-overlapping pixels is high, the optimization will move the image onto the defined area to decrease the error. On the other hand, if the cost of non-overlapping pixels is zero, the optimization will move the image off the defined area to decrease the error. However, the difference is that we start with a good initialization where most of the residuals are low. Hence, if we set the cost in undefined areas to zero, the error does not change much if pixels move off the defined area.

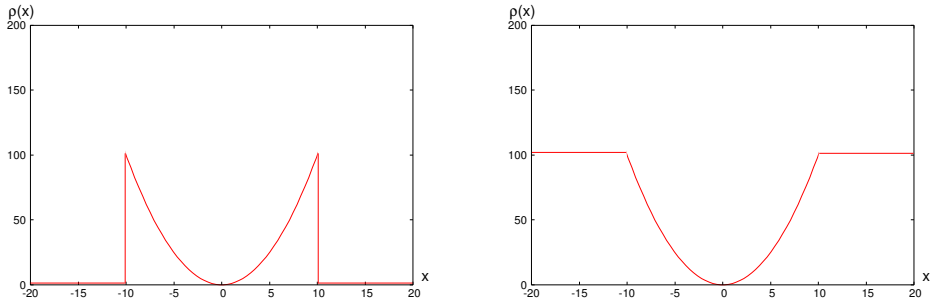


(a) Alignment using a concatenation of short-term motion parameters. Note the alignment errors, which are especially visible at the court line.

(b) Alignment using long-term motion parameters. The long-term parameters were determined as a refinement of the transform between the image and the mosaic.

Figure 5.8: *Alignment quality using short-term or long-term motion parameters. Only every* 12^{th} *input image has been included in the mosaic to make the misalignments more visible.*

background images, but they also contain foreground objects. These can create large matching errors which would dominate the optimization and therefore bias the estimation towards an inaccurate solution. The solution for this is to use an M-estimator instead of the squared error to limit the effect of non-matching areas. We use a saturated squared-error function which does not increase if the pixel difference exceeds a threshold τ. Other authors [171, 39] propose to use a clipped squared-error function, which drops to zero if the difference exceeds the threshold (Fig.5.9(a)). However, we think that this can have the unfavourable effect that a bad match can



(a) Ignore outliers. (b) Limit error to maximum.

Figure 5.9: *Robust M-estimators.*

result in a low matching error, especially with high-contrast backgrounds.

The third problem of Eq. (5.4) is that the transformed pixel positions $\mathbf{H}_t\mathbf{p}$ will usually not be integer positions, whereas the background image is discretized. Here, it is important that the transformed positions are not just rounded to the nearest-neighbor pixel, since this would degrade the sub-pixel accuracy of the solution. We applied a bi-linear interpolation[3] on I_B to obtain values also at sub-pixel locations.

Also note that the total matching error is not normalized to the mapped image size. This means that a smaller image would lead to less error. However, adding the normalization would make the equations for the optimization process far more complex. Fortunately, the behaviour of gradient-descent algorithms to lock to a local minimum is again to our benefit, since it prevents that the projected area collapses to a small size (Fig. 5.10).

Introducing the above-mentioned considerations into a modified cost function, we obtain the modified optimization equation

$$\breve{\mathbf{H}}_t = \arg\min_{\mathbf{H}_t} \sum_{\mathbf{p}\in\mathcal{A}} \rho(I_B(\mathbf{H}_t\mathbf{p})\,,\, I_t(\mathbf{p})), \tag{5.5}$$

where $I_B(\mathbf{H}_i\mathbf{p})$ is evaluated using bilinear interpolation and the M-estimator is defined as the clipped squared error

$$\rho(y_B, y_I) = \begin{cases} 0 & \text{if } y_B \text{ is } \textit{transparent}, \\ (y_B - y_I)^2 & \text{if } |y_B - y_I| < \tau, \\ \tau^2 & \text{else.} \end{cases} \tag{5.6}$$

[3]Bi-linear interpolation was chosen because this complies to the sprite-warping process as defined in the MPEG-4 standard.

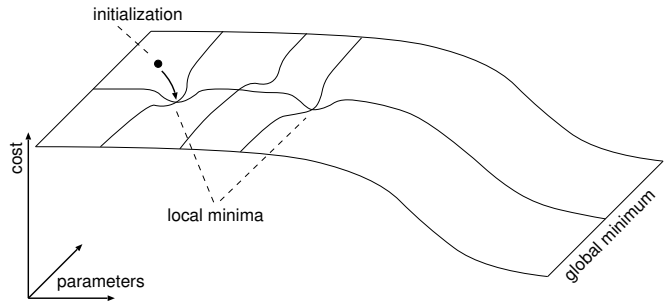


Figure 5.10: *Cost function of an optimization problem. Often, cost functions are defined such that the global minimum corresponds to a degenerate solution. For example, in our motion-estimation problem, we minimize the luminance difference in the overlapping image region. Consequently, the global solution would be to place the two images beneath each other, resulting in zero cost since there is no overlapping area. In these cases, it is an advantage to reach the nearest local optimum instead of the global solution.*

Parameter optimization

Gradient descent algorithms optimize the parameters with a number of iterations. Note that to evaluate the cost function, we have to transform the input image using a projective transform with bilinear interpolation and compare the result with the background image. Since this computation-intensive calculation has to be carried out in each iteration, it is important to keep the number of iterations low. For this reason, we apply the Levenberg-Marquardt minimization algorithm which is more complex than a steepest descent, but which is known to converge in a small number of iterations. We do not describe the Levenberg-Marquardt algorithm here (refer to [151, 179] for an in-depth description), but we show how to apply it to our motion estimation problem.

Let us write the transform parameters in vector form as $\boldsymbol{\theta} = (h_{00}, \ldots, h_{21})$. For the optimization process, the algorithm requires the gradient vector of the cost function

$$E_{\boldsymbol{\theta}} = \sum_{\mathbf{p} \in \mathcal{A}} \rho(I_B(\mathbf{H}_t \mathbf{p})\,,\, I_t(\mathbf{p})) \tag{5.7}$$

with respect to the optimization parameters, i.e., $\nabla E = \partial E_{\boldsymbol{\theta}} / \partial \boldsymbol{\theta}$ and the Hessian matrix $\nabla^2 E$. To get the expressions for the gradient vector, we substitute the motion model Eq. (2.10) into Eq. (5.5). Let us for the ease

of notation abbreviate the residual for a single pixel k as

$$e_k = I_B(\mathbf{H}_t \mathbf{p}_k) - I_t(\mathbf{p}_k) = I_B(x', y') - I_t(x, y). \tag{5.8}$$

Then we can obtain the derivatives for the case where $\rho(y_I, y_B) = |y_I - y_B|^2$ as

$$\nabla E = 2 \sum_k e_k \cdot \left(\frac{\partial e_k}{\partial \theta_1}, \cdots, \frac{\partial e_k}{\partial \theta_8} \right). \tag{5.9}$$

To compute the derivatives of e_k, we apply the chain rule to get, for example,

$$\frac{\partial e_k}{\partial \theta_1} = \frac{\partial e_k}{\partial h_{00}} = \frac{\partial I_B}{\partial x'} \frac{\partial x'}{\partial h_{00}} = \frac{\partial I_B}{\partial x'} \frac{x}{D} \tag{5.10}$$

or

$$\frac{\partial e_k}{\partial \theta_7} = \frac{\partial e_k}{\partial h_{20}} = \frac{\partial I_B}{\partial x'} \frac{\partial x'}{\partial h_{20}} + \frac{\partial I_B}{\partial y'} \frac{\partial y'}{\partial h_{20}} = -\frac{y}{D} \cdot \left(x' \frac{\partial I_B}{\partial x'} + y' \frac{\partial I_B}{\partial y'} \right) \tag{5.11}$$

with the abbreviation $D = h_{20}x + h_{21}y + 1$.

For the computation of the Hessian matrix, we start with Eq. (5.9), which we derive a second time. To simplify the result, we follow the suggestion in [151] to ignore the second-order derivatives of e_k^2

$$\frac{\partial^2 e_k^2}{\partial \theta_m \cdot \partial \theta_n} = 2 \left(\frac{\partial e_k}{\partial \theta_m} \cdot \frac{\partial e_k}{\partial \theta_n} + \underbrace{e_k \frac{\partial^2 e_k}{\partial \theta_m \cdot \partial \theta_n}}_{\text{ignore}} \right) \approx 2 \frac{\partial e_k}{\partial \theta_m} \cdot \frac{\partial e_k}{\partial \theta_n}, \tag{5.12}$$

because near the optimum, the errors e_k can be assumed to be distributed around zero. Hence, these terms will cancel out to a large extent when summing over all pixels. Moreover, since the Hessian matrix is only used to compute the direction of search, the optimization process is very tolerant to inaccuracies in the Hessian matrix (this is similar to Quasi-Newton approaches). Consequently, we compute the Hessian matrix $\nabla^2 E$ from the entries of the gradient vector as

$$(\nabla^2 E)_{mn} = 2 \cdot \sum_k \frac{\partial e_k}{\partial \theta_m} \cdot \frac{\partial e_k}{\partial \theta_n}. \tag{5.13}$$

Image alignment process

The complete background image is constructed by computing refined motion parameters $\breve{\mathbf{H}}_t$ between each input image and the background image, while the background image is updated with the new image after each step.

In the update of the background image, it is important that only the background pixels that have previously been *transparent* are replaced in the background to prevent a slow drift over time.

The construction process is as follows:

1. Copy the reference image I_r into the initially transparent background image I_B using the identity transform $\breve{\mathbf{H}}_\mathbf{r} = \mathbf{1}$. Set the next image t to be processed to $t = r + 1$.

2. Calculate optimized motion parameters $\breve{\mathbf{H}}_t$ for the current frame using the prediction $\breve{\mathbf{H}}_{t-1}\mathbf{H}_{t,t-1}^{-1}$.

3. Add I_t to the background I_B where only previously *transparent* pixels are modified.

4. Proceed to the next image $t := t + 1$ and continue at Step 2 until all images are processed.

Images t that are before the reference r, i.e., with $i < r$ are added similarly in a second pass.

An example result of the long-term parameter estimation is shown in Figure 5.8(b). It is visible that the accumulated errors from the short-term esimations have been corrected to a globally consistent background image.

5.3 Background estimation

The background image that was synthesized during the alignment process was obtained by simply copying the input frames into the background image. Consequently, the background image will still contain foreground objects. In this section, we will recompute the background image with the intention to remove the foreground objects and to get a pure background image. An example is shown in Fig. 5.8(b), which is actually the output of this background estimation step. Note that the foreground object has been removed, unlike Fig. 5.8(a).

5.3.1 Introduction and previous work

Background estimation is the problem of obtaining an image of the scene background from sequences where the background might be occluded by foreground objects during most of the time. Since there is no *a-priori* knowledge how the background looks like, it can only be estimated by observing the scene for a longer time and by considering anything as foreground that is not static. Note that this definition of background may differ from the usual intuitive meaning, which is very dependent on the context. For example, in a tele-conferencing scene with several participants in a meeting-room, we would consider the meeting-room to be the background, while the humans are foreground. Now consider that at the wall there is a clock, which apparently changes its appearance over time. Based on our intuition, we would still classify the clock as background, but in the sense of image background estimation, the clock would be foreground.

In many cases, the decision is even more difficult to make, or the decision can depend on the time-interval during which we observe the scene. For example, assume that we can see some book shelves at the back of our meeting-room. Normally, we would consider this shelf to be background. However, if one of our persons is removing a book from the shelf, this book is suddenly becoming a foreground object. Further difficulties can arise from gradual environmental changes like the direction of the sunlight or sudden changes when someone switches on a light. A good survey of the problems of background estimation can be found in [186].

Basically, we can identify two classes of background-estimation problems which arise from different applications. The difference between both is the duration for which we observe the scene.

- **Long-term.** In surveillance applications, we have long-term observations, where a scene is continuously recorded. Background-estimation algorithms for this application must use an update strategy, since the

(a) Constant update factor. (b) Motion adaptive update.

Figure 5.11: *Iterative background estimation for hall-and-monitor sequence up to image 120.*

number of available frames is too large to consider them all at once. A typical requirement for these long-term observations is that the background image in fact is not static, but also reflects a gradual change of the scene. Typical problems for this class of algorithm are sudden changes of the background or changes of the semantic meaning as described above.

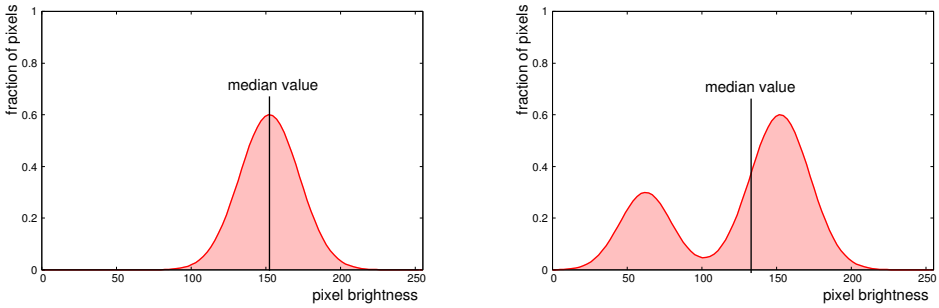
- **Short-term.** In a video-analysis application, where the video content includes movies or home-videos, we usually have only short scenes from which we want to create a background image. The main problem in this application is that the video sequence may be very short and yet the algorithm has to find the best possible solution for this limited input. Fortunately, the scenes are usually so short that we do not have to take gradual changes into account.

Iterative update algorithm

Since most algorithms proposed in the literature have surveillance applications in mind, they use a frame-based update strategy that maintains a current background estimate I_B which is updated iteratively with each new input frame I_t. In the simplest case, the update can be made with a constant update factor α as

$$I_B := \alpha I_t + (1 - \alpha) I_B. \tag{5.14}$$

The disadvantage is that slowly moving objects will appear in the background image as *shadows* (Fig. 5.11(a)). It has also been proposed [18,



(a) Without foreground object. (b) With foreground object.

Figure 5.12: *A foreground object adds a bias to the median filter, such that it will not output the mean background color.*

148, 156] to adapt the aging factor to the amount of motion in the input image to reduce the influence of moving objects. More specifically, the update factor is reduced if the frame-to-frame difference or the frame-to-background difference is high. However, the problem is that because there is no distinction between foreground and background, the update factor is not only reduced when a foreground object covers the background, but also when the background is uncovered again. Consequently, it also takes a longer time to remove erroneous objects from the background (Fig. 5.11(b)). This class of algorithms can only work reliably when foreground objects do not move too slowly and background is visible during most of the time. Otherwise, the reconstruction stays unstable or converges to an average of foreground and background color.

Temporal median algorithm

A different approach that works better for short image sequences is to use a temporal median-filter over all input frames [123]. This guarantees that the background color is correctly found if the background was visible for more than half of the time. The disadvantage of the algorithm is that a large number of input frames have to be stored. But note that the median value can be computed efficiently by computing the pixel histogram over time and deriving the median value from that histogram.

An advantage of the median-filter algorithm is that the effect of blurring is not as severe as with the weighted update algorithms. However, even with the median-filter algorithm, the background image content can deviate from the correct background color. To understand this, assume that a bright background is occluded for some period by a dark object. Since the median is computed including the dark foreground object pixels, as well as the

bright background pixels, the median will not be at the mean background luminance, but it will be shifted to slightly darker values, caused by shadows or image noise (see Fig. 5.12). The effect can be observed in the results shown in Fig. 5.21(d). Even though all foreground objects are dark and the background is bright, objects appear half-transparent in the reconstruction. At first glance, this should not happen since the median filter should either select the darker foreground or the bright background. However, because of image noise or because there has been a shadow on the background, darker pixels occur and the foreground objects give a bias to the estimation such that darker pixel values are selected. This bias from the correct background color is not always perceivable, but it can cause difficulties in automatic segmentation algorithms, because this bias might already be detected as a significant change.

Pixel mode algorithm

Another algorithm that follows the same approach as the median algorithm is the *mode* algorithm. Instead of computing the median of the pixel brightness over time, we select for each pixel the most frequently occurring color. At first view, this seems to be a good solution, but the problem lies in the definition of pixel similarity. If we count the number of occurrences of a specific luminance, we have to allow for some tolerance of the luminance. Otherwise, effects like quantization from an optimally preceding compression step might influence the result.

Manual synthesis

For the purposes of comparison and ground-truth generation, we wrote a tool that simplifies the manual extraction of the background from a video sequence. The program presents the input sequence to the user and lets him navigate through the sequence. When the user marks an area in a specific input frame, this area is copied into the background image. To simplify this process, the image is divided into small blocks, such that the complete block is copied to the background image when the user selects it.

5.3.2 The *SimMat* background-estimation algorithm

In this section, we discuss a new background-estimation algorithm that does not show the luminance bias as the median algorithm does, and that provides robust results even for difficult scenes. This new algorithm (*SimMat*) also succeeds in reconstructing the background image for areas that are visible in less than half of the frames. It achieves this by integrating

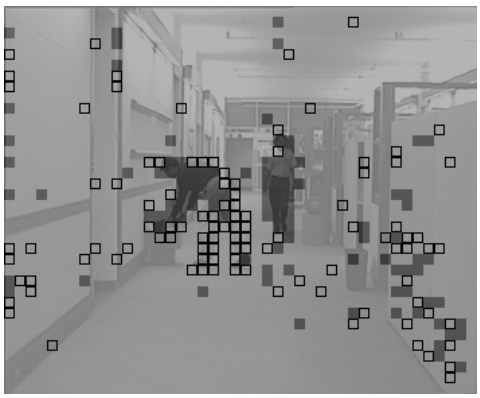


Figure 5.13: *Sample result of automatic block classification into the three classes (a) static background, bright areas; (b) moving foreground, dark areas; (c) static foreground, black squares. It is visible that the legs of the left human are static, while his upper body is moving.*

contextual information from neighboring areas, for which the decision is easier, and by also considering motion information.

When we examine typical video sequences, we can see that areas in each of the images can usually be assigned to one of three classes: static background, moving foreground, and static foreground. In this context, the terms *static* and *moving* are related to only a single frame. For a foreground object, the classification can change between moving foreground and static foreground during the sequence, since the object can move in some images, but it can also stay at the same position during some other time. It can even happen that parts of the object are in different classes in the same image, like a human that is standing still, but waving with his hands. However, foreground objects never belong to the static background class. An illustration of the three classes for an example picture can be found in Fig. 5.13.

To reconstruct the background image, it is required that we can detect these three classes. While it is not difficult to detect moving objects, the differentiation between static background and static foreground is the main problem. To help in this classification, we can further use the following two assumptions about backgrounds:

- A background never changes its appearance, even if it was occluded by a foreground object for some time. This means that the background pattern will be visible repeatedly without change, while a foreground object might appear static for a limited time, but not on a global

scale.

- If an image area is occluded by a foreground object for some periods, it is probable that a neighboring area is occluded for comparable periods (see Fig 5.17).

Algorithm overview

The principal idea of our algorithm is as follows. First, we apply a rough classification of the data in the input images into the two foreground classes and the background classes. Afterwards, the background image is synthesized from typical representatives of the background class data. Since foreground data is not considered in the background reconstruction, any bias towards the foreground color is prevented. Moreover, small classification errors do not lead to errors, because only one typical representative of the class is selected, so that outliers do not have much influence.

The classification is carried out on units of small blocks of about (8×8 pixels) to reduce the computational cost and to make the classification between background and foreground more robust. Periods in which a block shows background content are identified by searching for the subset of frames in which the block shows a stable content. The similarity of the contents of a single block over time is collected into a *similarity matrix* $\mathbf{M}$, which contains the difference $\mathbf{M}_{a,b}$ between the image content in this block for each pair of frames (a, b). High values correspond to similar content, whereas low values are found for each pair of frames that contains differing content. Every subset of frames $\mathcal{T}$ induces a decomposition of the matrix into elements that correspond to pairs of frames, where both frames are within $\mathcal{T}$, and entries where at least one corresponding frame is not in $\mathcal{T}$ (see Fig. 5.14). Our goal is to find a subset $\mathcal{T}$ that includes all the frame-numbers for which background content is visible in a specific block. For a good solution, the sum of the matrix elements that are covered by $\mathcal{T}$ should be large, since these entries correspond to static background. On the other hand, entries that are not covered by $\mathcal{T}$ should show considerably smaller similarity. This criterion is used in an optimization to find for each block the subset of background frames that has the most static content.

Block similarity matrices

We begin with presenting the definition of a block similarity matrix. Let the block size be $N \times N$ pixels and let the input sequence be of length L. Furthermore, let $I_t(x, y)$ be the luminance of pixel (x, y) in input frame t. In this section, we assume that $I_t(x, y) \in [0; 1]$. To simplify the notation,

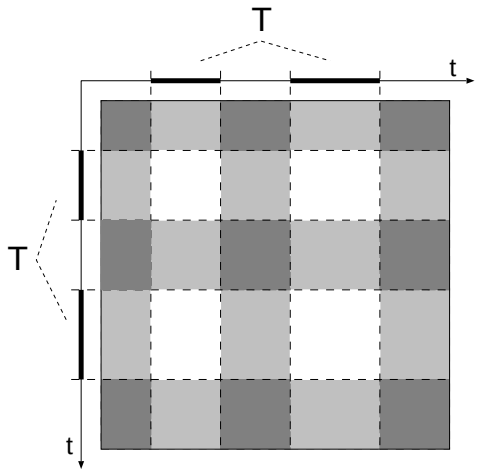


Figure 5.14: *A set of frames $\mathcal{T}$ induces a decomposition of the matrix into entries $M_{a;b}$, where $a, b \in \mathcal{T}$ (drawn in white), and entries $M_{a,b}$, where either a or b is not in $\mathcal{T}$.*

we assume in the following that the camera motion has been compensated as described previously. For each block (u, v) with top left pixel at position (uN, vN), we calculate a symmetric similarity matrix $\mathbf{M}^{(u,v)}$ of size $L \times L$ with

$$\mathbf{M}_{a,b}^{(u,v)} = 1 - \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \big| I_a(uN+i, vN+j) - I_b(uN+i, vN+j) \big|. \quad (5.15)$$

This equation states that each matrix element $\mathbf{M}_{a,b}$ is set according to the *sum of absolute differences* (SAD), measured between the blocks in frame a and frame b at the same block position in the image.

For time periods in which the content in the block does not change, a square block along the matrix diagonal will contain high values (Fig. 5.15). If a specific block content disappears for some time and reemerges later, a corresponding rectangle of matrix elements beneath the matrix diagonal will show low values. Periods with moving content show as low-valued matrix elements. If the content is only visible for a short time, the corresponding matrix rows and columns will contain mostly low values.

Matrix decomposition

To identify the periods in which only background is visible in a block, we decompose the matrix into two parts: the stationary elements (high

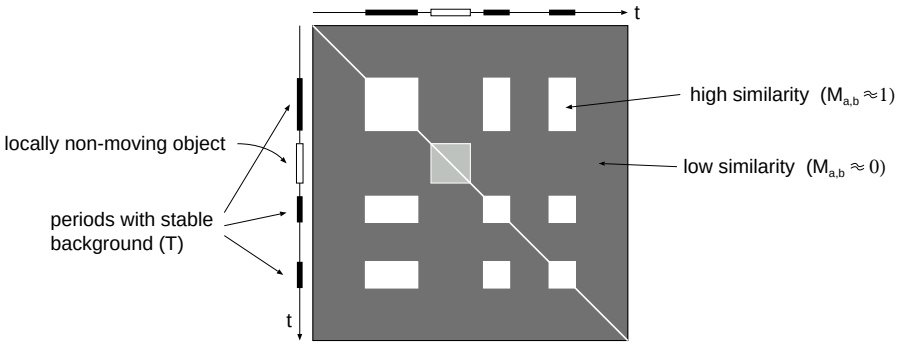
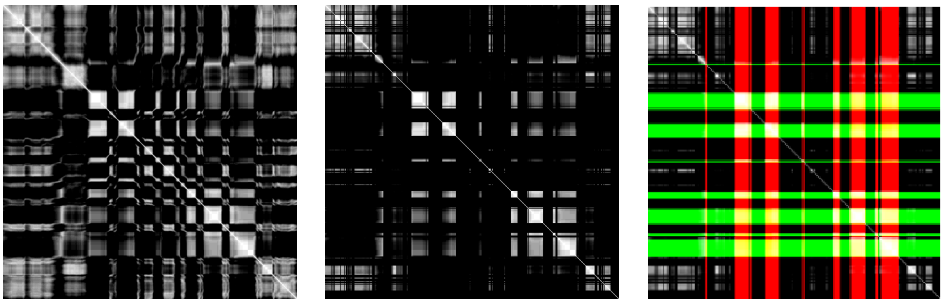


Figure 5.15: *Structure of a block similarity matrix* $\mathbf{M} = \{\mathbf{M}_{a;b}\}_{a;b}$. *White matrix elements indicate a high block similarity, while dark elements show low similarity pairs.*



(a) Sample similarity matrix.

(b) Same matrix after invalidating periods with motion.

(c) Decomposition result.

Figure 5.16: *A sample block similarity matrix taken from the sequence shown in Fig. 5.21.*

similarity values), and the non-stationary elements (low similarity values). Let $\mathcal{T}^{(u,v)} \subseteq \{1,\ldots,L\}$ be the set of frames for which we assume that block (u,v) only contains background content[4]. Because the background is static, we consider a matrix element $\mathbf{M}_{a,b}$ stationary iff a and $b \in \mathcal{T}$, i.e., background is seen in frame a and frame b.

Since stationary matrix elements should have large values and non-stationary elements should have small values, we can separate them by choosing $\mathcal{T}$ such that the stationary elements are as large as possible and the non-stationary elements are as small as possible. More specifically, we

[4]We will omit the superscript (u,v) to simplify notation when the meaning is clear. Since we are only considering single blocks in this section, no ambiguities will occur.

optimize the objective function C

$$\max_{\mathcal{T}} C = \max_{\mathcal{T}} \underbrace{\sum_{a,b\in\mathcal{T}} \mathbf{M}_{a,b}}_{\text{stationary elements}} + \underbrace{\sum_{a\notin\mathcal{T}\,\vee\, b\notin T} (1-\mathbf{M}_{a,b})}_{\text{non-stationary elements}}. \tag{5.16}$$

Optimization is carried out using an iterative process. Starting with a good estimate of $\mathcal{T}$ (obtaining a good initialization is described later), we calculate the difference that results from adding or removing each of the input frames from $\mathcal{T}$. If the objective function can be increased by adding or removing a frame, $\mathcal{T}$ is modified accordingly. Optimization is stopped when C cannot be increased further. We have found that this process converges in only about two or three passes over the input frames. Note that instead of the naive way of computing the objective function by summing over the complete matrix, it is sufficient to compute the difference, which can be obtained by summing only over a single matrix row. In the case of adding a frame k to $\mathcal{T}$, the difference is

$$\begin{aligned}\Delta C_{+k} &= 2\Bigg(\underbrace{\sum_{a\in\mathcal{T}} \mathbf{M}_{a,k} + \sum_{a\notin\mathcal{T}} (1-\mathbf{M}_{a,k})}_{\text{new costs}} - \underbrace{\sum_{a\in\{1,\ldots,L\}} (1-\mathbf{M}_{a,k})}_{\text{old costs}}\Bigg) \\ &= 2\Bigg(\sum_{a\in\mathcal{T}} (2\mathbf{M}_{a,k}-1)\Bigg).\end{aligned} \tag{5.17}$$

Clearly, for the case of removing a frame k from $\mathcal{T}$, the difference is just the negative value $\Delta C_{-k} = -\Delta C_{+k}$.

Since the above matrix decomposition process converges to a local minimum close to the initialization, an initialization near the correct minimum must be chosen. Note that the global minimum need not necessarily correspond to the correct background periods. If the sequence contains many foreground objects of the same color, and if the objects are visible during most of the time, the global optimum can correspond to those periods in which foreground objects are visible. To reduce this problem, we apply two additional steps, preceding the optimization step. First, we exclude periods from $\mathcal{T}$ for which we observe motion in the block. Since we assume that camera motion has been compensated beforehand, moving content cannot occur in background regions. Second, we exploit the correlation of background periods between neighboring blocks. Both steps are described in the next two sections.

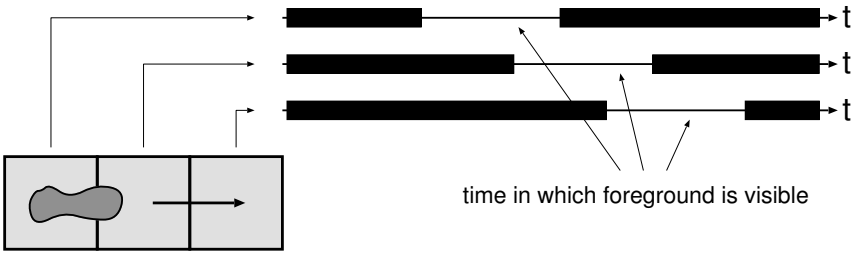


Figure 5.17: *A foreground object moves across three blocks in the image. The times during which the object is visible in the blocks are correlated.*

Integration of motion information into the similarity matrix

Motion estimation is carried out for each block using a block-matching algorithm in a small neighborhood. If the minimum block matching error is lower than 90% of the null-vector matching error, the block is considered as moving and the matrix row and column corresponding to the current input frame are artificially set to 0. This prevents the optimization algorithm from selecting the block in this frame as a background block. Figure 5.16 shows an example how this exclusion disambiguates an otherwise unclear situation.

Initializing the optimization by background periods prediction

If there is object motion visible in a block, it will most probably also be present in a neighboring block during a comparable time period (see Fig. 5.17). Hence, when calculating $\mathcal{T}^{(u,v)}$, we use the previously calculated $\mathcal{T}^{(u-1,v)}$ and $\mathcal{T}^{(u,v-1)}$ to initialize the optimization process. If an input frame a is contained in $\mathcal{T}^{(u-1,v)}$ and $\mathcal{T}^{(u,v-1)}$, it is also included in $\mathcal{T}^{(u,v)}$. If it is only contained in one of both, it is decided randomly whether to include it. At the left and top border, predictions are formed directly from the solution of the block above or to the left, respectively. The very first block (top-left) is initialized with all input frames active in $\mathcal{T}^{(0,0)}$. This is a sensible assumption, since image activity is usually centered in the image such that the border contains mainly background content.

The spatial prediction scheme has two advantageous properties. First, it provides an accurate initialization of the optimization, leading to fast convergence. Second, the prediction helps to select the correct local minimum, even when the object is visible for a longer time than the background. Since the prediction provides the initialization, even a strong minimum has not enough support in the beginning that the optimization could be at-

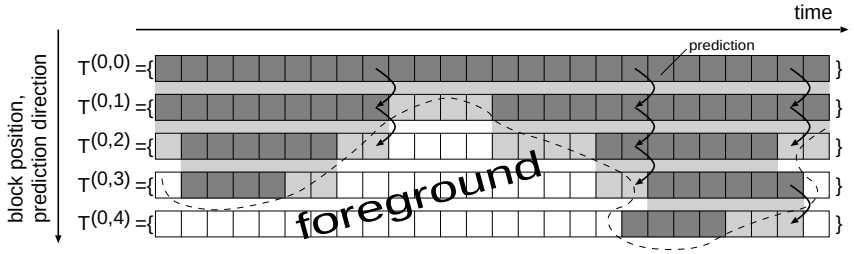


Figure 5.18: *Spatial background-period prediction (first column of blocks in the input images). The block at the top left $T^{(0,0)}$ contains background content throughout the sequence (background marked in a dark shade). The background periods of $T^{(0,i)}$ form the initialization for background periods of $T^{(0,i+1)}$ (prediction is drawn in a light shade). The matrix decomposition step then refines this prediction to get the final result for this block. Since optimization is started with the last block's result, the optimization will converge to the correct minimum even for blocks that are clearly dominated by foreground objects (e.g., $T^{(0,4)}$).*

tracted to it. This is illustrated in Figure 5.18 and a real-world example is shown in Fig. 5.19.

Using *SimMat* for background updating

The algorithm described so far was presented as an offline algorithm that is started when all input frames are available. In fact, this is the way the algorithm is used in our segmentation system. However, as noted previously, surveillance applications usually update the background during the observation to adapt to changing illumination and other changes in the background. If the *SimMat* algorithm should be used for this application, then it can be done with a simple modification.

Instead of using similarity matrices of size $L \times L$, where L is the sequence length, we set L to a history size. This is the number of past frames that are considered in the background reconstruction. Note that this need not be a continuous stream of frames, but it can also consist of only, e.g., every 10^{th} frame to same memory and computation time.

Whenever a new input frame t is added to the background estimation, we modify the row and column (t mod L) in the matrix. This means that the matrix is filled frame by frame and it cyclically starts over without deleting the previous content when the matrix is full. After the matrices are modified, one pass of the optimization algorithm is carried out to adapt

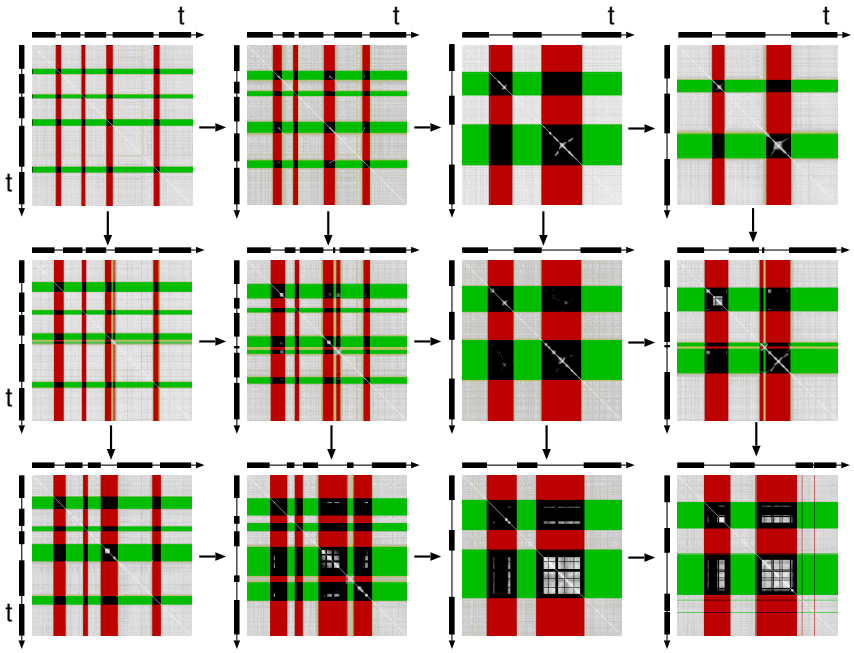


Figure 5.19: *Prediction of foreground time periods between adjacent blocks.*

the background classification $\mathcal{T}$.

An example result of background updating with *SimMat* is shown in Fig. 5.24 and it will be described in the following section.

5.3.3 Results

We have applied our algorithm to a variety of popular test sequences like the *hall-and-monitor*, *road1*, *road2*, or *urbicande* sequences. For these sequences, the background could be reconstructed without any visible errors. Even the background from *seq_17* of the Video Quality Expert Group (VQEG) test set was recovered without error (see Fig. 5.23). To discuss the properties of the algorithm in comparison to other algorithms, we will evaluate some scenes in more detail.

The *queue* sequence

To see the limits of the reconstruction algorithms, we applied our algorithm to a very difficult sequence (see Fig. 5.21) containing many people, where some persons are walking around and some are standing still for a long time. We carried out background reconstruction with all described

algorithms. Because parts of the background are never visible during the whole sequence, it is impossible to get a complete background reconstruction even with a manual synthesis (Fig. 5.21(b)). Some background areas in the center of the image are visible only for a very short time, such that the *iterative update*, *median*, and *mode* algorithms all fail to obtain a good reconstruction. Compared to these algorithms, the *SimMat* algorithm is able to recover larger areas of the background. Furthermore, it is visible that for the *median* algorithm (and even worse for the *iterative update* algorithm), there is a strong bias towards the dark foreground object color. This bias is not present with the *SimMat* algorithm, which reconstructs the background without any blurring.

The *road1* sequence

The *road1* sequence shows a traffic scene, where a car on the right lane slows down and finally stops (Fig. 5.22). We synthesized two background images for the *median* and the *SimMat* algorithm, respectively. One background image is computed for frames 1-150, while the other one is computed for the whole sequence (frames 1-300). Since the car slows down and finally stops in about frame 200, the *median* algorithm cannot decide clearly if the car should belong to the background or not. The result is a blurred region. The *SimMat* algorithm does not have this problem and it adapts the background image almost instantaneously. Consequently, we have a background image without the car if we consider the sequence up to frame 150, and a background image with the car for the whole sequence. Notice that the *median* algorithm also failed to remove the cars at the end of the street completely. However, applied to the whole 300 frames, the *median* algorithm provided a similar result as the *SimMat* algorithm.

The *hall-and-monitor* sequence

To further evaluate the behaviour for changing backgrounds, we applied the *median* and the *SimMat* algorithms also to the *hall-and-monitor* sequence. Figure 5.24 depicts the background images that were obtained after every 50 frames. For better visibility in the figure, the images have been cropped to the most important part. To compute the background images, a history size of 150 frames was used, but only every 5^{th} frame was used for the computations. The interesting point in this sequence is the bag that the left man is placing on the pedestral, and the monitor that the right man is carrying away. These objects change their status during the sequence to background or foreground, respectively. The left column in the figure shows the input frame, the middle column the output of the *median* algorithm,

	PSNR
iterative update	29.86 dB
median	30.89 dB
median (cropped)	32.02 dB
our algorithm	35.15 dB
camera noise	38.74 dB

Table 5.1: *PSNR between reconstructed background and real background (hall-and-monitor sequence).*

and the right column the *SimMat* algorithm. It is well visible that the median algorithm has difficulties to keep a clean background without the two men. Since the men are walking in the direction of the optical axis, they stay at the same position for a long time. Consequently, they appear partly in the background reconstruction. The same holds for the bag, which appears gradually on the pedestral. The *SimMat* algorithm does not show this problem. The background stays clean, and it adapts the background after some time to accomodate for the bag on the pedestral.

Quality of the reconstructed background

To evaluate the quality of the background image with respect to applicability for automatic segmentation algorithms, we measured the difference between the reconstructed background image and the ground-truth background image (Table 5.1). Since the real background image is only available for the *hall-and-monitor* sequence (in the first frames of the sequence), we used this sequence to obtain the results. We measured the PSNR of several reconstruction algorithms and estimated the camera noise by calculating the PSNR between the first two frames of the sequence. Because the median algorithm cannot remove the foreground objects completely, we calculated the PSNR a second time, now with the erroneous regions excluded. Even with these regions excluded, our algorithm achieves considerably higher PSNR than the median algorithm, which makes it a better choice for segmentation applications.

5.4 Summary of the background reconstruction module

This chapter presented the processing steps that are required to synthesize a background sprite image from a video sequence, when approximate models

for the camera parameters are already available (see Chapters 3 and 4). These two main processing steps are

- obtaining the accurate motion models to align all input images into the common background image, and
- fusing the input images to remove moving foreground objects from the background image.

The framework of these processing steps is illustrated in Figure 5.20. As input, we apply the approximate parameters $\mathbf{H}_{t,t-1}$ for describing camera motion between two frames $t-1$ and t. These parameters are refined and converted to frame-to-mosaic motion parameters with a long-term motion estimation algorithm that also provides a high accuracy. After each step, the accurate parameters $\check{\mathbf{H}}_{t-1}$ are used together with the frame-to-frame motion parameters $\mathbf{H}_{t,t-1}$ to initialize the parameters for the next frame t.

The accurate motion parameters are subsequently used in the background reconstruction algorithm to obtain camera-motion compensated input frames. The background-estimation algorithm generates one background image for the whole sequence, which is subsequently saved as background sprite and which will also be used for the subsequent segmentation step (see Chapter 7).

Clearly, the system can be simplified significantly if we know beforehand that the camera is static (as it is often the case for surveillance sequences). In this case, we can omit the camera-motion estimation steps and only keep the *SimMat* background synthesis algorithm.

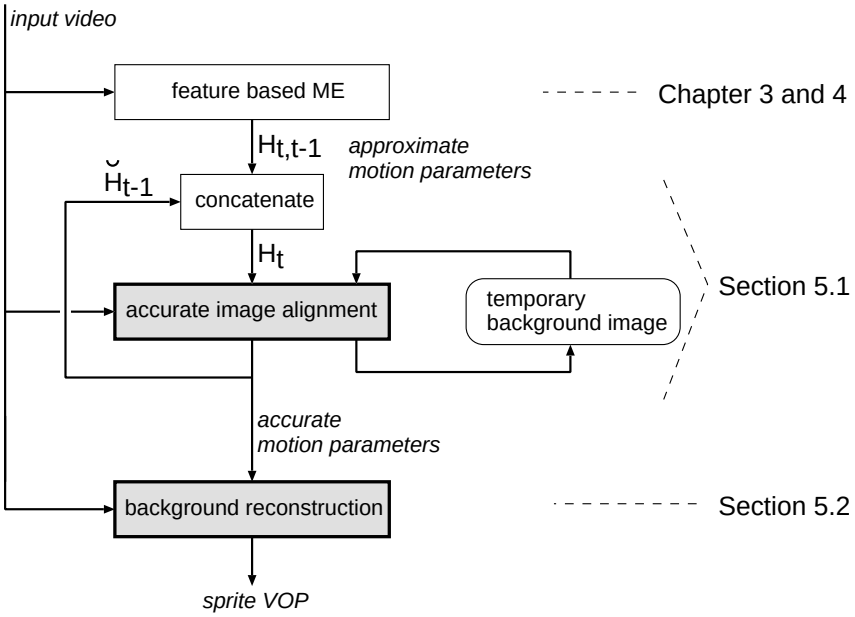


Figure 5.20: *Data-flow for motion parameter refinement and background estimation.*

(a) Typical input frame.

(b) Manual synthesiss

(c) Iterative update.

(d) Median.

(e) Mode.

(f) SimMat.

Figure 5.21: *Results for a very complex scene with many walking and standing people. Note that the background cannot be reconstructed completely, since some background regions are never visible in this sequence.*

(a) Input frame 150.

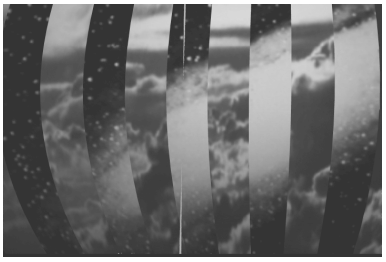
(b) Median, frame 1-150.

(c) SimMat, frame 1-150.

(d) SimMat, frame 1-300.

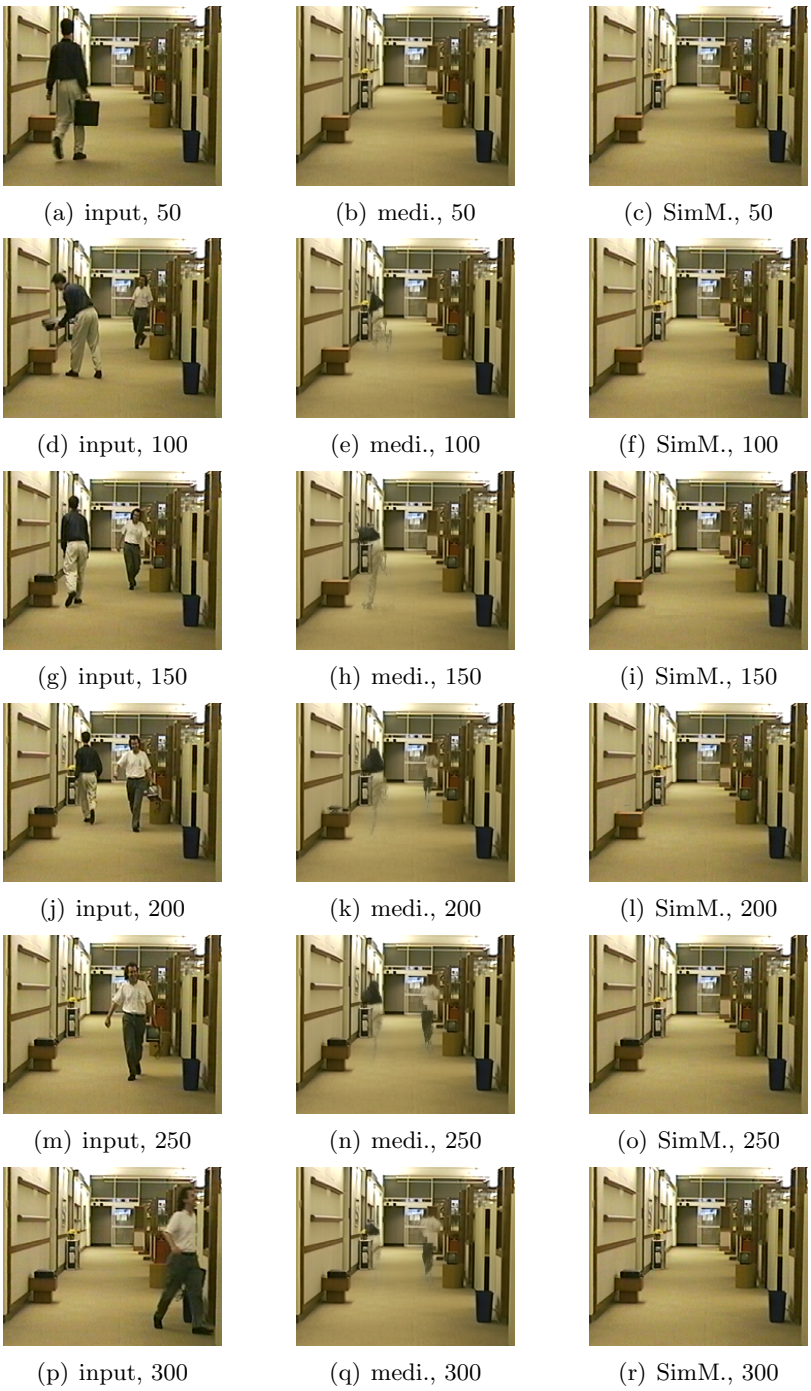
Figure 5.22: *Results for the well-known* road1 *test sequence. The Median algorithm over all 300 frames (not shown) has comparable quality as the SimMat algorithm in (d).*

(a) Typical input frame.



(b) SimMat result.

Figure 5.23: *Results for VQEG test sequence 17. Note that we have increased the background image brightness for clarity.*



(a) input, 50 (b) medi., 50 (c) SimM., 50

(d) input, 100 (e) medi., 100 (f) SimM., 100

(g) input, 150 (h) medi., 150 (i) SimM., 150

(j) input, 200 (k) medi., 200 (l) SimM., 200

(m) input, 250 (n) medi., 250 (o) SimM., 250

(p) input, 300 (q) medi., 300 (r) SimM., 300

Figure 5.24: *Results of online background generation for the* hall-and-monitor *sequence. History size is 150 frames.*